THESIS – SS142501

FIFTH GENERATION (5G) MOBILE NETWORKS: STUDY OF PERCOLATION THRESHOLD ON URBAN ROAD MODELS

NILA NOVITA GAFUR
06211450010019

SUPERVISORS
Dr. Elie Cali
Dr. Taoufik En-Najjary
Dr. Arnaud Guyader

MASTER PROGRAM
DEPARTMENT OF STATISTICS
FACULTY OF MATHEMATICS, COMPUTING AND DATA SCIENCE
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

Réseau Mobile 5G: Étude de Seuils de Percolation sur des Modèles de Voirie Urbaine

Confidentiel Orange

Réalisée par : **Nila Novita Gafur**
Responsable Entreprise : **Dr. Elie Cali**
Dr. Taoufik En-Najjary
Responsable Académique : **Dr. Arnaud Guyader**

Rapport du Stage
Master de Statistique M2
Université Pierre et Marie Curie
2016/2017

Fifth Generation (5G) Mobile Networks: Study of Percolation Threshold on Urban Road Models

In partial fulfillment of the requirements for the degree of Master of Sciences
In
Institut Teknologi Sepuluh Nopember

By :
NILA NOVITA GAFUR
NRP 06211450010019

Date : September 2017
Graduation Period : September 2018

Program Committee :
1. Dr. Elie Cali
2. Dr. Taoufik En-Najjary
3. Dr. Arnaud Guyader

Approved by :
Dean
Faculty of Mathematics, Computing and Data Science
Institut Teknologi Sepuluh Nopember

Prof. Dr. Basuki Widodo, M.Sc.
NIP.19650605 198903 1 002

I would like to dedicate this thesis to my beloved parents

FIFTH GENERATION (5G) MOBILE NETWORKS: STUDY OF PERCOLATION THRESHOLD ON URBAN ROAD MODELS

Student's name	: Nila Novita Gafur
Student's ID	: 06211450010019
First supervisor	: Dr. Elie Cali
Second supervisor	: Dr. Taoufik En-Najjary
Third supervisor	: Dr. Arnaud Guyader

ABSTRACT

Stochastic geometry is a mathematical discipline that combines geometry and probability. In particular, it model complex systems with a large number of elements distributed over a geographical area and has numerous applications in telecommunications. Based on stochastic geometry, mathematical models are designed to represent aspects of wireless networks. Talking about stochastic geometry models of wireless networks will not be detached from the important role of the continuum percolation. That is an extension of the percolation theory at R^2. It model a random network and analyse their behavior.

We apply these theories to our model "the connectivity of multi-hop D2D (Device-to-Device) networks" to predict some of their characteristics, such as to estimate minimum density of devices in a territory ensuring a long-distance communication called critical percolation threshold, to model percolation probability that a given devices is in the large connected component and to find the ratio of distance of two devices who want to communicate and the number of hops necessary to establish communication. We interpret the D2D communication refers to a random graph. Using Monte-Carlo simulation, we generate the data and propose some methods to get the best representation model for both urban and rural areas. We model the street systems as a Poisson-Voronoi tessellation and Poisson-Delaunay tessellation with varying street lengths.

Our results show that the estimated value of critical percolation threshold $\hat{\lambda}_C$ with selected method is almost same to the critical value λ_C of Poisson Boolean model (PBM) for Poisson-Voronoi tessellation (PVT) model and is quite different for Poisson-Delaunay tessellation (PDT) model, e.g for radius 0.225 km, $\hat{\lambda}_C$ PVT is 1.42 users/km of street, 1.51 users/km of street for PDT and 1.418 users/km of street for PBM. We notice also that PVT gives a very good representation for urban areas, meanwhile PDT is good for rural areas.

Keywords: Multi-Hop D2D Networks, Poisson Point Process, Percolation Threshold, Percolation Probability, Tessellation, and Random Graph.

Acknowledgements

In the name of Allah, the entirely merciful, the especially merciful. All praise and thanks are due to Allah, who has given the convenience and has guided the author to complete this thesis.

Firstly, I would like to begin by thanking Mr. Arnaud Guyader, the head of the statistics department in UPMC, who motivated me to work a lot and more seriously to study and also Mr. Michel Broniatowski for his wise advice during my Master. I also thank Ms. Lammart, the Statistics Secretariat, thank you for her help and generosity.

Second, I would like to express my gratitude to the entire OLN/NMP/MSA team at Orange Labs for their warm and friendly welcome throughout these six months. More specifically, I would like to sincerely thank my internship tutor, Mr. Elie Cali for his hospitality and his invaluable help, since the first day of my arrival at the company and to have given me all his confidence to work with him during all this period, not to mention his participation in the progress of this report and for Mr. Taoufik En Najjary as well.

Then, the more precious, I also thank my parents who have always supported me to go further, who always pray tirelessly for me, it is thanks to their love and affection that I was able to get to this point.

Finally, of course, I thank all my friends and especially Herfian Setiawan, Karen Alexandra Vásquez Vivas, David Garcia, and Kimsy Tor, thank you for being there in difficult times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background of the Study

Human needs related to technology of telecommunications increase year by year, especially, in terms of the speed of information diffusion, the increased capacity, the effectiveness and the profitability. That makes all sectors in the telecommunications field constantly evolve trying to provide better service, which leads to an increase in the speed of data transfer, expand the range of connectivity, but at a minimum cost. To meet those needs, a new generation of mobile network communication technology (the fifth generation: 5G) is being specified. It will bring many advanced features, including faster response times, a great capacity, a ubiquitous connectivity etc. Moreover, it concerns different applications with very heterogeneous needs. One of the technologies that will serve 5G is called device to device (D2D). The D2D feature allows direct communication between devices that are close to each other without necessarily going through the base stations. There are two types of D2D communication: (1) Direct communication between two devices, (2) Multi-hop communication. The D2D multi-hop communication is the communication that occurs between two devices which are distant using several nearby devices serving as relay. With the D2D multi-hop, a new operator could build a mobile network even if it does not have any base stations.

To anticipate this, the TEMS (The Effects of Marketization on Societies in Europe) research group in "*Modelling and Statistical Analysis*" (MSA) department, decided to study the D2D connectivity. It is important for Orange, a telecommunication company in France, to prepare for possibles consequences of this new feature.

In preparation for the arrival of this generation of mobile networks and in particular the technology of D2D, this research aims to model mathematically the D2D feature using stochastic geometric models that have been used for a long time [1], [2], [8], [9], [16] and

[17]. Stochastic geometry is a mathematical discipline combining geometry and probabilities. It particularly allows to model complex system with a large number of elements distributed over a geographical area and has many applications in telecommunications. In this research, these stochastic geometry models propose a representation of the morphology of street system and use the theory of percolation (specifically continuum percolation) in order to look for the value of some critical thresholds allowing to assure connectivity over long distances. The continuum percolation is an extension of the percolation theory on $\mathbb{R}^d$. It allows us to model random networks and to analyse their behavior, particularly the phase transition phenomenon [14]. These theories can be applied to modeling D2D networks to predict some of their characteristics.

1.2 Statement of the Problem

Based on the description above, the statement of the problem to be discussed in this study are as follows:

1. What is the minimum density of devices in the territory to ensure long-distance communications? $\rightarrow$ ***Percolation Threshold***
2. What is the probability that a given device is in the large connected component? $\rightarrow$ ***Fonction*** θ_0
3. What is the ratio between the distance of two users who want to communicate and the number of hops necessary to establish communication? $\rightarrow$ ***Fonction*** μ

1.3 Objectives of the Study

The purpose of this research is to answer the three questions bellow:

1. To know the minimum density of devices in the territory that ensure long-distance communications.
2. To know the probability that a given device is in the large connected component.
3. To know the minimum hops that it takes to establish communication between two devices.

1.4 Scope and Limitations

This research will be done by simulation using software R to know the characteristics of D2D communication and only focus on communication between mobile phone.

1.5 Significance of the Study

The finding of this study will help TEMS (The Effects of Marketization on Societies in Europe) organization and specifically for Orange company in their participation in future generation of mobile phone (5G). In addition, the company can also use it as a reference to find other related factors that need to be examined as this research is the first step in towards analysing D2D connectivity in multi-hop wireless networks. The most important is that the result of this research can help an Orange company in decision making about the need to build a BTS (Base Transceiver Station) in a certain region or not.

Chapter 2

Literature Review

2.1 Mathematical Model

In the first part, let us start by laying out some useful definitions, found in [12], [1] and [3].

2.1.1 Point Process

Point processes play an important role in stochastic geometry. These are random variables whose realization is a set of points in a certain space. Here, we will be in the context where the space is $\mathbb{R}^d$.

Let $\mathscr{S}$, the finite set or countable locally finite parts of $\mathbb{R}^d$: $\Phi \in \mathscr{S}$ if $\Phi \subset \mathbb{R}^d$ is at most countable and for any compact $K \subset \mathbb{R}^d$, $K \cap \Phi$ is finite. Each $\Phi \in \mathscr{S}$ defines a discrete measure $\varphi = \sum_{x \in \Phi} \delta_x$, where δ_x denotes the Dirac measure in x, for any Borel set A of $\mathbb{R}^d$,

$$\delta_x(A) = 1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases}.$$

It could then, in an equivalent way, work with all these measures, it will be noted $\mathbb{S}$. This set $\mathbb{S}$ suggests that the most important is to count the number of points in each Borel set, because if $\varphi \in \mathbb{S}$ and $A \subset \mathbb{R}^d$ is a Borel set of $\mathbb{R}^d$, the number of points of φ that belongs to A is $\varphi(A)$.

For each Borel set $A \subset \mathbb{R}^d$ we will note $N(A) : \mathbb{S} \to \mathbb{N}$, the application that at each φ associates $\varphi(A)$. The Borel algebra on $\mathbb{S}$ is the smallest σ-algebra such that $N(A)$ is measurable for any Borelian $A \subset \mathbb{R}^d$.

A random variable ϕ with values on $\mathbb{S}$ is a point process.

2.1.1.1 Poisson point process

Let μ be a measure fixed on $\mathbb{R}^d$. A Poisson point process with intensity μ is a point process ϕ such that for all Borel set A of $\mathbb{R}^d$, the random variable $N(A) \circ \phi$ follows a Poisson distribution with intensity $\mu(A)$ and for any sequence $A_1, ..., A_k$ of pairwise disjoint Borel, $N(A_1) \circ \phi, ..., N(A_k) \circ \phi$ are mutually independent.

Sometimes the intensity measure is given by:

$$\mu(A) = \int_A \lambda(x)dx \tag{2.1}$$

where $\lambda : \mathbb{R}^d \to \mathbb{R}_+$ is a measurable function called the region intensity of the Poisson point process [3]. If λ is a constant, the Poisson point process is called a homogeneous Poisson process or a stationary Poisson process. Its intensity becomes a real λ and:

$$\mu(A) = \lambda|A| \quad \text{where } |A| \text{ denotes the Lebesgue measure of A.} \tag{2.2}$$

Properties 2.1.1 (The homogeneous Poisson point process [4]) *The homogeneous Poisson point process ϕ with intensity $\lambda > 0$ has the following properties:*

1. *The number of points $N(A) = \#(\phi \cap A)$, contained in any region A is a Poisson random variable,*
2. *The expected number of points in A is $\boldsymbol{E}[N(A)] = \lambda|A|$,*
3. *If A_1,A_2 are two disjoint sets $N(A_1)$ et $N(A_2)$ are two independent random variables,*
4. *If $N(A) = n$, the n points are independent and uniformly distributed over A.*

2.1.1.2 Cox process

Now, considering a rich class of point process, a Cox process. The Cox process is a generalization of the Poisson process, where an intensity field is itself given by a random process and is independent of the underlying Poisson process [20].

2.1.2 Random Tessellations

A tessellation is the division of a surface into a pattern that covers the surface without overlap.

Definition 2.1.2 (Tessellation) *Let $\boldsymbol{S}$ be a connected region of $\mathbb{R}^d$ ($\boldsymbol{S} \subset \mathbb{R}^d$), called tessellation of $\boldsymbol{S}$ a countable collection $\{C_i\}$, where C_i are compacts of $\mathbb{R}^d$ and called cells such as:*

1. ***S*** *is the reunion of* C_i,
2. *The interior of each* C_i *is of dimension d,*
3. *The interiors of* C_i *are mutually disjoint,*
4. *Any compact of* $\mathbb{R}^d$ *is intersected by finite number of* C_i.

Definition 2.1.3 (Random tessellation) *If the* C_i *are randomly closed,* ***S*** *is a random tessellation.*

A random tessellation is stationary if its distribution is invariant by translation (same distribution) on $\mathbb{R}^d$ and isotropic if its distribution is invariant by rotation around the origin.

2.1.2.1 Poisson-Voronoi tessellation (PVT)

Let $\{x_i\} \in \mathbb{R}^d$ be a realization of a point process ϕ. It calls *Voronoi tessellation* of $\mathbb{R}^d$ with nuclei x_i, the tessellation with cells:

$$C_i = \{y \in \mathbb{R}^d : \|x_i - y\| \leq \|x_j - y\| \text{ for all } i \neq j\}, \tag{2.3}$$

where $\|.\|$ is the Euclidean norm.

The segments in a Voronoi tessellation correspond to all the points of the plane equidistant from the two nearest nuclei. If ϕ is a homogeneous Poisson point process with finite intensity and λ positive, the Voronoi tessellation is called *Poisson-Voronoi tessellation*.

2.1.2.2 Poisson-Delaunay tessellation (PDT)

Let $\{x_i\}$ be the nuclei of the Poisson-Voronoi tessellation, the Poisson-Delaunay tessellation is constructed by connecting all pairs of nuclei x_i that belong to neighboring cells. This tessellation is dual of the Poisson-Voronoi tessellation.

2.1.3 Random Graph

Some definitions about graphs that can be found in [13] [7].

Definition 2.1.4 (Network and graph) *A network is a set of interacting entities and a graph is a mathematical represenation of the network.*

Definition 2.1.5 (Graph) *A graph* $G = (V,E)$ *is composed of a set* $V = \{v_1, v_2 \ldots, v_n\}$ *of nodes and a set E of pairs of V xV which are called edges.*

When for all i, j the edge $e = \{v_i, v_j\} \in E$ is equal to the edge $e = \{v_j, v_i\} \in E$, this graph is called *an undirected graph*.

Definition 2.1.6 (Path) *A path is a sequence of nodes such that each node is connected to the next by an edge.*

Definition 2.1.7 (Chemical distance) *The length of the path is the number of edges visited to go from the first node of the path to the last one. The shortest length from node A to node B is called the chemical distance from A to B.*

Definition 2.1.8 (Random graph) *A random graph is a graph in which nodes and /or edges are generated by a random process.*

Definition 2.1.9 (Random geometric graph) *A random geometric graph is a graph such that the nodes are distributed randomly and an edge occurs if and only if the distance between two nodes is smaller than the radius r. It is a Gilbert graph.*

2.1.4 Continuum Percolation

Let ϕ be a homogeneous Poisson point process with intensity λ on $\mathbb{R}^2$. In each point of ϕ, we consider the sphere of radius ρ centered at this point, where ρ is a random variable identically distributed on each point of ϕ, such that the values taken by ρ at two distinct points are independent. This model is called the Boolean model. In this model, we can match to the Gilbert graph of radius 2ρ. Then, the connected component of the occupied region of the Boolean model is equal to the connected component of the Gilbert graph.

It is said that the graph percolated if there is an infinite connected component in this graph. The probability:

$$\Theta(\lambda) = P(\exists \text{ an infinite connected component in the graph }) \tag{2.4}$$

is called percolation function and the critical intensity of the graph:

$$\lambda_c = inf\{\lambda : \mathbb{P}(G(\phi) percolates) > 0\} \tag{2.5}$$

in other words for $\lambda < \lambda_c$ the system does not percolate and for $\lambda > \lambda_c$ the system percolates.

2.2 Mathematical Model of Mobile Network D2D

The D2D mobile network discussed in the introduction is a complex system. It is impossible to treat this complex system without describing the interactions between their components.

This description will be made through mathematical models. The purpose of this part is to present these models.

The different components that we had to model were: The territory's street, Users of mobile devices located on the street, and Communications between mobiles. A model development requires clearly the initial hypothesis. To obtain an efficient and a simulable model, we introduced some simplifications associated to the reality. Therefore, the following hypotheses are made:

1. The users are located on a limited territory, urban or rural,
2. There is only one type of users, mobile device,
3. The users are on the streets and immobile,
4. Neither buildings nor interferences are taken into account,
5. The radius communication of a mobile is constant,

2.2.1 Orange Street System Models : PVT, PDT

If the impasses are eliminated, the territory road constitute a certain tessellation of this territory. To model roads, Orange has developed a number of models based on random tessellations. If a good model is chosen for a given territory, the morphological characteristics of the roadways (number of intersections, road segments, house blocks or kilometers of roads) are found by calculating the average of the corresponding characteristics (number of nodes, edges, faces, or total length of edges) of the random tessellation.

One of the simplest models of point process is the Poisson point process which has a single parameter. In particular, Orange uses two tessellation models based on Poisson processes: Poisson-Voronoi Tessellation (PVT), Poisson-Delaunay Tessellation (PDT).

2.2.2 Users Model on the Streets

Having modeled the streets, the location of the users on the streets constitutes a set of random points on these streets. So we can model it through a point process on $\mathbb{R}^2$. The position of the users can be modeled as the realization of a linear Poisson process (on the streets, in other words on the edges of the previous random tessellation) because the users are independent of each other and are uniformly distributed at random on the territory street.

We have modeled the roads and users using Poisson processes. Hence, the user model is a double stochastic Poisson process. This type of point process is called Cox process.

2.2.3 Graph Communication Network

The D2D mobile network is formed by the possibility of communication between users. The principle of D2D is that users communicate directly with each other without going through base station. These links form a graph whose nodes are the users and an edge exists between two users when they can communicate with each other. The interactions are reciprocal and the links between the users are undirected.

A mobile can communicate directly with another mobile if the signal-to-noise ratio is above a certain threshold. As a first approximation (except in particular power adaptation), this corresponds to a constant maximum distance between the two users. We will then plot an edge of the graph if the distance between the two nodes is less than a given distance r. As a consequence, in our modeling, at any moment, the mobile network corresponds exactly to a Gilbert graph.

All devices on the territory constitutes a network that can route communications. Thus, if a user A wants to communicate with another user B on the territory, another users will be used as relays to route his communications. Communication will be possible immediately after there is a related path from user A to user B.

The objective of this study is to determine to which conditions the network formed (and the Gilbert graph corresponding) ensures the connection of a large part of the users on the territory. The answer is provided by the theory of continuum percolation [14] : below the percolation threshold, long-distance communications are not possible, beyond that, there is an infinite connected component and we can give indications the percentage of connected user pairs.

Chapter 3

Methodology

In this section will be described the steps of research that will be carried out during the study. These steps can be seen in Figure 3.1. In that Figure, the phase is divided into three phases that are identification and introduction phase, simulation phase and evaluation phase.

