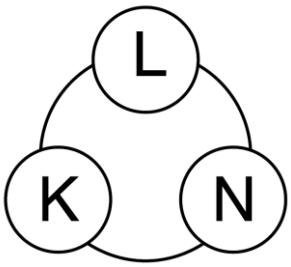

Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Bachelor‘s Thesis

Controller Placement Problem in Industrial Networks

Author:	Macián Ribera, Sergi
Address:	Clemenstrasse 127 80796 München Germany
Matriculation Number:	03679617
Supervisor:	Petra Stojsavljevic
Begin:	01. April 2016
End:	30. September 2016

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

München, 25.10. 2016

---	---
Place, Date	Signature

München, 25.10. 2016

---	---
Place, Date	Signature

Abstract

SDN (Software Defined Networking) architecture has become a trending topic nowadays due to it has been seen that this architecture satisfies the necessary requirements to be considered as a high capacity and reliable network by splitting the data and control plane. This control plane is performed by one or more controllers. An SDN controller is a software running on a host server, which is attached to one of the switches in case of in-band signalling. It has been proven that logically centralized architecture, provide really good results, e.g.., it can afford strict latency and reliability requirements, but it is still under study.

This thesis focuses on the primary problem that concerns SDN, where and how many controllers are needed to be deployed in a network. Generally, more than one controller is required because of reliability constraints. This problem is named Controller Placement Problem, also CPP, and there is not a unique answer to this question, since different criteria can be considered, such as switch to controller latency, load balance between the controllers, etc.

Moreover, the behaviour of industrial networks when using SDN architecture is studied. The motivation to apply SDN to industrial networks is to see how this architecture works with them, i.e., a study using SDN and industrial networks will be done in order to try to find a feasible solution for the CPP problem when using this kind of networks.

Another problem this thesis is focused on is how to decrease the elapsed time when evaluating the CPP problem for a network. An exhaustive evaluation of all possible controllers' placements is not realistic due to the fact that it can last several hours or even days and a fast response is needed in case that there is a failure. That is the reason why a metaheuristic algorithm named PSA is used and also studied. In this thesis, an algorithm that mixes Pareto Simulated Annealing (PSA) algorithm and Integer Linear Programming (ILP) problem is also studied in order to see if it can improve the way of computing the CPP problem making it faster and more precise.

Acknowledgments

In this section I am going to thank all staff who helped me with the development of this thesis and that has been close to me during this semester that I have lived abroad, in Munich.

First of all, I want to thank Petra Stojsavljevic, Carmen Mas and Jose Antonio Lázaro for helping and supporting me before and during the whole semester. Firstly, Jose Antonio helped me to contact TUM University and Carmen Mas and then, they agreed on my thesis project. During the development of the thesis, all my professors have been really attentive to me, i.e., they were always there for any doubts I had and they helped me to do my best.

I would also like to thank both Universities, TUM and UPC, and all institutions behind the Erasmus program for giving me the chance to study abroad during a semester. In my modest opinion, everyone should try this experience. It is a perfect way to open your eyes to new cultures, learn and improve languages and understand how other universities work.

Obviously, I really want to thank my family. It has always been there in all difficult moments, cheering me up and making this experience come true.

Finally, I also want to thank some friends from Barcelona like Jaume Baltasar for helping me with some doubts about SDN and making me laugh even from a large distance and Jordi Mañes for sharing with me some nice experiences during the Erasmus and helping me with the accommodation. From Munich, I have to thank Alba Luján for helping me with bureaucratic stuff and for keeping me motivated while studying together in the library.

Contents

Chapter 1 Introduction **7**

Chapter 2 Background **9**

2.1. Software-Defined Networking 9

2.2. Industrial networks 10

Chapter 3 Controller Placement Problem **13**

3.1. Introduction 13

3.2. State of the Art 13

3.3. Performance metrics 16

3.4. Pareto Frontier 19

Chapter 4 Integer Linear Programming **20**

4.1. Introduction 20

4.2. Problem formulations 20

Chapter 5 Pareto Simulated Annealing **22**

5.1. Introduction 22

5.2. State of the Art 22

5.3. Algorithm 24

5.4. Pareto Frontier Distances 28

Chapter 6 Implementation/Results **29**

6.1. Exhaustive evaluation 29

6.2. ILP 35

6.3. PSA 36

Chapter 7 Conclusions and Outlook 49

Chapter 8 Formatting 50

8.1. List of figures 50

8.2. List of Tables 52

8.3. Notation and Abreviations 53

8.4. References 54

Appendix A 55

Chapter