3.1 Identification and introduction phase

It is organized by conducting a preliminary study, determining the issues to be discussed and looking for literature sources related to the research and meeting with the various parties involved in the project.

3.2 Simulation phase

After determining the parameters necessary for modeling system D2D mobile network and its characteristic, we consider that the characteristics of D2D communication can be represented by using simulation technique where to generate the data, we will use Monte Carlo simulation. Simulation is done because it is a simple way to facilitate research by not spending costs and time-consuming when compared with real experiments but it still can give accurate results. In using simulation, it is important to pay attention to the hardware and the software that will be used. Therefore, this will affect the time and the results of research. There are many programming languages that exist, but in this study, researchers will chose **R** Software version 3.4.1 and use Windows 7 enterprise. It is a Windows operating system for business.

3.3 Evaluation phase

This is the most important point when doing research, especially in performing simulations. The statistical method will be used in this study.

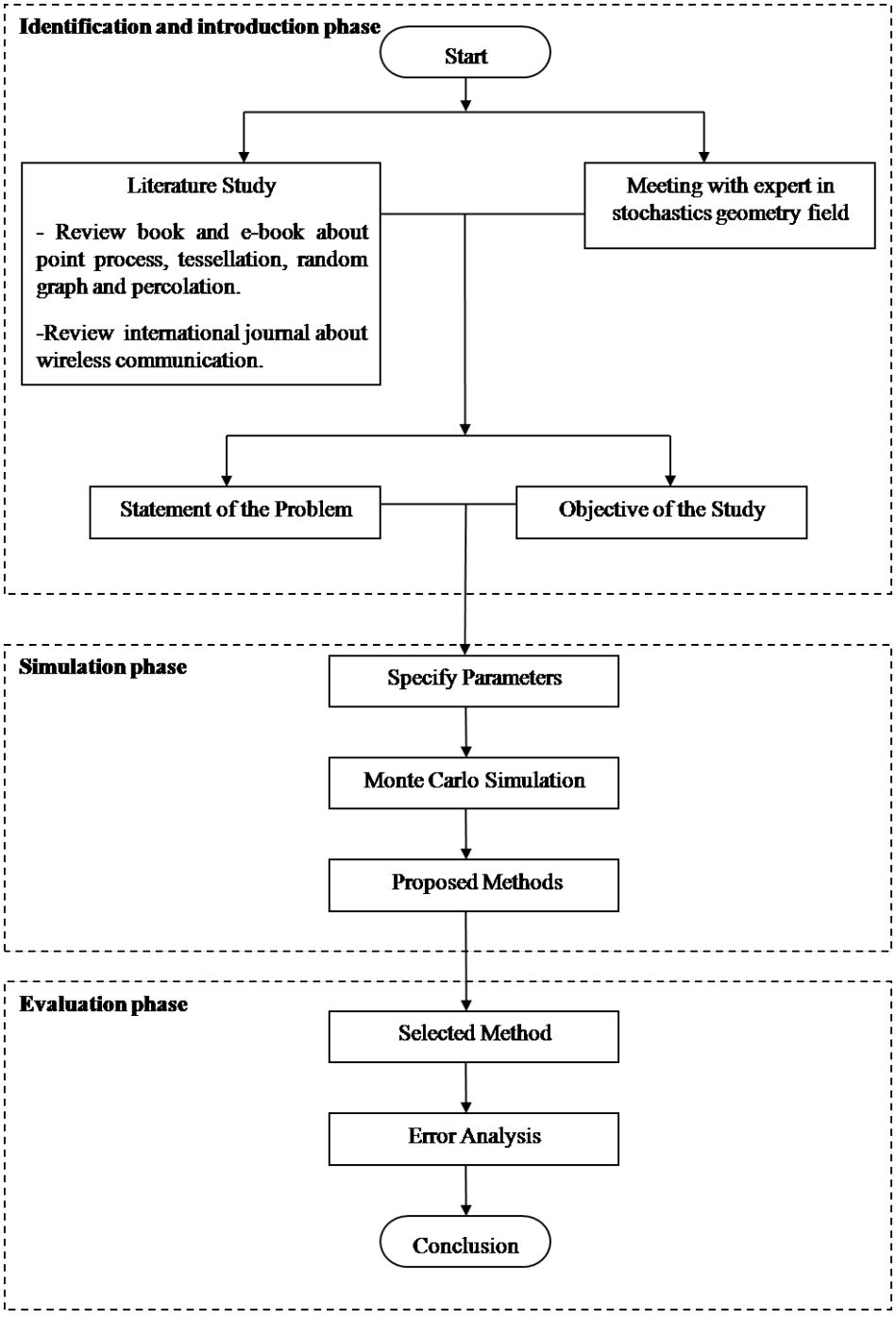


Fig. 3.1 flowchart of methodology research

Chapter 4

Results

4.1 Analytical Methods

In this part, using mathematical models, we simulate random graph to estimate critical percolation threshold ($\widehat{\lambda_c}$), percolation probabilities by origin device (θ_0) function), and stretch factor (μ function).

4.1.1 The First Type of Road: Poisson-Voronoi Tessellation (PVT)

4.1.1.1 Critical Percolation Threshold

In this study, the mobile network D2D is represented by Gilbert random graph. The objective is to find the percolation threshold in the graph and that will be studied by simulation.

In theory, the random network considered, naturally lives on an infinite graph while in simulation, only a finite graphs can be simulated. Consequently, when doing the simulation, a new method should be found which allows to find the percolation threshold in a finite graph. This method should fulfill the following criteria: it must be fast in execution time and be fair, that is, it gives the best estimate value of λ_c.

Obviously, an error would be obtained because of the simulation and the finite graph. Preferably this error should be as small as possible. Hence, we propose to try several methods and to compare them to find the best one according to the criteria and the errors. In order to obtain a good approximation of the percolation threshold value for the urban road network, the Monte Carlo method is used by simulating graphs and searching for a potentially infinite connected component.

Methods proposed

First, we will determine a sufficiently large square. We observe a region of the city of finite size represented by a square window s side. Unless otherwise indicated, the window size would be $s = 5$, that is 5 km by 5 km makes 25 km^2. We begin by simulating Poisson point process with intensity ζ in this window, where ζ is the expected number of points in a window of a unit area. For example, if we want to simulate Poisson point process on a 1 x 1 window and if we set the intensity of the Poisson point process $\zeta = 100$, then there will be an average of 100 points in the window as shown in Figure 4.1. In our case, we consider a size of 25 km^2. The number of points in the window is on average 100 x 25 (see Figure 4.2). From this, we draw the Poisson-Voronoi tessellation associated with these points. The edge of this tessellation will be the streets of the city.

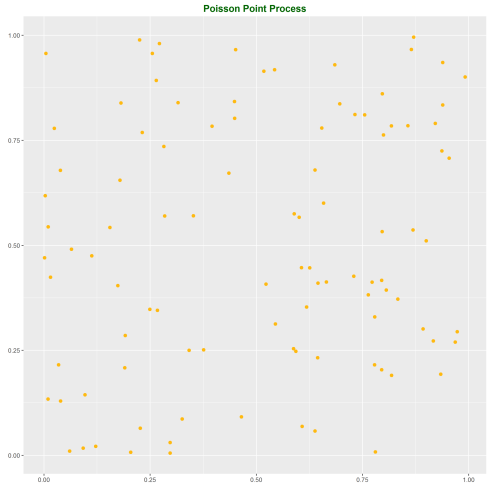


Fig. 4.1 Poisson process on a surface of 1 km^2

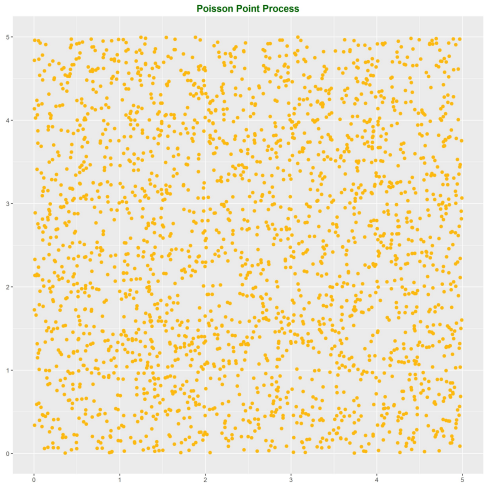


Fig. 4.2 Poisson process on a surface of 25 km^2

We measure the total length of streets. We will call γ the average length of streets per km^2. For the Poisson-Voronoi tessellation, we can show that $\gamma = 2\sqrt{\zeta}$ [19]. As a result, for $\zeta = 100$, γ is 20 km/km^2.

Then we will put some points on each street. In other words, users are placed on the streets. For each street, we place the points that follow the distribution of linear Poisson point process with intensity λ. Since random tessellation is obtained by Poisson point process, then each street segment is independent of the others, so that users can be distributed to each street segment independently. More precisely, the product $\lambda\gamma$ describes the spatial intensity of users, in other words the number of users per unit area. This realization can be seen in

Figure 4.5 for an intensity $\lambda = 0.79$ ($\lambda = 0.79$ means that there are 0.79 users per *km* of street).

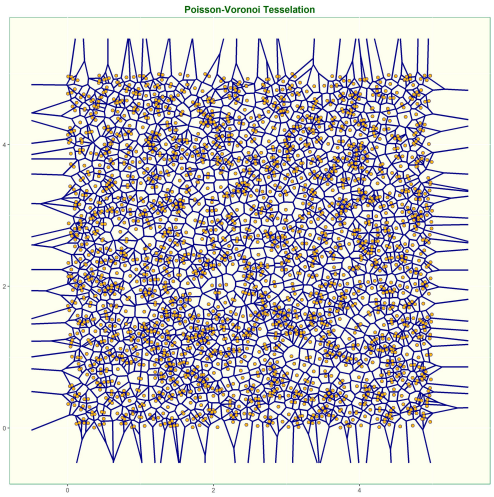


Fig. 4.3 Poisson-Voronoi Tessellation

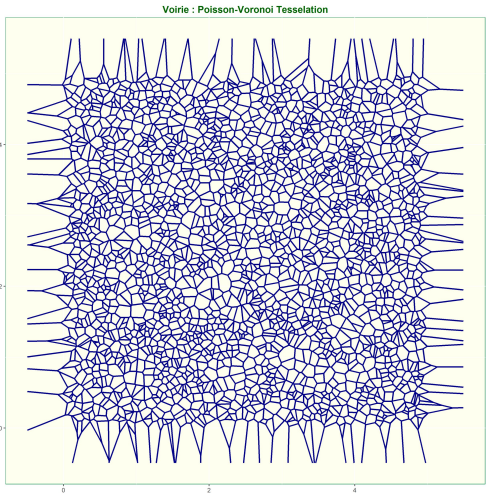


Fig. 4.4 Simulated roads

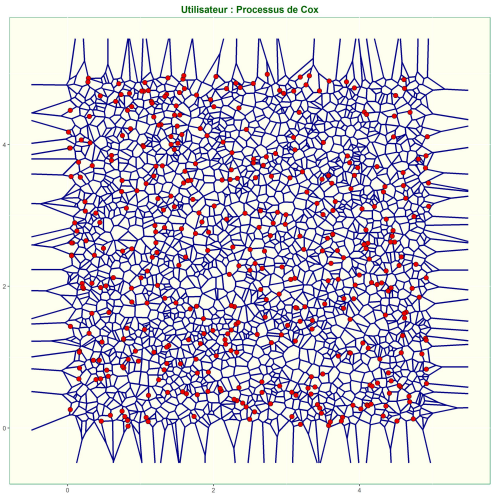


Fig. 4.5 Users on the roads by using Cox process

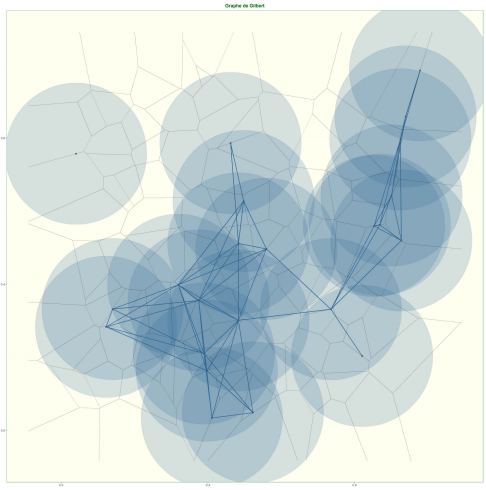


Fig. 4.6 Gilbert graph

After distributing the users on the streets, a Gilbert graph is created as a realization of the D2D mobile network. This is done by placing a disc on each point (users) with a fixed radius. An edge of the graph is drawn between two vertices once the Euclidean distance between these vertices is less than this radius.

$$d(A,B) = \|A - B\| \leq r \tag{4.1}$$

where $\|.\|$ is the Euclidean norm and r is the radius which is fixed at $300m$. It's within reach of a mobile to be able to communicate. This means that two users A and B are connected by an edge if and only if A is contained in the disc of radius equal to $300m$ centered in B. As a consequence, we have a graph driven by three parameters: the Poisson-Voronoi tessellation γ, the users λ, and the radius r. This graph is then a Gilbert graph with the parameters (γ, λ, r) illustrated in Figure 4.6. Besides, if A and B are in the same connected component of the graph, they can communicate by a finite number of jumps (hops).

We tested three methods to find $\widehat{\lambda_c}$.

Method 1: Crossing Window

In this method, we try to determine if there is a connected component that crosses the observation window. For γ and r given, the function $\Theta(\lambda)$ is either 0 or 1 in an infinite graph (see Formula 2.4). However, we have finite graph $G_f(\gamma, \lambda, r)$, we define the function:

$$\theta_f(\gamma, \lambda, r) = P(\exists \text{ a connected component that crosses the window } s \text{ in } G_f(\gamma, \lambda, r)) \quad (4.2)$$

Algorithm 1: Percolation Algorithm

Input: Graph
Output: State of percolation
create a small window s
1. The flow function
$Z_1 := \{\text{points beyond the upper limit of s}\}$
for $(x_i, y_j) \in Z_1$ **do**
    search all connected points with points in Z_1 ;
    puts the attribute 1 at the connected points ;
end
2. The percolation function
$Z_2 := \{\text{points below the lower limit of s}\}$
for $(x_i, y_j) \in Z_2$ **do**
    if *a point has the attribute* 1 *in* Z_2 **then**
        percolate ;
    else
        does not percolate ;
    end
end

where $0 \leq \theta_f(\lambda) \leq 1$ for γ and r given and the curve function $\theta_f(\lambda)$ is represented in the Figure 4.9. We then know by definition θ_f and Θ that:

$$\lim_{s \to \infty} \theta_f(\lambda) = \Theta(\lambda) \tag{4.3}$$

To determine if there is a connected component that crosses the window, we consider the points that are in a bandwidth $r/2$ at the top or bottom of the window [18]. In other words, we consider the set Z_1 of the coordinates points (x,y) such that $y \geq 5 - r/2$ and the set of the Z_2 of coordinates points (x,y) such that $y \leq r/2$. After we search if there is a vertical path (see definition 2.1.6) from a point of Z_1 to a point of Z_2 (see Algorithm 1).

Since it is a simulation of a random process, the result can vary from one simulation to another as can be seen in the Figures (4.8) and (4.7), the system percolate and the system does not percolate for the same value $\lambda = 0.79$.

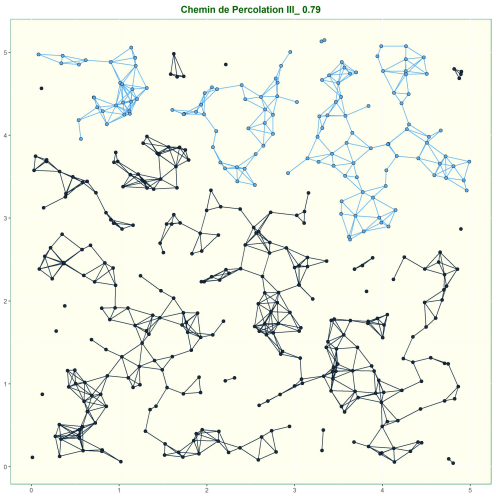


Fig. 4.7 Non percolated

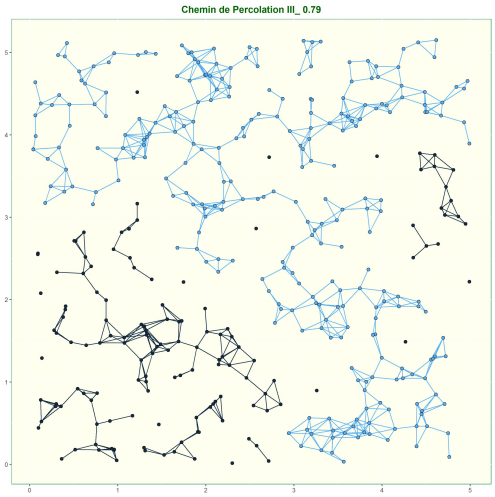


Fig. 4.8 Percolated

For each value λ, a number of Gilbert graphs are randomly generated by running algorithm 1 many times and counting the percentage of percolation times is done. If the experiment is repeated a lot of times and the average is calculated: If we note y_i the results of each simulation, that could be written:

$$\frac{y_1 + y_2 + \cdots + y_n}{N} = \bar{y} \tag{4.4}$$

where y_i be 1 if in simulation i, the finite graph percolate and 0 otherwise and N is the number of simulations. We search $\bar{y}$ for different values of λ. Then, when we trace $\bar{y}$ in function of λ, this defines the function $\theta_f(\lambda)$. This function seems like a logistic function

curve. Since we have data that are binary random variables, we will estimate the percolation threshold using the logistic model. Recall the logistic model: y is a dependent variable and x is an explanatory variable. We denote p the probability of the dependent variable equal to 1 (percolate): $p = \mathbb{P}(y = 1|x)$. Then:

$$logit(p) = ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_i \tag{4.5}$$

Here, the x_i are the values of λ and the y_i are the results of simulations contained in $\{0,1\}$. We then consider that the value p is a good approximation for $\theta_f(\lambda)$. Since the curve $\theta_f(\lambda)$ is approximately symmetrical, we assume that we have percolation for $\theta_f(\widehat{\lambda_c}) \approx 0.5$, that is to say that the critical value is at this probability. Therefore, replacing p by 0.5, we found that:

$$\widehat{\lambda_c} \approx \frac{-\beta_0}{\beta_1}$$

In consequence, we find $\widehat{\lambda_c}$ looking at the value of λ corresponding to $p = 0.5$.

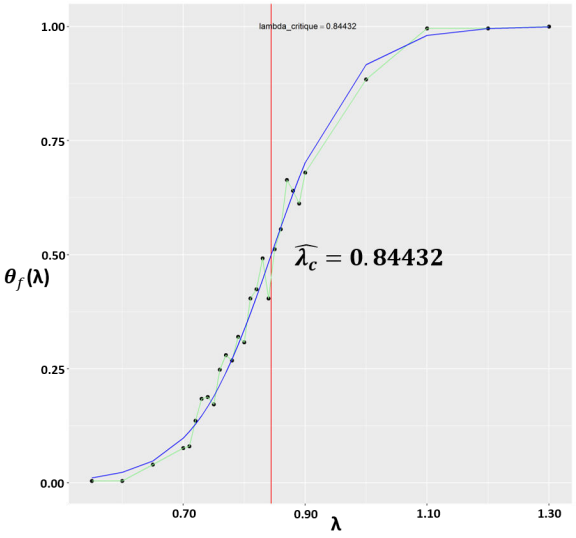


Fig. 4.9 $\widehat{\lambda_c}$ method 1: First algorithm with 250 iterations

To test the method in the simulations, we put in constant the parameters of the Gilbert graph γ and r at $20\,km/km^2$ and $0.3\,km$ respectively and we vary λ. As the road network is dense, it is known from [5] that λ_c for the standard Boolean model is

$$\lambda_c \approx \frac{4.51}{\pi\gamma r^2} \tag{4.6}$$

Here, it gives $\lambda_c \approx 0.8\,km^{-1}$. It is thought that the $\widehat{\lambda_c}$ value obtained by simulation will be close because the road for $\gamma = 20\,km/km^2$ is quite dense. For that reason, we will vary λ from $0.5\,km^{-1}$ à $1.3\,km^{-1}$. Figure 4.9 shows the estimated value of the percolation threshold $\widehat{\lambda_c}$. It expresses $\theta_f(\lambda)$ as a function of λ in $G_f(\gamma,\lambda,r) = G_f(20,\lambda,0.3)$ where $0.5 \leq \lambda \leq 1.3$ et $N = 250$.

The function $\theta_f(\lambda)$ obtained by simulation is represented in green, while the function $\theta_f(\lambda)$ obtained by the logistic model is represented in blue. This graph clearly shows that the logistic model is a good approximation to find the percolation threshold accurately. Note that when N (the number of simulations) is larger, the function $\theta_f(\lambda)$ obtained by the simulation coincides better with the function $\theta_f(\lambda)$ obtained by the logistic model. In addition, the value $\theta_f(\lambda)$ increases when λ increases, and if λ is superior than 1.1, the probability is equal to one, which means that the model always percolate for $\lambda \geq 1.1$. Conversely, for $\lambda \leq 0.5$ the probability is zero, the model never percolate. Therefore, the percolation threshold will be in the intervalle $0.5 \leq \lambda \leq 1.1$.

Algorithm 2: Percolation Algorithm

Input: Graph
Output: State of percolation
create a small window s
1. The flow function
$Z_3 = \{$points lying on the left side of s$\}$
for $(x_i, y_j) \in Z_3$ **do**
    search all connected points with points in Z_3 ;
    puts the attribute 1 at the connected points ;
end
2. The percolation function
$Z_4 = \{$points lying on the right side of s$\}$
for $(x_i, y_j) \in Z_4$ **do**
    if *a point has the attribute* 1 *in* Z_4 **then**
        percolate ;
    else
        does not percolate ;
    end
end

On the other hand, we can also have percolation with a horizontal path. In the same way, with the same parameters as the previous algorithm, we examine the position of points to the left or right of the window. We consider the set Z_3 of the coordinate points (x,y) such

that $x \leq r/2$ and the set Z_4 of the coordinate points (x,y) such that $x \geq 5 - r/2$. After that we search if there is a horizontal path from a point of Z_3 to a point of Z_4 (see Algorithm 2).

We will combine these two algorithms and look at the percolation of vertical path or of horizontal path simultaneously. We suppose that we will have a better estimate of the critical threshold 3).

Algorithm 3: Percolation Algorithm

Input: Graph
Output: State of percolation
create a small window s
$Z_1 = \{$points beyond the upper limit of s$\}$
$Z_2 = \{$points below the upper limit of s$\}$
$Z_3 = \{$points lying on the left side of s$\}$
$Z_4 = \{$points lying on the right side of s$\}$
1. The flow function
for $(x_i, y_j) \in Z_1$ **do**
    search all connected points with points in Z_1;
    puts the attribute 1 at the connected points ;
end
2. The percolation function
for $(x_i, y_j) \in Z_2$ **do**
    if *a point has the attribute* 1 *in* Z_2 **then**
        percolate ;
    else
        does not percolate ;
    end
end

if *does not percolate* **then**
    1. The flow function
    for $(x_i, y_j) \in Z_3$ **do**
        search all connected points with points in Z_3;
        puts the attribute 1 at the connected points ;
    end
    2. The percolation function
    for $(x_i, y_j) \in Z_4$ **do**
        if *a point has the attribute* 1 *in* Z_4 **then**
            percolate ;
        else
            does not percolate ;
        end
    end
end

Method 2: Torus

As in the previous method, we carry out the three algorithms (see the algorithms [1, 2, 3]) to find $\widehat{\lambda_c}$. On the other hand, in this method, we position the road network on a torus, which makes it possible to simulate an infinite connected component.

In the first algorithm, before the roadway is drawn, the set of coordinate points (x,y) such that $y \leq r/2$ are copied beyond the upper limit of s where now all ordinates y for each copied point are larger than s. After that, we can trace the tessellation from these points. The points (users) are then placed on all the edges of the tessellation, except the edges of the points copied. We copy again all the points (users) of coordinates (x,y) such that $y \leq r/2$ on the edges which lie beyond the upper limit of s. If we wind this graph of the tessellation with user points, it's a cylinder.

Then in the second algorithm, we do the same for the set of coordinate points (x,y) such that $x \leq r/2$ which are copied beyond the upper limit of s where all the x-coordinates for each copied point are larger than s. We proceed in the same way as with the first algorithm to put users on the street. We do the same for the third algorithm that combines the two previous algorithms. By this algorithm, we obtain a torus. we only consider the components that crosses the window by passing through both a user and his copy. At the end, we can search for $\widehat{\lambda_c}$ in the same way as in method 1.

Method 3: Cluster

From the point of view of graph theory, the infinite connected component can be expressed as the largest cluster of the graph. Hence, by this method, we estimate λ_c by looking at the largest cluster and the second largest. When λ reaches the critical value λ_c, the largest clusters of the graph begin to merge to form the infinite connected component. Thus, in a finite window, the size of the second largest cluster begins to decrease. The second cluster merging with the larger one is the third, smaller one becomes second, and so on.

The algorithm of this method allows to define the largest cluster and the second larger by counting the number of points in each cluster. Then we look for the percentages and we draw the curves of the two clusters illustrated in Figure 4.10. We also look for the maximum of the second cluster. The value $\widehat{\lambda_c}$ is obtained by the maximum of the curve of the second cluster associated with the value of λ [11].

This figure emphasizes that the cluster size increases progressively as λ increases. It is easy to see by looking at the first curve.

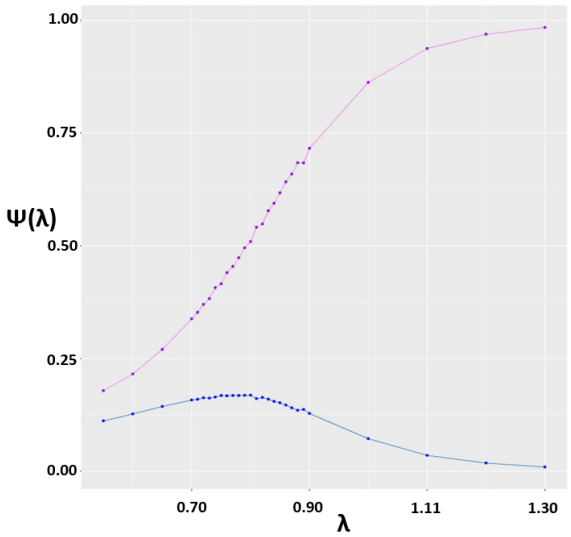


Fig. 4.10 Cluster Method: ($\Psi(\lambda)$ = percentage of points in the cluster)

Comparison of the three methods

In order to compare these three methods, we look at the values $\widehat{\lambda_c}$ obtained by each method using the logistic model for $p = 0.5$, as well as the number of simulations and the simulation time. These values $\widehat{\lambda_c}$ are given in the following Tables:

Table 4.1 $\widehat{\lambda_c}$: The First Two Methods

The number of simulations	**Method 1**			**Method 2**		
	Alg. 1	Alg. 2	Alg. 3	Alg. 1	Alg. 2	Alg. 3
50	0.813	0.825	0,787	0.818	0.830	0.777
100	0.820	0.835	0,767	0.836	0.843	0.779
150	0.816	0.827	0,772	0.836	0.834	0.775
200	0.820	0.833	0,803	0.838	0.836	0.770

It is found that the critical values obtained by the three methods are not very dissimilar. For the first two methods, the $\widehat{\lambda_c}$ values obtained by the first algorithm and second algorithm are almost identical. Which is logical because the observation window is symmetrical. Furthermore, the values of $\widehat{\lambda_c}$ are stable even with different numbers of simulations. Besides, we find that the estimated percolation threshold decreases when we combine the two algorithms (see the results of the third algorithm). On the other hand, by observing the execution time in simulation with **R** software of each algorithm of all the methods, these two methods are very long whereas the cluster method is very fast to make simulations. It is clear in the table that

the cluster method can achieve simulations up to $N = 2000$, we vary N, even for large values of N, we find that the cluster method $\widehat{\lambda_c}$ is unstable. This is due to the fact that the maximum value of the curve of the second cluster is difficult to determine because the interval of the maximum value is large especially when N the simulation number becomes large, this curve becomes very flat, consequently to specify the value $\widehat{\lambda_c}$ is very difficult.

Table 4.2 $\widehat{\lambda_c}$: Cluster Method

The number of simulations	**Cluster**
500	0.77
1000	0.75
1500	0.78
2000	0.80

In conclusion, among the three methods, method 2 with the third algorithm (combination of the two algorithms) is chosen. This is the best method for estimating $\widehat{\lambda_c}$ because there is an infinite path in the graph and the estimate value of $\widehat{\lambda_c}$ is stable for different values of N.

4.1.1.2 Percolation Probabilities by Origin Point (θ_0 Function)

After estimating the percolation threshold, we want to determine the probability that a given node of the graph (which we choose at random as the origin point of $\mathbb{R}^2$) belongs to the infinite connected component. We note this function θ_0. It also depends on the parameters (γ, λ, r). Thus for γ and r given, the function θ_0 becomes $\theta_0(\lambda)$. We write [10]:

$$\theta_0(\lambda) = \mathbb{P}(\text{Percolation of the origin point}) \tag{4.7}$$

To estimate this probability, we will use the following method: we simulate a Poisson-Voronoi tessellation. At first, we create a single user on the edges of the tessellation according to a uniform distribution. Then, a new user is added on the road. We will also add users until we get the percolation in the window. We will use the *union-find* algorithm in this method to identify the connected components of the graph.

The *union-find* algorithm works very differently from the algorithms of the previous methods. It is an algorithm that will perform two crucial operations:

1. *Find* : It is used to determine if two points belong to the same connected component related.

2. *Union* : Gathers two points (or connected components) into a single connected component.

Algorithm 4: Union-Find Algorithm

Input: Poisson-Voronoi tessellation

Output: Percolation Graph

$p(.)$: parent

$r(.)$: rank

add a point on the Poisson-Voronoi tessellation

1. MakeSet Function

(*x : the point that is added*)

$p(x) :=$ x ;

$r(x) := 1$;

2. Find Algorithm

$i :=$ x

while $p(i) \neq i$ **do**

  $i := p(i)$;

  $p(x) := i$;

end

return $p(x)$;

3. Union Algorithm

merge two points i and j

search $p(i)$ et $p(j)$ by using *Find* algorithm

if $p(i) = p(j)$ **then**

  break ;

end

else

  if $r(i) < r(j)$ **then**

    $p(i) := j$;

  end

  if $r(i) > r(j)$ **then**

    $p(j) := i$;

  end

  if $r(i) = r(j)$ **then**

    $p(j) := i$;

    $r(i) := r(i) + 1$;

  end

end

In each of these simulations, one point is added at a time, stopping as soon as the graph crosses the window. The connected components are followed at each step using the *union-find* algorithm [15]. Ideally, in this algorithm, each connected component is represented only by one of its members, called "*parent*" and has a weight called "*rank*". For each new point added, the parent of that point is equal to itself, while the rank is set to one. We associate at this point all the points which are at a distance less than r (see Equation (4.1)). During the union of two points (i) and (j), we look for the parents of (i) and (j). If they have the same parent, they are in the same connected component. Otherwise we associate them with the parent with the highest *rank*. On the other hand, if the two *ranks* are equal, one of them is declared as the parent of the other and one is added to his *rank*. The *union-find* algorithm is shown in the Algorithm 4. To find the probability θ_0, it follows the following reasoning (see Appendix 2).

Recall that we have two sources of randomness. These are the T roads and devices (the number of users J and the position of users on the road). We know that T has a complicated distribution with the average street length $\nu_1(T)$ equal to γ per unit area, and for T given, J follows a Poisson distribution of parameter $\lambda\nu_1(T)$ where λ is the average of the number users per *km* of street. Using the definition of discrete conditional probability, we have:

$$\begin{aligned}\theta_0(\gamma,\lambda,r) &= \mathbb{P}(\text{Percolation of the origin point}) \\ &= \sum_{j,s} \mathbb{P}(\text{Percolation of the origin point}\,|\,J=j,T=t)\mathbb{P}(J=j\,|\,T=t)\mathbb{P}(T=t)\end{aligned} \tag{4.8}$$

We will estimate the three probabilities. We simulate a tessellation t. For each simulation, we draw at random the origin point and we add one by one points drawn uniformly on the streets until the connected component passing the origin crosses the window. As a result, we then have percolation the origin point for $J=j$ and $T=t$, that is:

$$\mathbb{P}(\text{Percolation of the origin point}\,|\,J=j,T=t)=1$$

then, for a given tessellation t, users follow a Poisson distribution, we have:

$$\mathbb{P}(J=j\,|\,T=t)=e^{-\lambda\nu_1(t)}\frac{\lambda\nu_1(t)^j}{j!}$$

and the last probability is a probability to generate roads or tessellation. We simulate some tessellations $t_1,\ldots,t_m$, where m is the number of tessellations. These tessellations are

generated randomly and independently of each other. As a result, the probability of the road generated:

$$\mathbb{P}(T = t) = \frac{1}{m}$$

The algorithm is described below:

Algorithm 5: θ_0 Function Algorithm

Input: Roads

Output: Percolation Graph

n := the number of simulations

m := the number of Poisson-Voronoi tessellation

1. generate a Poisson-Voronoi graph t, $i \leftarrow 0$,
2. $j \leftarrow 0$,
3. place a point uniformly at random somewhere on one of the streets of t, $j \leftarrow j+1$,
4. use the *union-find* algorithm to check the percolation that goes through the origin,
   (a) if the graph does not percolate, we add another point uniformly randomly somewhere and we continue until it is percolated,
   (b) if the graph percolated, we note the number of points added and we calculate the probabilities $\theta_0(\lambda)$ by the following formula,

$$X_{i,t} \leftarrow \Sigma_{j' \geq j}^{\infty} \mathbb{P}(J = j' | T = t) = \Sigma_{j' \geq j}^{\infty} e^{-\lambda \nu_1(t)} \frac{\lambda \nu_1(t)^{j'}}{j'!} \tag{4.9}$$

   where $\nu(t)$ is the road length of t in the window s.

The algorithm (5) is simulated for several realizations of t. For one t, if we generate n simulations, we have $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$, we can take the mean of $\overline{X_t}$. As a consequence, the estimate of the probability θ_0 for γ and r given is:

$$\theta_0(\lambda) \approx \frac{1}{m} \sum_{k=1}^{m} \overline{X_k} \tag{4.10}$$

and its variance:

$$\sigma^2 := \frac{1}{m} \left(\frac{1}{m} \sum_{k=1}^{m} \overline{X_k^2} - \left(\sum_{k=1}^{m} \overline{X_k} \right)^2 \right) \tag{4.11}$$

To calculate Formula (4.9), we approach it by the normal distribution. Recall that if:

$$X \sim \mathscr{P}(\lambda)$$

then, for λ sufficiently large, we have:

$$Y \sim \mathscr{N}(\mu = \lambda, \sigma^2 = \lambda)$$

Since X is a discrete variable and Y is a continuous variable, we must apply the continuity correction [6]. For example in this case, if $j' = x_0$ is a non-negative integer,

$$\begin{aligned} \mathbb{P}_X(X \geq x_0) &\approx \mathbb{P}_Y(Y > x_0 - 0.5) \\ &\approx \mathbb{P}\left(Z > \frac{(x_0 - 0.5) - \lambda}{\sqrt{\lambda}}\right) \end{aligned} \tag{4.12}$$

where Z follows a standard normal distribution and with the software **R**, the function *pnorm* makes it possible to calculate this probability.

Method 4: Torus + Union-Find for $\widehat{\lambda_c}$

The problem of the method we have chosen to estimate the critical threshold is that the execution time of the algorithm is very long. After having simulated several times the function θ_0 by the *union-find* algorithm, we note that this algorithm allows us to reduce the execution time to find the connected component of the graph. Therefore, we also tried to apply this algorithm to find the critical threshold.

The idea of this method is the same as method two (torus). However, we look for percolation using the *union-find* algorithm. In contrary at the algorithm for the function θ_0 which adds the points one by one, here, for each simulation, the points are already known and the Gilbert graph already plotted. We will look at the points one by one, and apply the method of k-nearest neighbors. This method is based on the minimum distance from one point to the others to determine the nearest k points. This avoids the calculation of all the distances between the points. The algorithm follows the following steps:

1. After placing users on the streets (as in Method 2: Torus), determine the k nearest neighbors of each user. Here, we choose $k = 10$,
2. Calculate the distance from each point to the neighboring points only,
3. Sort the points from the smallest to the largest distance,

4. Choose points that have a distance smaller than r (see Formula 4.1),

5. Use the *Union-Find* algorithm (see algorithm 4). *Find* to find two points that have a distance smaller than r (connected), and *union* to join them or put them in the same connected component,

6. To check percolation, use the *union-find* algorithm again. Since we choose the third algorithm to look at the percolation path, so we look if the parents of the points at the top and the bottom or the points to the left and the right are the same. The graph percolate if they have the same parent and it does not percolate otherwise.

Then, we estimate λ_c by the logistic model. We note that in this method, the execution time is much faster, so we can increase the size of the observation window. In addition, we see that if the size of the window is large, we will have a more vertical curve which ensures that we can have a better estimated value of λ_c. We can see in Figure 4.11 that the curve becomes more vertical for $s = 30$ than the curve for $s = 5$.

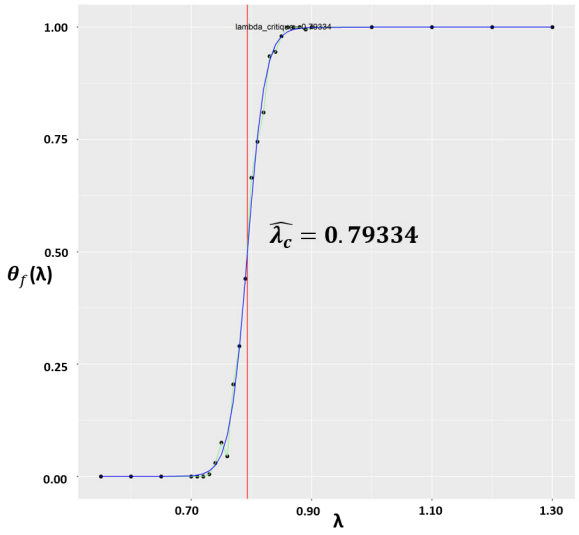


Fig. 4.11 $\theta_f(\lambda)$ Function for s = 30

Besides,we see once again that for N large, the function $\theta_f(\lambda)$ obtained by the simulation coincides better with the function $\theta_f(\lambda)$ obtained by the logistic model. The simulation results of this method are shown in Table 4.3.

The function $\theta_f(\lambda)$ for s differents

As we can increase the size of the observation window, we draw the curve $\theta_f(\lambda)$ for several window sizes. We execute $s = (5, 10, 15, 20, 25)$ as illustrated in Figure 4.12.

Table 4.3 $\widehat{\lambda_c}$: Tore + *Union-Find* Method

The number of simulations	**Tore + *Union-Find***
50	0.79558
100	0.79769
150	0.79425
200	0.79334

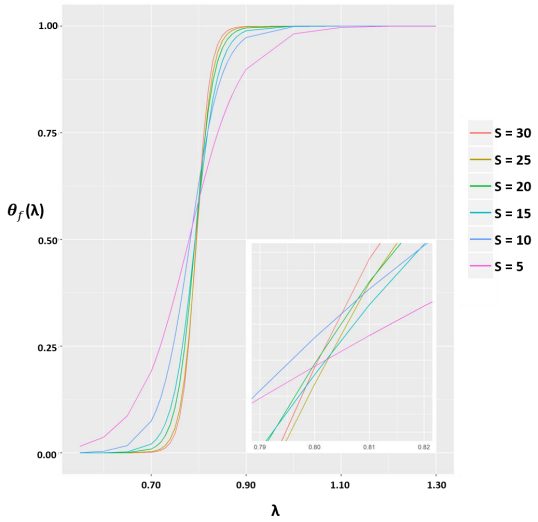


Fig. 4.12 Function $\theta_f(\lambda)$ for s = (5, 10, 15, 20, 25)

In Figure 4.12, we see that these curves intersect in a very localized region. However, when the width of the window s more and more, the curve of $\theta_f(\lambda)$ converges to $\Theta(\lambda)$ when s goes to infinity (4.3). Thus, the curve $\Theta(\lambda)$ must also pass in this region. Taking as a new value of $\theta_f(\lambda_c)$, the approximate value where the curves intersect, we can obtain a better approximation of λ_c. We will take the value $p = 0.6$. Critical threshold estimated for $p = 0.6$ are shown in Table 4.4.

Table 4.4 $\widehat{\lambda_c}$: Method 2: $p = 0.6$

The number of simulations	**Method 2:** $p = 0.6$
50	0.79900
100	0.80083
150	0.79763
200	0.79229

By this table, it is also shown that the estimated values for λ_c with $p = 0.6$ for $s = 5$ are close to the value λ_c given by the Boolean model (see Formula (4.6)).

4.1.1.3 Stretch Factor (μ Function)

As has been said throughout this paper, we want to study multi-hop D2D communication. It will be appropriate to try to look at the number of jumps (the number of users) necessary to establish a D2D communication. The objective of this method is to find a shorter path from one user to another user. We look for the chemical distance to find the number of jumps. It is recalled that the chemical distance between two nodes A and B of a graph is the number of edges of the shortest path from A to B. In this part, we consider a Gilbert graph that percolate. There are therefore many pairs of nodes (users) in the graph. We're looking for the shortest way for all couples using the Floyd-Warshall algorithm. The principle of this algorithm is explained in algorithm 6. The objective is to calculate the shortest paths and their chemical distance.

Algorithm 6: Floyd-Warshall Algorithm

```
Input: Graph
Output: Percolation Graph
J := number of users
Calculate the Euclidean distance (see (4.13))
Save in a matrix J x J
if d ≤ 0.3 then
    1 ;
else
    ∞ ;
end
Floyd-Warshall Algorithm
for k in 1:J do
    for i in 1:J do
        for j in 1:J do
            d[i,j] = min(d[i,j], d[i,k]+ d[k,j])
        end
    end
end
```

First, after placing the users on the streets, we calculate the Euclidean distance between all the users. Recall that the Euclidean distance between two points (A,B) of coordinates (x,y) is:

$$\begin{aligned} d(A,B) &= \|A-B\| \\ &= \sqrt{(X_A-X_B)^2+(Y_A-Y_B)^2} \end{aligned} \tag{4.13}$$

Then, we put the results in a matrix of size mxm. The elements of the matrix are initialized to "one" if the Euclidean distance is smaller than the radius $(r = 0.3\,km)$ and to the "infinite" if the Euclidean distance is greater. We then look for the minimum between the chemical distance from the (user) node i to node j and the chemical distance from node i to node j through another node (node k) (see algorithm 6). This minimum value is defined as the number of hops. The ratio between the number of hops of users want to communicate and their Euclidean distance is called the function μ.

We have an Orange internal research report which gives us the Theorem 4.1.1. saying that there is a value μ for a graph that percolate (that is, for a graph in which there is an infinite connected component). This value ensures that for a very large distance, the number of hops is proportional to the distance for a given territory.

Theorem 4.1.1 *If $\lambda > \lambda_c$, let C be the infinite connected component in the Gilbert graph. Let $N(X_i,X_j)$ be the smallest number of hops from X_i to X_j with $X_i,X_j \in C$. There exists $\mu(\gamma,\lambda,r)$ such that:*

$$\lim_{|X_i-X_j|\to\infty} \frac{N(X_i,X_j)}{|X_i-X_j|} = \mu(\gamma,\lambda,r) \tag{4.14}$$

The algorithm of the method to find the function μ follows the following steps:

1. Generate a Gilbert graph for $s = 5$ with the given parameters (γ,λ,r),
2. For a percolating graph, use the Floyd-Warshall algorithm to find the shortest path of each pair of (user) nodes in the graph and count the number of hops in each shorter path,

In the **R** software, there is a function to execute the Floyd-Warshall algorithm, hence we use this function in this method.

3. In the largest connected component, keep users who have a distance greater than $4\,km$. We choose this distance because we want the greatest possible distance for $s = 5$,
4. Calculate the function $\mu = \frac{number\,of\,hops}{distance}$ (see Equation (4.14)),
5. Remake N simulations and calculate the average.

4.1.2 The Second Type of Road: the Poisson-Delaunay Tessellation (PDT)

We looked for the percolation threshold $(\widehat{\lambda_c})$, the percolation function at a given point $(\theta_0(\lambda))$ and also the ratio function of the number of hops at the distance $(\mu(\lambda))$ for the first type of road (PVT). We are now looking for the second type of road: the Poisson-Delaunay tessellation (PDT). We proceed in the same way as for the Poisson-Voronoi tessellation. The parameter γ (the average street length per km^2) for this tessellation is $\gamma = \frac{32\sqrt{\zeta}}{3\pi}$. Recall that ζ is the expected number of points in a window of a unit area. We have set γ equal to $20\,km/km^2$, therefore the value ζ must be equal to 34.69783 in order to have $\gamma = 20\,km/km^2$.

The road type difficulty is to plot the PDT graph. When we simulate the tessellation, we connect together all the points that are at the edge of the window as a result that the number of edges of this tessellation is exorbitant. In order to obtain the appropriate graph, it is necessary to eliminate the tessellation edges on the edge of the observation window. For this, we will increase the window size.

In the first part, to find the critical threshold, we use the torus + *union-find* method with $s = 30$. Before the tessellation is drawn, the points of the Poisson process with coordinates (x,y) such that $x \leq 1$ and the coordinate points (x,y) such that $y \leq 1$ are copied beyond the upper limit of s. The size of the window is equal to the size of the previous window $+1$. We draw the tessellation in this window and then we consider the tessellation in the sub-window of size $s = 30$ with coordinates x and y equal to 0.5 up to x and y equal to 30.5. This avoids having the edges that accumulate at the edge of the window. The user's points are then placed on the edges of the tessellation in the sub-window and then all the points of the coordinates (x,y) users such as $y \leq (r/5 - 0.5)$ are copied to the edges which are beyond the limit of the sub-window. We do in the same way for the set of points of the users of coordinates (x,y) such that $x \leq (r/5 - 0.5)$. This is done to apply the torus concept as in the torus + *union-find* PVT method. Finally, we look for the value $\widehat{\lambda_c}$ using the logistic model with $p = 0.6$.

In the second part, we use the same algorithm as for the Poisson-Voronoi tessellation to find the function curve of $\theta_0(\lambda)$ of the Poisson-Delaunay tessellation. Here, we draw a tessellation in the observation window and then we consider the tessellation for a sub-window. For our algorithm, we only need to count the total length of the edges of the tessellation, then we propose a calculation method: we trace the tessellation in the window size $s = 6$. We take a sub window of size $s = 5$ so that the window is cut to size 0.5 to 5.5 (see Figure 4.13).

For all the edges that cross the limit of the sub-window, we look for the length of the part of these edges which are in the window $s = 5$. We calculate this length u (see Figure 4.14) by the Formula (4.15), if the edges come from two points (A,B) such that the point A

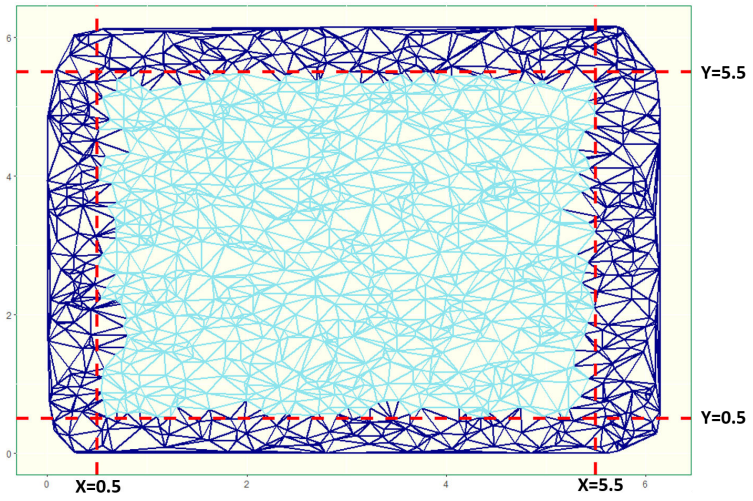


Fig. 4.13 Poisson-Delaunay Tessellation

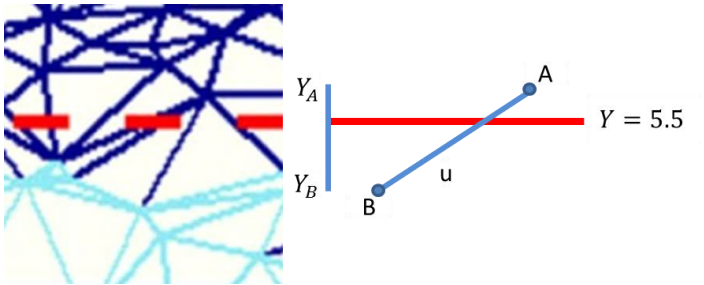


Fig. 4.14 The calculation of the length of PVT

has coordinates (x_A, y_A) such that $y_A > 5.5$ and the point B has coordinates (x_B, y_B) such that $y_B < 5.5$. For two points (A, B) such that point A has coordinates (x_A, y_A) such that $y_A > 0.5$ and point B has coordinates (x_B, y_B) such that $y_B < 0.5$, we use the Formula (4.16). Due to these formulas, we can obtain the length of the edges.

$$\frac{u}{d(A,B)} = \frac{5.5 - y_B}{y_A - y_B} \tag{4.15}$$

$$\frac{u}{d(A,B)} = \frac{0.5 - y_B}{y_A - y_B} \tag{4.16}$$

We do the same for the edges (A, B) for which the point A has coordinates (x_A, y_A) such that $x_A > 5.5$ and the point B has coordinates (x_B, y_B) such that $x_B \leq 5.5$. It is also done for $x < 0.5$.

In the last part, we look for the function curve of $\mu(\lambda)$. Since we do not need to calculate the length of the edges here, we proceed as in the first part to find the critical threshold.

4.2 Outcome Analysis

After launching the algorithms of the methods to find $\widehat{\lambda_c}$, the function $\theta_0(\lambda)$ and the function $\mu(\lambda)$, the following results were obtained:

4.2.1 Estimation of the Percolation Threshold ($\widehat{\lambda_c}$)

We have already chosen the torus + *union-find* method to find $\widehat{\lambda_c}$ with the parameters $\gamma = 20\ km/km^2$ and $r = 0.3\ km$ for the two types of roads: the Poisson-Voronoi tessellation (PVT) and the Poisson- Delaunay tessellation (PDT). The estimated values are shown in Table 4.5. For $\gamma = 20\ km/km^2$, we have $\widehat{\lambda_c} = 0.8$ users/km of street be $\gamma\lambda = 16$ users/km^2.

Table 4.5 $\widehat{\lambda_c}$: Tore + *Union-Find* Method

r	$\widehat{\lambda_c}$		
	PVT	PDT	PBM
0.300	0.79	0.80	0.797

Now, we also want to find the values $\widehat{\lambda_c}$ for different values of γ. By scale invariance, a small value of r corresponds to a small value of γ, which makes it possible to obtain the value of the percolation threshold for urban or rural roads. However, it is a bit difficult to vary γ in the simulation. Consequently, instead of varying γ, we can vary r due to Formula γ. We choose the interval of r:

$$\frac{500}{20} \leq r < 500 \tag{4.17}$$

The maximum value is 500 m, which means that users can communicate up to 500 meters on a territory. We know that:

$$\lambda_c\left(\frac{\gamma}{a}, r\right) = \frac{1}{a}\lambda_c\left(\gamma, \frac{r}{a}\right) \tag{4.18}$$

The values estimated by this algorithm for different values of r are given in Table 4.6 for the two types of roads with the parameters $\gamma = 20\ km/km^2$ and $r = variable$ and we compare them with the values λ_c for the Boolean model.

Table 4.6 $\widehat{\lambda_c}$ by the Simulations for $\gamma = 20$

r (en km)	$\widehat{\lambda_c}$ (in users/km)		
	PVT	PDT	PBM
0.015	237	265	318.889
0.025	111.63	115.66	114.846
0.075	14.34	15.56	12.761
0.125	4.60	5.34	4.594
0.175	2.27	2.58	2.343
0.225	1.42	1.51	1.418
0.275	0.91	0.99	0.949
0.325	0.65	0.70	0.680
0.375	0.48	0.52	0.510
0.425	0.38	0.40	0.397
0.475	0.30	0.32	0.318

Some remarks in Table 4.6, we see that:

1. The $\widehat{\lambda_c}$ values are not very different among the three models, especially for the values $\widehat{\lambda_c}$ from $r = 0.225\,km$ to $r = 0.475\,km$.
2. When r decreases, for example for $r = 0.025\,km$ up to $r = 0.175\,km$, these values $\widehat{\lambda_c}$ are a little farther away from the values λ_c of the Boolean model. It's normal because the Boolean model is valid for dense roads, whereas when r is small, it is equivalent to small γ, the road becomes less and less dense,
3. We also see that the values of $\widehat{\lambda_c}$ are different for PVT and PDT. The percolation threshold depends well on the road model, it is higher for PDT,
4. For $r = 0.015$ (which corresponds to $\gamma = 1$ and $r = 0.300$), the values are very different from the Boolean model.

This table gives the estimated values for each r (see Table 4.6). Therefore, from this table, we can find $\widehat{\lambda_c}$ easily for different values of γ with the calculation by the Formula (4.18) of the parameters $\gamma = variable$ in km/km^2 and $r = 0.3\ km$ (see Table 4.7).

4.2.2 Estimation of the Percolation Probability ($\theta_0(\lambda)$ function)

In this part, we look for the probability that a given user is in the D2D communication graph. The Algorithm (5) is launched with the parameters $\gamma = 20\ km/km^2$, $\lambda = variable$,

Table 4.7 $\widehat{\lambda_c}$ by the Calculations for $r = 0.3\,km$

γ	$\widehat{\lambda_c}$		
	PVT	PDT	PBM
1	11.85	13.25	15.94
1.67	9.32	9.36	9.55
5.00	3.59	3.72	3.19
8.33	1.91	2.19	1.91
11.67	1.32	1.48	1.37
15.00	1.06	1.11	1.06
18.33	0.83	0.89	0.87
21.67	0.70	0.74	0.74
25.00	0.60	0.65	0.63
28.33	0.54	0.57	0.56
31.67	0.48	0.51	0.50

and $r = 0.3\ km$ and we find the function θ_0 shown in Figure 4.15 for the Poisson-Voronoi tessellation and in Figure 4.16 for the Poisson-Delaunay tessellation.

From these curves, we see that if the value of λ is greater than one $(\lambda > 1)$, we have a high probability (that is, we are almost certain) that the given point (given user) is in the D2D communication graph. On the contrary, if the value of λ is less than 0.5 $(\lambda < 0.5)$, we are almost sure that it is not in the communication graph D2D, because the probability $\theta_0 \approx 0$.

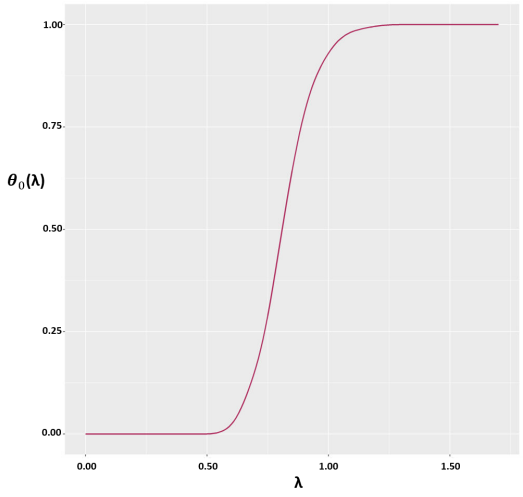


Fig. 4.15 Function $\theta_0(\lambda)$ PVT

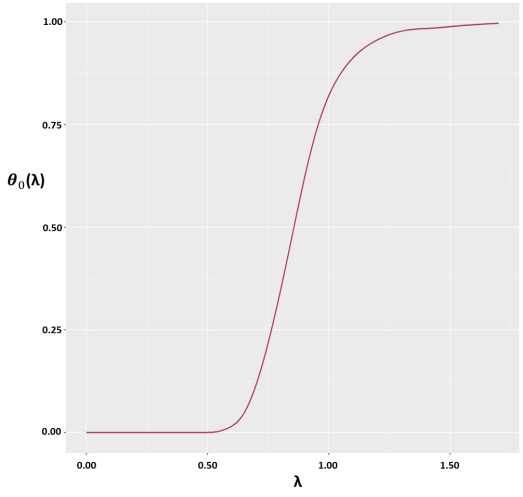


Fig. 4.16 Function $\theta_0(\lambda)$ PDT

In addition, instead of varying γ, it is possible to vary r also to find the curves $\theta_0(\lambda)$ with different values of r in the interval (4.17). Some examples of θ_0 curves can be seen in Figure 4.17 for PVT type roads and in Figure 4.18 for PDT type roads.

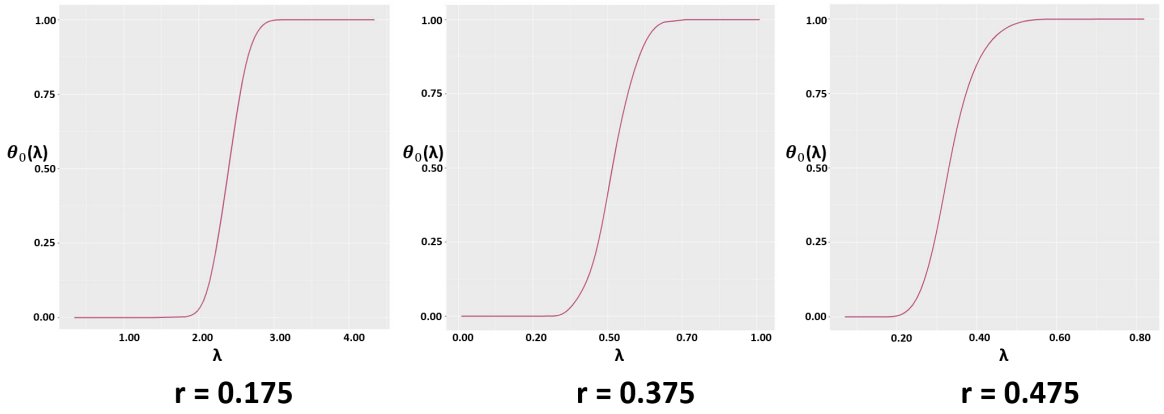


Fig. 4.17 The function $\theta_0(\lambda)$ r varied for PVT

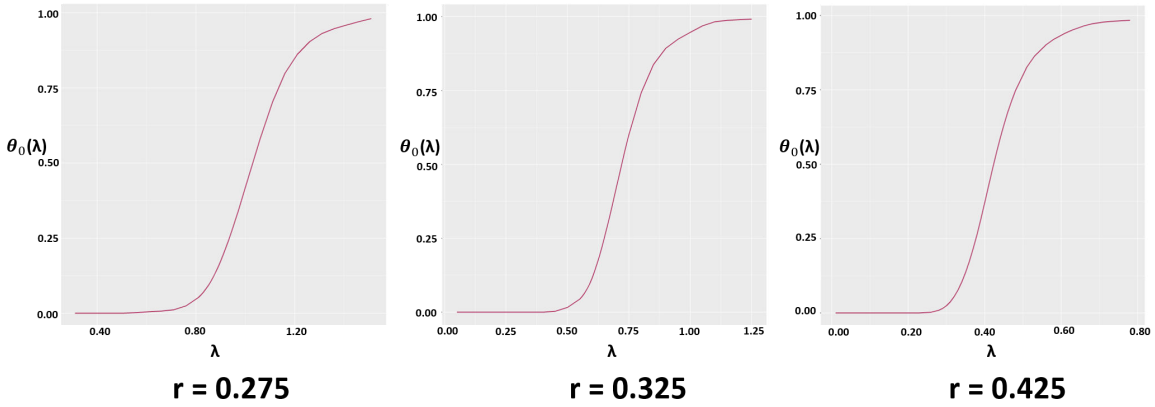


Fig. 4.18 The function $\theta_0(\lambda)$ r varied for PDT

We can find the curves for different values of γ due to the Formula (4.19) for PVT type roads and also for PDT type roads.

$$\theta_0\left(\frac{\gamma}{a}, r, \lambda\right) = \theta_0\left(\gamma, \frac{r}{a}, \frac{\lambda}{a}\right) \tag{4.19}$$

The function $\theta(\lambda)$ is then compared for different window sizes. We do $s = 5$ and $s = 30$ with $r = 0.475$ for PDT type roads (see Figure 4.19 and Figure 4.20). From these figures, we can see that the curve of the window $s = 30$ is more vertical than the curve of the window $s = 5$. Seeing at Table 4.6, the value $\widehat{\lambda_c}$ by the torus + *union find* method is equal at 0.32 for PDT. Here, by the curve of $\theta_0(\lambda)$ with $s = 5$, we find that the first value of λ such that the probability $(\theta_0(\lambda))$ is different from 0 is far from the value $\widehat{\lambda_c}$, while for the curve of $(\theta_0(\lambda))$ with $s = 30$, we are close to this value.

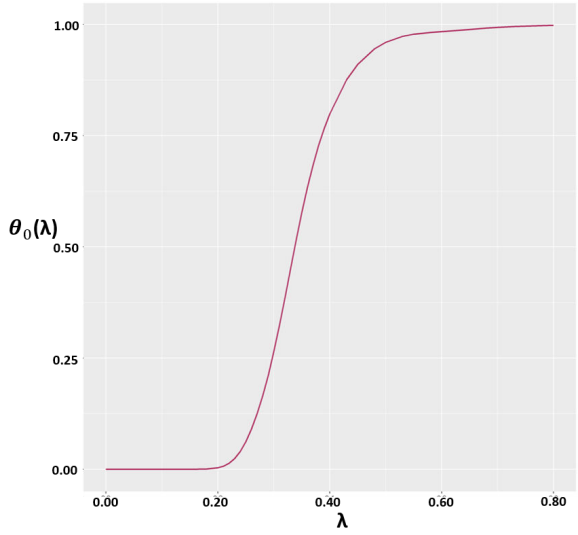


Fig. 4.19 Function $\theta_0(\lambda)$ PDT $s = 5$

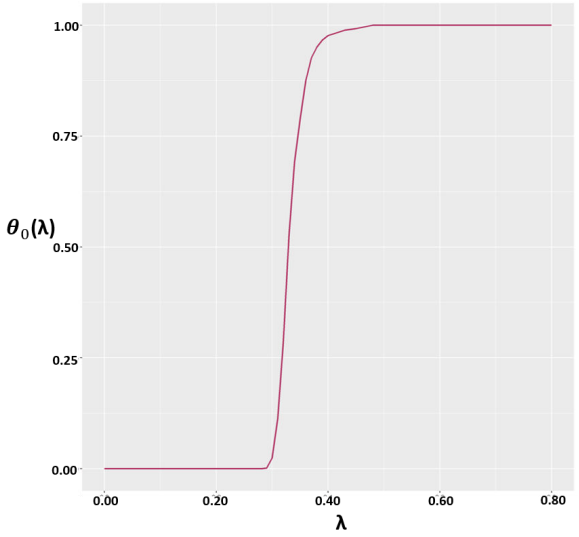


Fig. 4.20 Function $\theta_0(\lambda)$ PDT $s = 30$

4.2.3 Estimation of the Stretch Factor ($\mu(\lambda)$ function)

We also find the function curve $\mu(\lambda)$ for the two roads types with the same parameters of γ, λ and r (see Figure 4.21 and Figure 4.22).

It is a graph of the function $\mu(\lambda)$ which represents the value μ $(hops/km)$ as a function of λ $(users/km$ of street$)$. When two users communicate, the value μ is the ratio of the number of jumps to the distance. It can be seen that these curves decrease asymptotically towards a value greater than 3.3 because to cross a distance of 1 km with radius $r = 300\,m$, at least three users are needed. In addition, by these curves, we can easily find the number of jumps for a given distance. For example, for the function curve $\mu(\lambda)$ of the PDT type road network, we want to know how many jumps (hops) we need to cross a distance $d = 2$ km. We see that for $\lambda = 1.25$, we have $\mu \approx 5.35$ hence there are 5.35 $hopss/km$ times2 km, that is about 11 jumps (11 hops) over a distance of 2 km to establish the D2D communication.

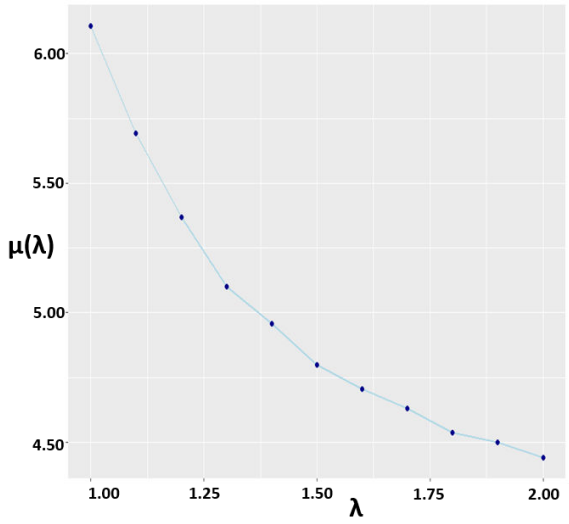


Fig. 4.21 Function $\mu(\lambda)$ PVT

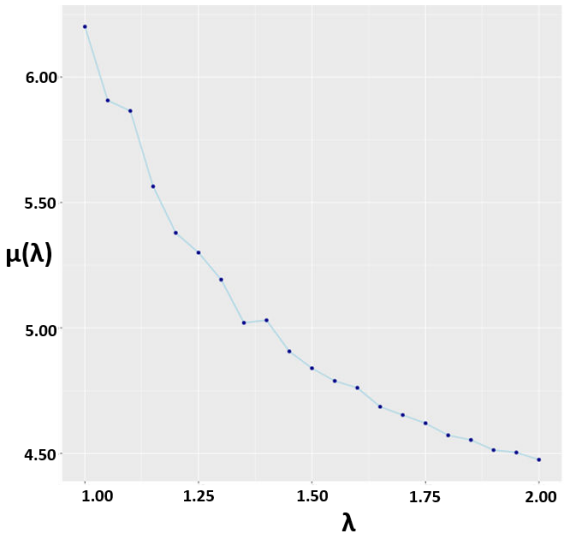


Fig. 4.22 Function $\mu(\lambda)$ PDT

We can also vary r to find the curves $\mu(\lambda)$ with different values of r in the interval(4.17). The following curves are examples of μ curves (see Figure 4.23) for PVT type roads. Then, it is also done to find the curves of the function $\mu(\lambda)$ for PDT type roads (see Figure 4.24).

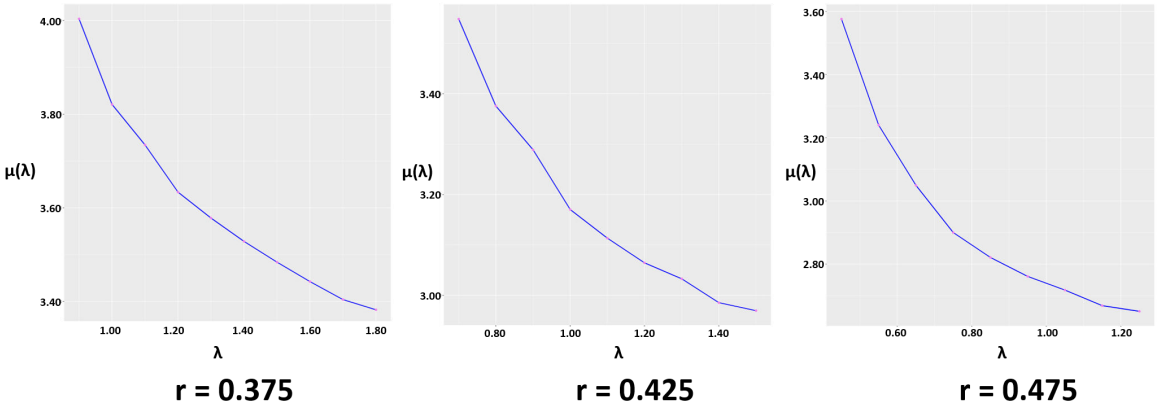


Fig. 4.23 Function $\mu(\lambda)$ r varied for PVT

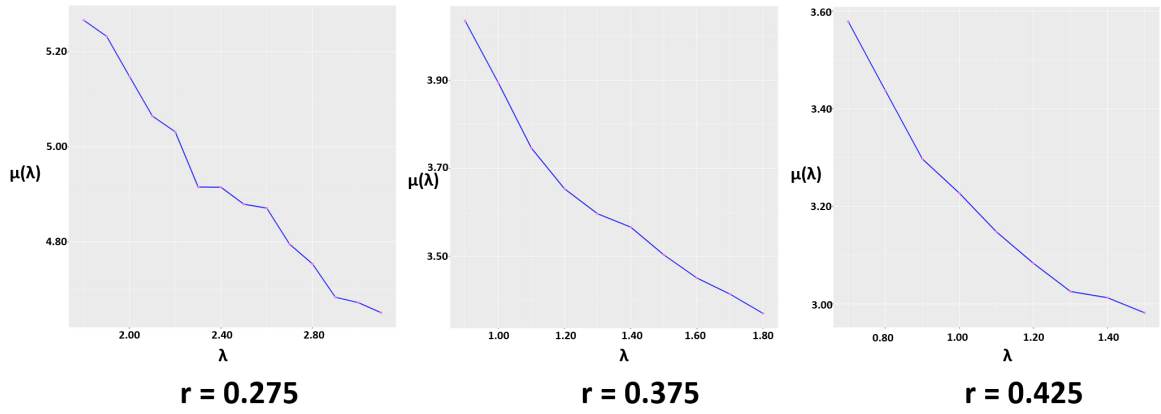


Fig. 4.24 Function $\mu(\lambda)$ r varied for PDT

Due to Formula (4.20), we can also plot the function of $\mu(\lambda)$ with different values of γ for Poisson-Voronoi tessellation and Poisson-Delaunay tessellation.

$$\mu\left(\frac{\gamma}{a}, r, \lambda\right) = \mu\left(\gamma, \frac{r}{a}, \frac{\lambda}{a}\right) \tag{4.20}$$

4.3 Error Analysis of the Simulation Result

Since in this study we are doing simulations, the values found are estimated value. Therefore, we must look at the confidence interval of these values. We simulate the algorithm of the tore + *union find* PVT method with $N = 200$ simulations 30 times, then there are 200×30 data. For each simulation we obtained the probability $\widehat{\theta_f(\lambda)}$, we then look for the mean of 30 data of the probability $\widehat{\theta_f(\lambda)}$ and the confidence interval of $\widehat{\theta_f(\lambda)}$. A threshold of 90% data is set to obtain the upper limit of the interval and 10% data for the lower limit of the interval (see Figure 4.25 and Figure 4.26). In these figures, we find that the values obtained by simulation for $s = 5$ are more dispersed than the values obtained by simulation for $s = 25$. This shows that the selection of the observation window size is very important to avoid the large variance of the data.

Furthermore, we also want to see the confidence interval for the estimated value of λ_c. A confidence interval provides a range of values that is likely to contain the value of interest of the actual case. By doing N simulations and taking the mean of the results by the Equation (4.4), we obtained an estimated value of the percolation threshold $\widehat{\lambda_c}$ by the logistic model for s given. This estimate is repeated M times, so we have M values $\widehat{\lambda_c}$ for s given. Let $(\lambda_c^i)_{1 \leq i \leq M}$ be the estimated values of λ_c. We search $\overline{\lambda_c^i}$ for the mean and the standard deviation S, a measure of the threshold dispersion. Suppose that M is sufficiently large (here

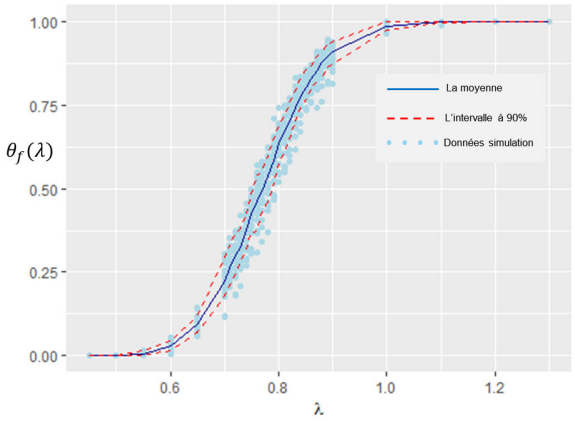


Fig. 4.25 Function $\theta_f(\lambda)$ PVT $s = 5$

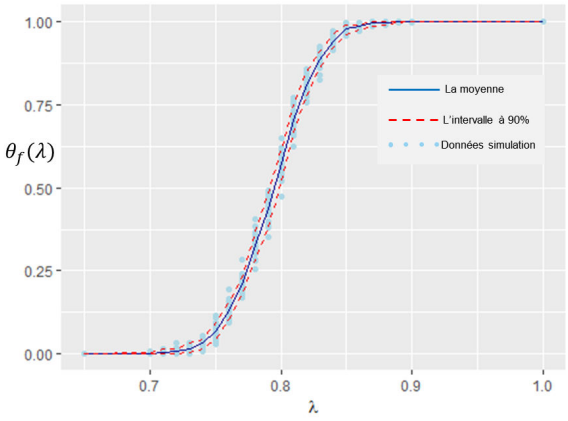


Fig. 4.26 Function $\theta_f(\lambda)$ PVT $s = 25$

we have $M = 30$), by the central limit theorem (CLT), we can construct a 95% confidence interval given by:

$$\left[\overline{\lambda_c^i} \pm \frac{t_\alpha S}{\sqrt{M}}\right] \tag{4.21}$$

where t_α, the confidence level coefficient for the 95% confidence interval is 1.96. In addition, we can easily find the average as in Equation (4.4) and from this, we can find the standard deviation S by Formula (4.22):

$$S = \sqrt{\frac{\sum(\lambda_c^i - \overline{\lambda_c^i})^2}{M-1}} \tag{4.22}$$

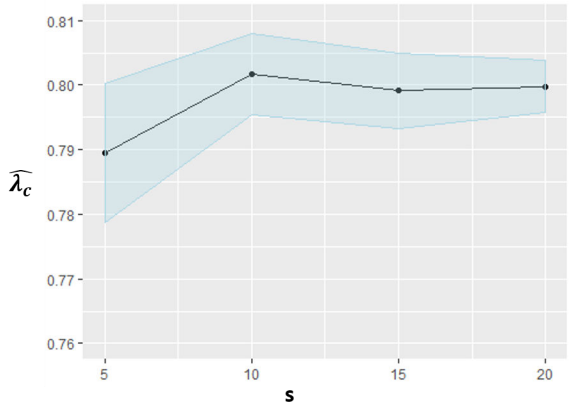


Fig. 4.27 Confidence interval for $\widehat{\lambda_c}$

We try different observation window size $s = (5, 10, 15, 20)$. In Figure 4.27, shows that the critical value becomes stable when the window size increases (this is also shown in Figure 4.25). We see that for a small window s, the confidence interval is wide when we enlarge the window size, the size of the confidence interval is as well.

Chapter 5

Conclusion and Suggestion

5.1 Conclusion

5.1.1 Conclusion of the Research Results

The best method has been selected according to the closest estimated value $\widehat{\lambda_c}$ to λ_c Boolean model and time execution in **R** programming. Hence, union-find + tore method is selected based on these criteria and the smallest error as has been described in the earlier section of chapter 4. Besides, the results obtained in this study show that critical percolation threshold for rural and urban areas is very different. That means the length and density of the type road is very influential on the D2D communication network. On the other side, for rural areas, Poisson-Delaunay tessellation (PDT) is well represented on this type of road meanwhile Poisson-Voronoi tessellation (PVT) is well for urban areas.

5.1.2 Conclusion of Researchers

This internship was very enriching for me, because it allowed me to discover new scientific fields and the domain of probability theory and also stochastic geometry for proposing some solutions to the problem of the communication D2D graph which will be used in the fifth generation mobile network: 5G. It allowed me to participate concretely in this new generation through the Orange TEMS project that I particularly appreciated and I am very proud to have been able to contribute and participate in this revolution. With this experience, I am confident that I will be able to bring myself for working in the future.

5.2 Suggestion

We can also try another road type like Manhattan road type and other tessellation type like a line tessellation model.

References

[1] Baccelli, F. and Blaszczyszyn, B. (2009a). *Stochastic Geometry and Wireless Network, Volume I - Theory*. Bacceli, F and Blaszczyszyn, B., 1, pp.148, <inria-00403039v1>.

[2] Baccelli, F. and Blaszczyszyn, B. (2009b). *Stochastic Geometry and Wireless Network, Volume II - Applications*. Bacceli, F and Blaszczyszyn, B., 1, pp.148, <inria-00403040v1>.

[3] Baddeley, A. (2004). Spatial point process modelling and its applications.

[4] Baddeley, A. et al. (2008). Analysing spatial point patterns in r. Technical report, CSIRO, 2010. Version 4. URL https://research. csiro. au/software/r-workshop-notes.

[5] Balister, P., Bollobás, B., and Walters, M. (2005). Continuum percolation with steps in the square or the disc. *Random Structures Algorithms*, 26(4:392–403).

[6] Breton, J.-C. (2008). *Statistiques, IUT Biotechnologie 2ème année*. site https://perso.univ-rennes1.fr/jean-christophe.breton/, La Rochelle.

[7] Collard, P., Verel, S., and Clergue, M. (2013). *Systèmes complexes: Une introduction par la pratique*. Presses polytechniques et univesitaires romandes, Lausanne.

[8] Frank, F., Gloaguen, C., Schmidt, V., and Voss, F. (2009). Simulation of the typical poisson–voronoi–cox–voronoi cell. *Journal of Statistical Computation and Simulation*, 79(7):939–957.

[9] Gloaguen, C., Fleischer, F., Schmidt, H., and Schmidt, V. (2006). Fitting of stochastic telecommunication network models via distance measures and monte—carlo tests. *Telecommun. Syst.*, 31(4):353–377.

[10] Grimmett, G. (1999). *Percolation*. Springer Verlag.

[11] Karrer, B., Newman, M., and Zdeborová, L. (2014). Percolation on sparse networks. *Phys. Rev. Lett.*, 113(208702).

[12] Kingman, J. (1993). *Poisson Processes*. Oxford University Press, Oxford.

[13] Matias, C. (2015-2016). *Analyse statistique de graphes*. http://cmatias.perso.math.cnrs.fr/, Paris.

[14] Meester, R. and Roy, R. (1996). *Continuun Percolation*. Cambridge University Press.

[15] Mertens, S. and Moore, C. (2012). Continuum percolation thresholds in two dimensions. *Physical Review E*, 86.6(061109).

[16] Neuhäuser, D., Hirsch, C., Gloaguen, C., and Schmidt, V. (2014). Ratio limits and simulation algorithms for the palm version of stationary iterated tessellations. *Journal of Statistical Computation and Simulation*, 84(7):1486–1504.

[17] Neuhäuser, D., Hirsch, C., Gloaguen, C., and Schmidt, V. (2016). A stochastic model for multi-hierarchical networks. *Methodology and Computing in Applied Probability*, 18(4):1129–1151.

[18] Newman, M. E. and Ziff, R. M. (2001). Fast monte carlo algorithm for site or bond percolation. *Physical Review E*, 64.1(016706).

[19] Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. (2000). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, New York.

[20] Schnoerr, D., Grima, R., and Sanguinetti, G. (2016). Cox process representation and inference for stochastic reaction–diffusion processes. [online] http://dx.doi.org/10.1038/ncomms11729.

Appendix A

$\mathbb{R}$ Codes

Here, we can find the code of some methods. We chose the Torus + *Union-Find* PVT method, the function of $\theta_f(\lambda)$ PDT method, and the function of $\mu(\lambda)$ PDT method.

Packages used by all methods

- library(spatstat)
- library(deldir)
- library(geomnet)
- library(igraph)
- library(ggraph)
- library(reshape2)
- library(ggplot2)
- library(ggthemes)
- library(foreach)
- library(plyr)
- library(Rfast)

Torus + *Union Find* PVT Method

```
# fenêtre d'observation
win <- 5
# L'intensité de ppp
i <- 100
# Paramètres
gamma  <- 2*sqrt(i)
max_rad <- .5
min_rad <- max_rad/gamma
# essayer avec différentes valeurs de radius
#seq_radius <- seq(min_rad, max_rad, .05)
seq_radius <- seq(.325, max_rad, .05)
# Itération
REPLICATE <- 2
# Lien de stockage
path = "C://Users//PZDB7620//Documents//Local//RESULTAT//M2//"
###############
## SIMULATION DU PROCESSUS DE POISSON
###############
simulate.poisson = function (i, win, image = F ){
  poisson <- rpoispp(i, win = owin(c(0,win), c(0,win)))
  # modifier dans une trame de données
  poisson_df <- as.data.frame(poisson)
  # identifier les points en bas et à gauche qui sont inférieurs au rayon/2
  coord_bottom = poisson_df[poisson_df$y<=radius/2,]
  coord_bottom$y  = abs(coord_bottom$y-radius/2) + win
  coord_left = poisson_df[poisson_df$x<=radius/2,]
  coord_left$x  = abs(coord_left$x-radius/2) + win
  # ajouter des points
  addpoint_bott = coord_bottom
  addpoint_left = coord_left
  # Combiner ces nouveaux points avec tous les points avant
  poisson_df = rbind (poisson_df, addpoint_bott, addpoint_left)
  # tracer des points de processus de Poisson en utilisant la fonction ggplot
  if (image){
    print(ggplot(poisson_df) +
```

```
            geom_point(aes(x,y), size = 2, color ='darkgoldenrod1') +
            labs(x="", y="") +
            ggtitle("Poisson Point Process") +
            theme(plot.title = element_text(hjust=.5,
            face ='bold', color ='darkgreen', size = 14)))
    # créer un fichier avec le graphique
    title = paste(path,"pppn",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  attributes(poisson_df)$i=i
  attributes(poisson_df)$win=win
  return(poisson_df)
}
poisson_df = simulate.poisson(i, win, image = F)
###############
## POISSON-VORONOI TESSELATION POUR LA VOIRIE
###############
simulate.voirie = function (poisson_df = NA, i = NA, win = NA, image = F,
include.poisson_df = F){
  if (is.na(poisson_df)){
    poisson_df = simulate.poisson(i, win)
  }
  # poisson-Voronoi tesselation
  voirie <- deldir(poisson_df$x, poisson_df$y)
  # tracer Poisson-voronoi
  if (image){
    print(ggplot(data = poisson_df, aes(poisson_df$x, poisson_df$y)) +
            geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2), size = 1,
            linetype = 1, color ="darkblue", data = voirie$dirsgs) +
            geom_point(fill = 'darkgoldenrod1', pch=21, size = 2,
            color = 'navyblue') +
            ggtitle('Poisson-Voronoi Tesselation') +
            theme(
              plot.title = element_text(hjust =.5, face="bold",
              color = 'darkgreen', size = 14),
```

```
              panel.background = element_rect(fill = 'ivory1',
              colour = 'springgreen4'),
              axis.title.x = element_blank(), axis.title.y = element_blank()))
    # créer un fichier avec le graphique poisson-voronoi
    title = paste(path,"pvtn",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  if (image){
    print(ggplot(data=voirie$dirsgs) +
            geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2), size = 1,
            linetype = 1, color= "darkblue") +
            ggtitle('Voirie : Poisson-Voronoi Tesselation') +
            theme(
              plot.title = element_text(hjust =.5, face="bold",
              color = 'darkgreen', size = 14),
              panel.background = element_rect(fill = 'ivory1',
              colour = 'springgreen4'),
              axis.title.x = element_blank(), axis.title.y = element_blank()))
    # créer un fichier avec le graphique poisson-voronoi
    title = paste(path,"voirien",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in",
    res = 300)
    dev.off()
  }
  attributes(voirie)$i <- attributes(poisson_df)$i
  attributes(voirie)$win <- attributes(poisson_df)$win
  if (include.poisson_df){
    attributes(voirie)$poisson_df = poisson_df
  }
  return(voirie)
}
voirie = simulate.voirie(poisson_df = poisson_df, image = F)
##############
## PROCESSUS de COX POUR LES UTILISATEURS
##############
```

```
simulate.utilisateurs = function(lambda, voirie = NA, i = NA, win = NA,
image = F, include.voirie = F){
  if (is.na(voirie)){
    voirie = simulate.voirie(i = i, win = win)
  }
  # définir la fenêtre de la voirie
  window <- owin(c(min(voirie$dirsgs$x1,voirie$dirsgs$x2),
  max(voirie$dirsgs$x1, voirie$dirsgs$x2)), c(min(voirie$dirsgs$y1,
  voirie$dirsgs$y2), max(voirie$dirsgs$y1,voirie$dirsgs$y2)))
  # commençer par dessiner les rues que nous avons trouvées
  line <- psp(voirie$dirsgs$x1, voirie$dirsgs$y1, voirie$dirsgs$x2,
  voirie$dirsgs$y2, window = window)
  # placer les utilisateurs sur les rues
  users <- rpoisppOnLines(lambda, line)
  # transformer en trame de données
  users_df <- as.data.frame(users)
  colnames(users_df) <- c('userx', 'usery')
  # supprimer des points qui traversent la fenêtre
  users_df <- users_df[(users_df$userx <= win & users_df$userx >= 0 &
  users_df$usery <= win & users_df$usery >= 0),]
  # identifier les points en bas et à gauche qui sont inférieurs au rayon/2
  coord_bottom = users_df[users_df$usery<=radius/2,]
  coord_bottom$usery  = abs(coord_bottom$usery-radius/2) + win
  coord_left = users_df[users_df$userx<=radius/2,]
  coord_left$userx  = abs(coord_left$userx-radius/2) + win
  # ajouter des points
  addpoint_bott = coord_bottom
  addpoint_left = coord_left
  # Combiner ces nouveaux points avec tous les points avant
  users_df = rbind (users_df, addpoint_bott, addpoint_left)
  # tracer des utilisateurs sur les rues
  if (image){
    print(ggplot(data = voirie$dirsgs) +
            geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2), size = 1,
            linetype = 1, color= "darkblue") +
            geom_point(aes(userx, usery), fill = 'red2', pch=21, size = 3,
```

```
            color = 'black', data = users_df)+
            ggtitle('Utilisateur : Processus de Cox') +
            theme(
              plot.title = element_text(hjust =.5, face="bold",
              color = 'darkgreen', size = 14),
              panel.background = element_rect(fill = 'ivory1',
              colour = 'springgreen4'),
              axis.title.x = element_blank(), axis.title.y = element_blank()))
    # Pour créer un fichier avec le graphique.
    title = paste(path,"utilisateurn",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  attributes(users_df)$i = attributes(voirie)$i
  attributes(users_df)$win = attributes(voirie)$win
  attributes(users_df)$lambda = lambda
  if (include.voirie){
    attributes(users_df)$voirie = voirie
  }
  return(users_df)
}
users_df = simulate.utilisateurs(lambda = lambda, voirie = voirie,
win = win, image = F, include.voirie = T)
##############
## ALGORITHME UNION FIND
##############
union.find = function(users_df){
  connected_nodes <- c(var1 = integer(), var2 = integer(), distance = integer()
  users <- c()
  weight <- c()
  if (nrow(users_df) != 0){
    rownames(users_df) <- seq(1:nrow(users_df))
    for (index in 1:10){
      var1 <- seq(1 : nrow(users_df))
      var2 <- nnwhich(users_df, k = index)
      distance <- nndist(users_df, k = index)
```

```
      connected_nodes <- rbind(connected_nodes, cbind(var1, var2, distance))
    }
    # préciser qu'il existe une arête ssi la distance entre deux points
    (utilisateurs) est plus petite que le rayon r.
    connected_nodes = connected_nodes[which((connected_nodes[,3] <= radius) &
    connected_nodes[,1]< connected_nodes[,2]),]
    ## initialiser users arrays
    users <- seq(1, nrow(users_df))
    weight <- rep(1, nrow(users_df))
  }
  if (length(connected_nodes) != 0){
    for (index in seq(1, nrow(connected_nodes))) { # for each connection
      p <- connected_nodes[index, 1] # sample random objects
      q <- connected_nodes[index, 2]
      x <- users[p]; while (users[x] != x) x <- users[x] # FIND
      y <- users[q]; while (users[y] != y) y <- users[y]
      if (x == y) next
      if (weight[x] < weight[y]) {# UNION
        users[x] <- y; weight[y] <- weight[y] + weight[x];
      }
      else{ users[y] <- x; weight[x] <- weight[x] + weight[y]; }
    }
  }
  return(cbind(users,weight,users_df))
}
#gr = union.find(users_df)
##########
## VERIFIER la PERCOLATION
##########
percolates.comb2r = function(users, win, radius){
  users_union <- users$users
  weight <- users$weight
  users_df <- as.data.frame(cbind(users$userx,users$usery))
  colnames(users_df) <- c("userx","usery")
  reponse <- F
  top <- as.numeric(rownames(users_df[users_df$usery >= win,]))
```

```
  right <- as.numeric(rownames(users_df[users_df$userx >= win,]))
  bottom <- as.numeric(rownames(users_df[users_df$usery <= (radius/2),]))
  left <- as.numeric(rownames(users_df[users_df$userx <= (radius/2),]))
  for (p in top){
    for (q in bottom){
      x <- users_union[p]; while (users_union[x] != x) x <- users_union[x] # FIND
      y <- users_union[q]; while (users_union[y] != y) y <- users_union[y]

      if (x == y){
        return (reponse = T)
      }
    }
  }
  for (p in left){
    for (q in right){
      x <- users_union[p]; while (users_union[x] != x) x <- users_union[x] # FIND
      y <- users_union[q]; while (users_union[y] != y) y <- users_union[y]
      if (x == y){
        return (reponse = T)
      }
    }
  }
  return(reponse)
}
#percolates.comb2r(gr, win = win, radius = radius)
##########
## SEUIL de PERCOLATION TORE
##########
# essayer avec différentes valeurs de radius et différentes valeurs de lambda
pt = proc.time()
perc.result <- c(radius=integer(),lambda=integer(),percolates=integer())
for (radius in seq_radius){
  prop_lambda <- (4.51*(1/radius^2))/(pi*gamma)
  if (radius>=0.3){
    seq_lambda <- sort(unique(abs(round(c(seq(prop_lambda-0.75,
    prop_lambda, 0.25),
```

```
  seq(prop_lambda-(radius/2), prop_lambda+(radius/2), .01),
  seq(prop_lambda,
  prop_lambda+0.75, 0.25)), digits = 2))))
}else if (radius>=0.2){
  seq_lambda <- sort(unique(abs(round(c(seq(prop_lambda-0.75,
  prop_lambda, 0.25),
  seq(prop_lambda-(radius), prop_lambda+(radius), .01),
  seq(prop_lambda,
  prop_lambda+0.75, 0.25)), digits = 2))))
}else if (radius>0.1){
  seq_lambda <- sort(unique(abs(round(c(seq(prop_lambda-2,
  prop_lambda, 0.5),
  seq(prop_lambda-(radius*3), prop_lambda+(radius*6), .04),
  seq(prop_lambda,
  prop_lambda+2, 0.5)), digits = 2))))
}else if(radius>=0.75){seq_lambda <- sort(unique(abs(round(
c(seq(prop_lambda-3,
prop_lambda, 1), seq(prop_lambda-(radius*3)+2, prop_lambda+
(radius*8)+2, .05),
seq(prop_lambda,  prop_lambda+8, 1)), digits = 2))))
}else if(radius>=0.05){seq_lambda <- sort(unique(abs(round(
c(seq(prop_lambda-3,
prop_lambda, 1), seq(prop_lambda-(radius*3)+2, prop_lambda+
(radius*8)+8, 0.5),
seq(prop_lambda,  prop_lambda+12, 2)), digits = 2))))
}else {seq_lambda <- sort(unique(abs(round(c(seq(prop_lambda-16,
prop_lambda, 2.5),
seq(prop_lambda-(radius*8)-5, prop_lambda+(radius*3)-1, 0.5),
 seq(prop_lambda,
prop_lambda+12, 2.5)), digits = 2))))}
perc.comb2r1 <- foreach(lambda = seq_lambda, .combine = rbind)
%do% {
  voirie = simulate.voirie(i=i,win=win)
  data.frame(radius,lambda, percolates = replicate(REPLICATE,
  percolates.comb2r(union.find(simulate.utilisateurs(lambda = lambda,
  voirie=voirie, win = win)), win, radius)))
```

```
  }
  perc.result = rbind(perc.result, perc.comb2r1)
  }
proc.time() - pt
perc.result
perc.result.summary= ddply(perc.result, .(radius, lambda), summarize,
mean = mean(percolates))
perc.result.summary
#title_result= paste(path,"result_summary", ".csv", sep="")
#write.csv(perc.result, file = tittle_result, row.names=FALSE)
##########
## APPROXIMATION avec REGRESSION LOGISTIQUE BINAIRE COMB II
##########
result_lambda_critique = data.frame(radius=integer(),lambda_critique=integer()
for (radius in seq_radius){
  X1 = perc.result.summary[perc.result.summary$radius == radius,]
  # modèle logistique
  logistic.comb2.fit = glm(mean ~ lambda, family=binomial(logit), data = X1)
  summary(logistic.comb2.fit)
  X1$fit = logistic.comb2.fit$fitted
  # estimer le seuil de percolation
  lambda_crit.comb2 = - logistic.comb2.fit$coefficients[1] /
  logistic.comb2.fit$coefficients[2]
  #sauvegarder le lambda critique pour chaque radius
  result_lambda_critique[nrow(result_lambda_critique)+1,] = c(radius,
  lambda_crit.comb2)
  # tracer le graphique du seuil de percolation
  title= paste(path, "crit.pvt_", REPLICATE, "_", radius,".png",sep="")
  print(ggplot(X1, aes(lambda, mean)) +
          geom_point(size = 2) +
          geom_vline(xintercept = lambda_crit.comb2, color = "red") +
          geom_path(color = 'lightgreen') +
          geom_line(aes(x = lambda, y = fit), linetype = 1, col = 'blue') +
          annotate("text", label = sprintf("lambda_critique == %
          .5f",lambda_crit.comb2), x = (.06+lambda_crit.comb2), y = 1,
          parse = TRUE) +
```

```
          ggtitle('Seuil de Percolation MCOMB2') +
          labs(x=expression(lambda), y="Pr_lambda(percole)") +
          theme(
            plot.title = element_text(hjust =.5, face="bold",
            color = 'darkgreen', size = 14)))
  dev.copy(png, file =title, width=10, height=10, units="in", res=300)
  dev.off()
}
```

The Function of θ_0 PDT Method

```
# fenêtre d'observation
win <- 6
# L'intensité de ppp
i <- 34.69783
# Paramètres
gamma  <- (32*sqrt(i))/(3*pi)
seq_radius <- .3
#seq_radius <- seq(0.125, .175, .05)
# Nombre de tessellations
ns = 10
# Nombre de simulations
n = 30
# Lien de stockage
path = "C://Users//PZDB7620//Documents//Local//RESULTAT//THETA//"
##########
## VERIFIER la PERCOLATION
##########
## a. la fonction percole
Percole = function (df, top, bottom, left, right){
  reponse = F
  point_origine = 1; while (df[point_origine] != point_origine)
  point_origine <- df[point_origine]
  for (p in top){
    for (q in bottom){
      x = df[p]; while (df[x] != x) x <- df[x] # FIND
      y = df[q]; while (df[y] != y) y <- df[y]
```

```
      if ((x == y) & (x == point_origine)){
        return (reponse = T)
      }
    }
  }
  for (p2 in left){
    for (q2 in right){
      x2 = df[p2]; while (df[x2] != x2) x2 <- df[x2] # FIND
      y2 = df[q2]; while (df[y2] != y2) y2 <- df[y2]
      if ((x2 == y2) & (x2 == point_origine)){
        return (reponse = T)
      }
    }
  }
  return (reponse)
}
## b. la fonction va s'arreter quand il percole
Until_Percole = function (df, radius, users_df, line, top, bottom, left, right){
  weight <- rep(1,100000)
  #if (nrow(users_df) ==1)
  df[1] = 1
  index_users=2
  while (Percole(df, top, bottom, left, right) == FALSE) {
    #users_add <- as.data.frame(runifpointOnLines(1, line))
    # supprimer des points qui traversent de la fenêtre
    #users_add <- users_add[(users_add$x <= win & users_add$x >= 0 &
     users_add$y <= win & users_add$y >= 0),]
    # sélectionner les points en top et en bottom
    ifelse((users_df[index_users,]$y >= win-radius/2-0.5),
    top <- append(top, index_users),ifelse( (users_df[index_users,]$y
    <=(radius/2)+0.5), bottom <- append(bottom, index_users),"nothing"))
    ifelse((users_df[index_users,]$x >= win-radius/2-0.5), right
     <- append(right, index_users),ifelse( (users_df[index_users,]$x
     <=(radius/2)+0.5),
    left <- append(left, index_users),"nothing"))
    df[index_users]=index_users
```

```
    # calcule la distance
    #if(nrow(users_df)==1)next;
    distance = nndist(users_df[1:index_users,],k=1:10)[index_users,]
    V2 = nnwhich(users_df[1:index_users,], k=1:10)[index_users,]
    distance_points = as.data.frame(cbind(index_users, V2, distance))
    # préciser qu'il existe une arrête ssi la distance entre deux points
    (utilisateur) est plus petite que le rayon r.
    connected_nodes <- distance_points[which((distance_points[,3] <= radius)),]
    if (nrow(connected_nodes) != 0){
      for (i in 1:nrow(connected_nodes)){
        #df <- Union(df,connected_nodes[i,1], connected_nodes[i,2])
        p <- connected_nodes[i,1]
        q <- connected_nodes[i,2]
        x = df[p]; while (df[x] != x) x <- df[x] # FIND
        y = df[q]; while (df[y] != y) y <- df[y]
        if (x == y) next
        if (weight[x] < weight[y]) # UNION
        { df[x] = y; weight[y] <- weight[y] + weight[x]; }
        else{ df[y] = x; weight[x] <- weight[x] + weight[y]; }
      }
    }
    index_users=index_users+1
  }
  return(index_users)
}
## sauvegarder le resultat
result <- data.frame(si = integer(), radius= integer(),n_users_percole =
integer(), total_length_street = integer())
##########
## ESTIME la PROBABILITE de la PERCOLATION THETA
##########
pt = proc.time()
for (radius in seq_radius){
  for (s in 1:ns) {
    ### Algorihtme de fitting theta
    ## a. générer un graphique de Poisson-Voronoi
```

```
poisson_df <- as.data.frame(rpoispp(i, win = owin(c(0,win), c(0,win))))
# Poisson-Voronoi Tesselation
voirie <- deldir(poisson_df$x, poisson_df$y)
window <- owin(c(min(voirie$delsgs$x1, voirie$delsgs$x2),
max(voirie$delsgs$x1,
voirie$delsgs$x2)), c(min(voirie$delsgs$y1, voirie$delsgs$y2),
max(voirie$delsgs$y1, voirie$delsgs$y2)))
for (si in 1:n) {
  ## b. j <- 0 ; il n y a pas d'utilisateur; 100000 array =0
  df <- numeric(1000000)
  # commençons par dessiner les voiries que nous avons trouvées ci-dessus
  line <- psp(voirie$delsgs$x1, voirie$delsgs$y1, voirie$delsgs$x2,
  voirie$delsgs$y2, window = window)
  users_df <- as.data.frame(runifpointOnLines(1000000, line))
  users_df <- users_df[(users_df$x <= win-0.5 & users_df$x >= 0.5 &
  users_df$y <= win-0.5 & users_df$y >= 0.5),]
  row.names(users_df)<- seq(1:nrow(users_df))
  ## c. placer un point uniformément au hasard quelque part sur l'une des
  rues de s; j <- j + 1
  # placer les utilisateurs sur les voiries
  #users_df <- as.data.frame(runifpointOnLines(1, line))
  # supprimer des points qui traversent la fenêtre
  #users_df <- users_df[(users_df$x <= win & users_df$x >= 0 &
  users_df$y <= win & users_df$y >= 0),]
  top <- c()
  bottom <- c()
  left <- c()
  right <- c()
  nombre_users_percol <- Until_Percole(df, radius,users_df, line, top,
  bottom, left, right)
  ## d. Tant que Percolate est FALSE, ajoute un point uniformément
  au hasard quelque part et arrete quand il percole
  ## e. si Percolate est TRUE, calcule les probabilités de chaque n
  length_street =0
  for ( j in 1: length(voirie$delsgs$x1)) {
    x0 <- voirie$delsgs$x1[j]
```

```
y0 <- voirie$delsgs$y1[j]
x1 <- voirie$delsgs$x2[j]
y1 <- voirie$delsgs$y2[j]
if ((y0> win-0.5) & (y1> win-0.5) ) next;
if ((y0< 0.5) & (y1<0.5)) next;
if ((x0> win-0.5) & (x1> win-0.5) ) next;
if ((x0< 0.5) & (x1<0.5)) next;
if ((y0> win-0.5) & (y1<= win-0.5)){
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * ((((win-0.5) - y1)) / (y0 - y1))
}else if ((y1> win-0.5) & (y0<= win-0.5)) {
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * ((((win-0.5) - y0)) / (y1 - y0))
}else if ((y1< 0.5) & (y0>= 0.5)) {
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * (abs((((0.5) - y0)) / (y0 - y1)))
}else if ((y0< 0.5) & (y1>= 0.5)) {
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * (abs((((0.5) - y1)) / (y1 - y0)))
}else if ((x0> win-0.5) & (x1<= win-0.5)){
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * ((((win-0.5) - x1)) / (x0 - x1) )
}else if ((x1> win-0.5) & (x0<= win-0.5)) {
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * ((((win-0.5) - x0)) / (x1 - x0))
}else if ((x1< 0.5) & (x0>= 0.5)) {
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = ab[2,1] * (abs((((0.5) - x0)) / (x0 - x1)))
}else if ((x0< 0.5) & (x1>= 0.5)) {
  ab = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist =ab[2,1] *  (abs((((0.5) - x1)) / (x1 - x0)))
}else{
  dist = pairdist(rbind(c(x0, y0), c(x1, y1)))
  dist = dist[2,1]
}
length_street <- (length_street + dist)
```

```
      }
      result[nrow(result)+1,] <- c(s, radius, nombre_users_percol, length_street)
    }
  }
}
title = paste(path,"Resultat_", n , ".csv",sep="")
write.csv(result, file = title, row.names=FALSE)
proc.time() -pt
##########
## La PROBABILITE THETA
##########
# calcul pour chacun des radius
for (radius in seq_radius){
  #prop_lambda <- (4.51*(1/radius^2))/(pi*gamma)
  if (radius==0.475){
    seq_lambda <- sort(unique(abs(round(c(seq(0.30490-0.3,0.30490+0.5, 0.025),
    seq(0.30490-0.1,0.30490+0.1, .01)), digits = 2))))
  }else if (radius==0.425){
    seq_lambda <- sort(unique(abs(round(c(seq(0.38260-0.4,0.38260+0.4, 0.025),
    seq(0.38260-0.1,0.38260+0.1, .01)), digits = 2))))
  }else if (radius==0.375){
    seq_lambda <- sort(unique(abs(round(c(seq(0.48426-0.5,0.48426+0.5, 0.025),
    seq(0.48426-0.1,0.48426+0.1, .01)), digits = 2))))
  }else if (radius==0.325){
    seq_lambda <- sort(unique(abs(round(c(seq(0.64763-0.6,0.64763+0.6, 0.05),
    seq(0.64763-0.1,0.64763+0.1, .01)), digits = 2))))
  }else if (radius==0.3){
    seq_lambda <- seq(0, 1.7, .01)
  }else if (radius==0.275){
    seq_lambda <- sort(unique(abs(round(c(seq(0.90620-0.6,0.90620+0.6, 0.05),
    seq(0.90620-0.1,0.90620+0.1, .01)), digits = 2))))
  }else if (radius==0.225){
    seq_lambda <- sort(unique(abs(round(c(seq(1.36199-0.8,1.36199+0.8, 0.1),
    seq(1.36199-0.1,1.36199+0.1, .01)), digits = 2))))
  }else if (radius==0.175){
    seq_lambda <- sort(unique(abs(round(c(seq(2.27460-1,2.27460+1, 0.1),
```

```
  seq(2.27460-0.2,2.27460+0.2, .01)), digits = 2))))
}else if (radius==0.125){
  seq_lambda <- sort(unique(abs(round(c(seq(4.59880-1.5,4.59880+1.5, 0.1),
  seq(4.59880-0.2,4.59880+0.2, .01)), digits = 2))))
}else if (radius==0.075){
  seq_lambda <- sort(unique(abs(round(c(seq(14.34265-2,14.34265+2, 0.1),
  seq(14.34265-0.3,14.34265+0.3, .01)), digits = 2))))
}else if (radius==0.025){
  seq_lambda <- sort(unique(abs(round(c(seq(109.79710-4,109.79710+4, 0.2),
  seq(109.79710-0.5,109.79710+0.5, .01)), digits = 2))))
}
#le corps de la boucle
res = as.data.frame(result[which(result$radius==radius),])
# calculer la poisson distribution
for (lambda in seq_lambda) {
  pnormal <- c()
  for (t in 1: nrow(res)){
    Z <- (res[t,3]-0.5 - (res[t,4]*lambda)) / sqrt(res[t,4]*lambda)
    Prob_Z <- 1-pnorm(Z)
    pnormal <- rbind(pnormal, Prob_Z)
  }
  colnames(pnormal) <-lambda
  res <- cbind(res, pnormal)
}
# la moyenne de chaque n
Xbar <- aggregate(. ~ si, res[], mean)
# la probabilite de theta
theta <- data.frame(lambda = integer(), theta = integer())
index = 1
for (lambda in seq_lambda) {
  theta[index, 1] <- lambda
  theta[index, 2] <- ( 1/(nrow(Xbar)) * sum(Xbar[, index+4]) )
  index <- index + 1
  # enregistre le fichier
  title= paste(path,"Theta",radius,".csv",sep="")
  write.csv(theta, file =title, row.names=FALSE)
```

```
  }
  # la variance de chaque n
  variance <- data.frame(lambda=integer(), variance=integer())
  index = 1
  for (lambda in seq_lambda) {
    variance[index, 1] <- lambda
    variance[index, 2] <- var(Xbar[, index+3])
    index <- index + 1
    # enregistre le fichier
    title= paste(path,"Variance",radius,".csv",sep="")
    write.csv(variance, file =title, row.names=FALSE)
  }
  CI = cbind(theta, variance[,2], sqrt(variance[,2]))
  title = paste(path,"CI",radius,".csv",sep="")
  write.csv(CI, file = title, row.names=FALSE)
  # tracer theta en fonction de lambda
  print(ggplot(theta) +
          geom_line(aes(lambda, theta), size = 1, color = 'maroon') +
          #geom_point(aes(lambda, theta), size = 1, color ='darkgoldenrod1') +
          #geom_vline(xintercept = .78, color = "gold2", size = .7) +
          ggtitle("La Fonction de Theta") +
          labs(x=expression(lambda), y=expression(theta)) +
          theme(plot.title = element_text(hjust=.5, face ='bold',
          color ='darkgreen', size = 14)))
  title= paste(path,"Theta",radius,".png",sep="")
  dev.copy(png, file =title, width=10, height=10, units="in", res=300)
  dev.off()
}
```

The function of $\mu(\lambda)$ PDT Method

```
# fenêtre d'observation
win <- 5
# L'intensité de ppp
i <- 34.69783
# Paramètres
```

```
gamma  <- (32*sqrt(i))/(3*pi)
lambda <- .9
radius <- .3
# Paramètres seq
#seq_radius <- seq(0.025, 0.5, .05)
seq_radius <- 0.475
# Itération
REPLICATE = 100
# Lien de stockage
path = "C://Users//PZDB7620//Documents//Local//RESULTAT//SAUTS//"
###############
## SIMULATION DU PROCESSUS DE POISSON
###############
simulate.poisson = function (i, win, image = F ){
  poisson <- rpoispp(i, win = owin(c(0,win), c(0,win)))
  # modification dans une trame de données
  poisson_df <- as.data.frame(poisson)
  # identifier les points en bas
  coord_bottom = poisson_df[poisson_df$y<=1,]
  coord_bottom$y  = abs(coord_bottom$y-1) + win
  # ajouter des points
  addpoint_bott = coord_bottom
  # Combiner ces nouveaux points avec tous les points avant
  poisson_df = rbind (poisson_df, addpoint_bott)
  # identifier les points à gauche
  coord_left = poisson_df[poisson_df$x<=1,]
  coord_left$x  = abs(coord_left$x-1) + win
  # ajouter des points
  addpoint_left = coord_left
  # Combiner ces nouveaux points avec tous les points avant
  poisson_df = rbind (poisson_df, addpoint_left)
  # tracer Poissson p.p.par utilisation de la fonction ggplot
  if (image){
    print(ggplot(poisson_df) +
            geom_point(aes(x,y), size = 2, color ='darkgoldenrod1') +
            labs(x="", y="") +
```

```
            ggtitle("Poisson Point Process") +
            theme(plot.title = element_text(hjust=.5, face ='bold',
            color ='darkgreen', size = 14)))
    # créer un fichier avec le graphique
    title = paste(path,"pppd",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  attributes(poisson_df)$i=i
  attributes(poisson_df)$win=win
  return(poisson_df)
}
poisson_df = simulate.poisson(i, win, image = F)
###############
## POISSON-DELAUNAY TESSELATION POUR LE VOIRIE
###############
simulate.voirie = function (poisson_df = NA, i = NA, win = NA, image = F, include.poi
   if (is.na(poisson_df)){
    poisson_df = simulate.poisson(i, win)
  }
  # poisson-delaunay tesselation
  voirie <- deldir(poisson_df$x, poisson_df$y)
  # tracer Poisson-delaunay
  if (image){
    print(ggplot(data = poisson_df, aes(poisson_df$x, poisson_df$y)) +
            geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
            size = 1, linetype = 1, color ="darkblue", data = voirie$delsgs) +
            geom_point(fill = 'darkgoldenrod1', pch=21, size = 2,
            color = 'navyblue') +
            ggtitle('Poisson-delaunay Tesselation') +
            theme(
              plot.title = element_text(hjust =.5, face="bold",
              color = 'darkgreen', size = 14),
              panel.background = element_rect(fill = 'ivory1',
              colour = 'springgreen4'),
              axis.title.x = element_blank(), axis.title.y = element_blank()))
```

```
    # créer un fichier avec le graphique poisson-delaunay
    title = paste(path,"pvtd",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  if (image){
    print(ggplot(data=voirie$delsgs) +
            geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2), size = 1,
            linetype = 1, color= "darkblue") +
            ggtitle('Voirie : Poisson-delaunay Tesselation') +
            theme(
              plot.title = element_text(hjust =.5, face="bold",
              color = 'darkgreen', size = 14),
              panel.background = element_rect(fill = 'ivory1',
              colour = 'springgreen4'),
              axis.title.x = element_blank(), axis.title.y = element_blank()))
    # créer un fichier avec le graphique poisson-delaunay
    title = paste(path,"voiried",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  attributes(voirie)$i <- attributes(poisson_df)$i
  attributes(voirie)$win <- attributes(poisson_df)$win
  if (include.poisson_df){
    attributes(voirie)$poisson_df = poisson_df
  }
  return(voirie)
}
voirie = simulate.voirie(poisson_df = poisson_df, image = F)
###############
## PROCESSUS de COX POUR LES UTILISATEURS
###############
simulate.utilisateurs = function(lambda, voirie = NA, i = NA, win = NA,
image = F, include.voirie = F){
  if (is.na(voirie)){
```

```
  voirie = simulate.voirie(i = i, win = win)
}
# définir la fenêtre de la voirie
window <- owin(c(min(voirie$delsgs$x1,voirie$delsgs$x2), max(voirie$delsgs$x1,
voirie$delsgs$x2)), c(min(voirie$delsgs$y1,voirie$delsgs$y2), max(voirie$delsgs
$y1,voirie$delsgs$y2)))
# commençons par dessiner la voirie que nous avons trouvées ci-dessus
line <- psp(voirie$delsgs$x1, voirie$delsgs$y1, voirie$delsgs$x2,
voirie$delsgs$y2, window = window)
# placer les utilisateurs sur la voirie
users <- rpoisppOnLines(lambda, line)
# transformer en trame de données
users_df <- as.data.frame(users)
colnames(users_df) <- c('userx', 'usery')
# supprimer des points qui traversent la fenêtre
users_df <- users_df[(users_df$userx <= win+0.5 & users_df$userx >=
(0.5-(radius/2)) & users_df$usery <= win+0.5 & users_df$usery >=
(0.5-(radius/2))),]
# identifier les points en bas qui est inférieur au rayon/2
coord_bottom = users_df[users_df$usery<=0.5,]
coord_bottom$usery  = abs(coord_bottom$usery-0.5) + win+0.5
coord_left = users_df[users_df$userx<= 0.5,]
coord_left$userx  = abs(coord_left$userx-0.5) + win+0.5
# ajouter des points
addpoint_bott = coord_bottom
addpoint_left = coord_left
# Combiner ces nouveaux points avec tous les points avant
users_df = rbind (users_df, addpoint_bott, addpoint_left)
# tracer des utilisateurs sur la voirie
if (image){
  print(ggplot(data = voirie$delsgs) +
          geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2), size = 1,
          linetype = 1, color= "darkblue") +
          geom_point(aes(userx, usery), fill = 'red2', pch=21, size = 3,
          color = 'black', data = users_df)+
          #ggtitle('Utilisateur : Processus de Cox') +
```

```
            theme(
              plot.title = element_text(hjust =.5, face="bold",
              color = 'darkgreen', size = 14),
              panel.background = element_rect(fill = 'ivory1',
              colour = 'springgreen4'),
              axis.title.x = element_blank(), axis.title.y = element_blank()))
    # Pour créer un fichier avec le graphique.
    title = paste(path,"utilisateurd",".png",sep ="")
    dev.copy(png, file = title, width = 10, height = 10, unit = "in", res = 300)
    dev.off()
  }
  attributes(users_df)$i = attributes(voirie)$i
  attributes(users_df)$win = attributes(voirie)$win
  attributes(users_df)$lambda = lambda
  if (include.voirie){
    attributes(users_df)$voirie = voirie
  }
  return(users_df)
}
#users_df = simulate.utilisateurs(lambda = lambda, voirie = voirie, win = win,
image = F, include.voirie = T)
##########
## FLOYD - WARSHALL ALGORITHM
##########
# distance_points <- pairdist(users_df)
# D <- ifelse(distance_points > 3, Inf, distance_points)
# D <- ifelse(D <= 3, 1, D)
# diag(D) = 0
# Floyd.Warshall = function(d){
#   n = dim(d)[1]
#   for (k in 1:n){
#     for ( i in 1:n){
#       for ( j in 1:n){
#         d[i,j] = min(d[i,j], d[i,k] + d[k,j])
#       }
#     }
```

```
#   }
#   return(d)
# }
# DK = Floyd.Warshall(D)
###############
## l'ALGORITHME D'UNION FIND
###############
union.find = function(users_df){
  connected_nodes <- c(var1 = integer(), var2 = integer(), distance = integer())
  users <- c()
  weight <- c()
  distance_points <- pairdist(users_df)
  # data manipulation
  molten <- melt(distance_points)
  # préciser qu'il existe une arête ssi la distance entre deux points
  (utilisateurs) est plus petite que le rayon r.
  connected_nodes = molten[which((molten[,3] <= radius) & molten[,1]< molten[,2]),]
  # calculer le plus court chemin
  users <- seq(1, nrow(users_df))
  weight <- rep(1, nrow(users_df))
  if (!is.null(nrow(connected_nodes))){
    if (nrow(connected_nodes)!=0){
      for (index in seq(1, nrow(connected_nodes))) { # pour chaque connection
        p <- connected_nodes[index, 1] # Échantillons d'objets aléatoires
        q <- connected_nodes[index, 2]
        x <- users[p]; while (users[x] != x) x <- users[x] # FIND
        y <- users[q]; while (users[y] != y) y <- users[y]
        if (x == y) next
        if (weight[x] < weight[y]) {# UNION
          users[x] <- y; users[users==x] <- y; weight[y] <- weight[y] +
          weight[x]; }
        else{ users[y] <- x; users[users==y] <- x; weight[x] <- weight[x] +
        weight[y]; }
      }
    }
  }
```

```
  return(cbind(users, weight, users_df))
}
#users = union.find(users_df)
##########
## VERIFIER la PERCOLATION PDT
##########
percolates.sauts = function(users, win, radius){
  users_union <- users$users
  weight <- users$weight
  users_df <- as.data.frame(cbind(users$userx, users$usery))
  colnames(users_df) <- c("userx", "usery")
  distance_points <- pairdist(users_df)
  D <- ifelse(distance_points > radius, Inf, ifelse(distance_points == 0, 0, 1))
  D_floyd <- floyd(D)
  reponse <- F
  hops <- 0
  distance <-0
  top <- as.numeric(rownames(users_df[users_df$usery >= win+0.5,]))
  right <- as.numeric(rownames(users_df[users_df$userx >= win+0.5,]))
  bottom <- as.numeric(rownames(users_df[users_df$usery <= 0.5,]))
  left <- as.numeric(rownames(users_df[users_df$userx <= 0.5,]))
  for (p in top){
    for (q in bottom){
      x <- users_union[p]; while (users_union[x] != x) x <- users_union[x] # FIND
      y <- users_union[q]; while (users_union[y] != y) y <- users_union[y]
      if (x == y){
        reponse <- T
        break()
      }
    }
    if (x == y){
        break()
    }
  }
  # s'il n'y a pas de connection entre top et bottom, chercher pour left et right
  if (reponse != T){
```

```
  for (p in left){
    for (q in right){
    x <- users_union[p]; while (users_union[x] != x) x <- users_union[x] # FIND
    y <- users_union[q]; while (users_union[y] != y) y <- users_union[y]
      if (x == y){
      reponse <- T
      break()
      }
    }
      if (x == y){
      break()
      }
  }
}
if (reponse ==T){
  # calculer les distances
  distance_points <- pairdist(users_df)
  # sélectionner les noeuds qui sont dans la composante connexe infinie
  users_percole <- which(users_union==x, arr.ind <- TRUE)
  # sélectionner les noeuds de distance correspondant au noeud percole
  distance_points_perc <- distance_points[users_percole, users_percole]
  # calculer le plus court chemin
  matrix_dist <- ifelse(distance_points_perc > radius, Inf,
   ifelse(distance_points_perc == 0, 0, 1))
  D_floyd <- floyd(matrix_dist)
  # prendre les distances le plus élevé
  # pour les distances plus grandes que 4 km
  dist_perc_fin <- melt_dist_perc[which(melt_dist_perc$value>4),]
  distance <- dist_perc_fin$value
  # créer index matrice
  mid <- cbind(dist_perc_fin$Var1, dist_perc_fin$Var2)
  # algorithme Floyd - Warshall pour obtenir le nombre de sauts
  hops <- D_floyd[mid]
  # Proportionnalité à la distance euclidienne
  proportion <- hops/distance
  #calculer les moyennes
```

```
    distance <- mean(distance)
    hops <- mean(hops)
    proportion <- mean(proportion)
    result <- as.data.frame(cbind(hops, distance, proportion))
    colnames(result) = c("Nb de sauts","distance","proportion")
    return(result)
  }
  else{
    hops <- 0
    distance <- 0
    proportion <- 0
    result <- as.data.frame(cbind(hops,distance,proportion))

    colnames(result) <- c("Nb de sauts","distance","proportion")
    return(result)
  }
}
#percolates = percolates.sauts(users, win = win, radius = radius)
##########
## PAIRES de DEVICES CONNECTES
##########
# Essayer pour qq valeurs de lambda
pt = proc.time()
perc.result <- c(radius=integer(),lambda=integer(),Nb.de.sauts=integer(),
distance=integer(), proportion=integer())
for (radius in seq_radius){
  if (radius==0.475){
    seq_lambda <- seq(0.45, 1.30, 0.1)
  }else if (radius==0.425){
    seq_lambda <- seq(0.70, 1.50, 0.1)
  }else if (radius==0.375){
    seq_lambda <- seq(0.90, 1.80, 0.1)
  }else if (radius==0.325){
    seq_lambda <- seq(1.10, 2, 0.1)
  }else if (radius==0.3){
    seq_lambda <- seq(1.30, 2.10, 0.1)
```

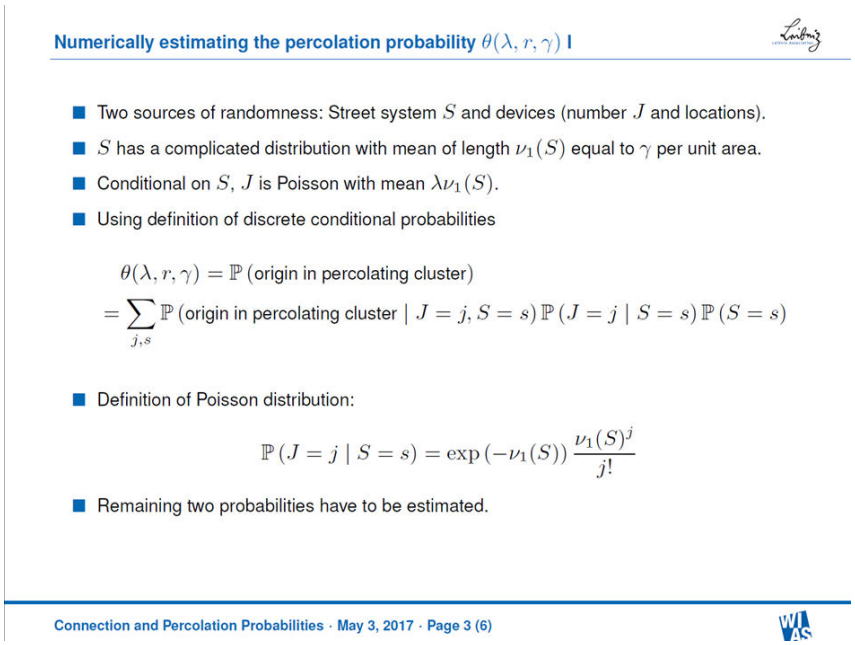
```
  }else if (radius==0.275){
    seq_lambda <- seq(1.80, 3.10 , 0.1)
  }else if (radius==0.225){
    seq_lambda <- seq(2.30, 3.50, 0.1)
  }else if (radius==0.175){
    seq_lambda <- seq(2.80, 4.10, 0.1)
  }else if (radius==0.125){
    seq_lambda <- seq(5.50, 6.50, 0.1)
  }else if (radius==0.075){
    seq_lambda <- seq(16, 17, 0.1)
  }else if (radius==0.025){
    seq_lambda <- seq(115, 130, 2)
  }
  # Chercher le seuil de percolation avec la fonction suivante :
  perc_hops = foreach(lambda = seq_lambda, .combine = rbind) %do% {
    voirie = simulate.voirie(i=i,win=win)
    data.frame(radius, lambda, do.call("rbind", replicate(REPLICATE,
    percolates.sauts(union.find(simulate.utilisateurs(lambda = lambda,
    voirie = voirie, win = win)), win, radius), simplify = FALSE)))
  }
  perc.result = rbind(perc.result, perc_hops)
}
# Summary
hops.summary = ddply(perc.result, .(radius, lambda), summarize,
Nb.de.sauts = mean(Nb.de.sauts), distance = mean(distance),
proportion = mean(proportion))
hops.summary = hops.summary[which(hops.summary$proportion!=0),]
title_hops = paste(path,"perc_hops2",".csv",sep ="")
title_sum = paste(path,"hops2.summary",".csv",sep ="")
write.csv(perc_hops, file = title_hops, row.names = FALSE)
write.csv(hops.summary, file = title_sum, row.names = FALSE)
#hops.summary = read.csv('hops2.summary_iter50.csv', header = TRUE, sep = ',')
for (radius in seq_radius) {
  hops.summary.radius = hops.summary[which(hops.summary$radius==radius),]
  # tracer la fonction de proportion
  print(ggplot(hops.summary.radius, aes(x = lambda, y = proportion, group = 1))+
```

```
          geom_line(size = 1, color = 'blue')+
          geom_point(size = 2, color ='violet') +
          ggtitle("La Fonction de Proportion PDT") +
          labs(x=expression(lambda), y=expression(mu)) +
          theme(plot.title = element_text(hjust=.5, face ='bold',
          color ='darkgreen', size = 14)))
  title = paste(path,"hops_",  REPLICATE, "_", radius, ".png",sep ="")
  dev.copy(png, file=title, width=10, height=10, units="in", res=300)
  dev.off()
}
```

Appendix B

Slide of WIAS

Slides of WIAS detail the simulation method of the function $\theta_0(\lambda)$:



Numerically estimating the percolation probability $\theta(\lambda, r, \gamma)$ I

- Two sources of randomness: Street system S and devices (number J and locations).
- S has a complicated distribution with mean of length $\nu_1(S)$ equal to γ per unit area.
- Conditional on S, J is Poisson with mean $\lambda\nu_1(S)$.
- Using definition of discrete conditional probabilities

$$\begin{aligned}&\theta(\lambda, r, \gamma) = \mathbb{P}\,(\text{origin in percolating cluster})\\ &= \sum_{j,s} \mathbb{P}\,(\text{origin in percolating cluster} \mid J = j, S = s)\,\mathbb{P}\,(J = j \mid S = s)\,\mathbb{P}\,(S = s)\end{aligned}$$

- Definition of Poisson distribution:

$$\mathbb{P}\,(J = j \mid S = s) = \exp\,(-\nu_1(S))\,\frac{\nu_1(S)^j}{j!}$$

- Remaining two probabilities have to be estimated.

Connection and Percolation Probabilities · May 3, 2017 · Page 3 (6)

Numerically estimating the percolation probability $\theta(\lambda, r, \gamma)$ II

$$\theta(\lambda, r, \gamma) = \mathbb{P}\,(\text{origin in percolating cluster})$$
$$= \sum_{j,s} \mathbb{P}\,(\text{origin in percolating cluster} \mid J = j, S = s)\, \mathbb{P}\,(J = j \mid S = s)\, \mathbb{P}\,(S = s)$$

- Definition of Poisson distribution:

$$\mathbb{P}\,(J = j \mid S = s) = \exp\left(-\nu_1(S)\right) \frac{\nu_1(S)^j}{j!}$$

- $\mathbb{P}\,(S = s)$: Generate street systems $s_1, \ldots, s_{n_S}$ independently at random and estimate probability as $1/n_S$.
- Efficient method for $\sum_j \mathbb{P}\,(\text{origin in percolating cluster} \mid J = j, S = s)\, \mathbb{P}\,(J = j \mid S = s)$ in Mertens & Moore, "Continuum percolation thresholds in two dimensions", Phys. Rev. E, 2012.

Numerically estimating the percolation probability $\theta(\lambda, r, \gamma)$ III

Efficient method for $\sum_j \mathbb{P}\,(\text{origin in percolating cluster} \mid J = j, S = s)\, \mathbb{P}\,(J = j \mid S = s)$ in Mertens & Moore, "Continuum percolation thresholds in two dimensions", Phys. Rev. E, 2012:

1. Generate a graph s, $i \leftarrow 0$
2. $j \leftarrow 0$.
3. Place a point uniformly at random somewhere on one of the streets of s, $j \leftarrow j + 1$.
4. Use the union–find algorithm to check for percolation.
   `www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf`
   `en.wikipedia.org/wiki/Disjoint-set_data_structure`
5. If percolation, $X_{i,s} \leftarrow \sum_{j' \geq j} \mathbb{P}\,(J = j' \mid S = s) = \sum_{j' \geq j} \exp\left(-\nu_1(s)\right) \frac{\nu_1(s)^{j'}}{j'!}$. (Approximation via normal distribution possible.)
6. Else go to 3.

Potentially useful to calculate several estimates for one s: $X_{1,s}, X_{2,s}, \ldots$ and take the average $\overline{X}_s$.

Numerically estimating the percolation probability $\theta(\lambda, r, \gamma)$ III

Efficient method for $\sum_j \mathbb{P}\left(\text{origin in percolating cluster} \mid J = j, S = s\right) \mathbb{P}\left(J = j \mid S = s\right)$ in Mertens & Moore, "Continuum percolation thresholds in two dimensions", Phys. Rev. E, 2012:

1. Generate a graph s, $i \leftarrow 0$
2. $j \leftarrow 0$.
3. Place a point uniformly at random somewhere on one of the streets of s, $j \leftarrow j + 1$.
4. Use the union–find algorithm to check for percolation.
   `www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf`
   `en.wikipedia.org/wiki/Disjoint-set_data_structure`
5. If percolation, $X_{i,s} \leftarrow \sum_{j' \geq j} \mathbb{P}\left(J = j' \mid S = s\right) = \sum_{j' \geq j} \exp\left(-\nu_1(s)\right) \frac{\nu_1(s)^{j'}}{j'!}$.
   (Approximation via normal distribution possible.)
6. Else go to 3.

Potentially useful to calculate several estimates for one s: $X_{1,s}, X_{2,s}, \ldots$ and take the average $\overline{X}_s$.

BIOGRAPHY

Nila Novita Gafur was born in Gorontalo, Indonesia on the 1st of November, 1991 as the first child of Gafur's family. In 1997-2003, author studied in primary school at SDN 3 Sukamakmur and in 2003-2006, author studied in junior high school at SMP Muhammadiyah 2 Tolangohula. Then, in 2006 to 2009, author studied in senior high school at SMAN 1 Boliyohuto.

Author is interested in mathematics and its application. Hence, author continued to take mathematic in the department of mathematics education at Univesitas Negeri Gorontalo in 2009 and graduated in 2013. Then, the author worked as a teacher in mathematics for junior and senior high school at Brilliant Brain for 3 months and as a private tutor for several months in 2013 to 2014. In 2014, the author continued to study in the department of statistics at Institut Teknologi Sepuluh Nopember (ITS), Surabaya. Then, for the academic year 2015-2017, author followed a Double Degree program in France at Université Pierre et Marie Curie (UPMC) or Sorbonne University, Paris. Author took same major, Statistics, but in the field of Data Science.

Alhamdulillah, in 2018, author has completed her master degree at ITS by collecting thesis entitled "*Fifth Generation (5G) Mobile Networks: Study of Percolation Threshold on Urban Road Models*", which is the thesis held at Orange Laboratory and UPMC.

The author can be contacted via electronic mail nila.novita.gafur@gmail.com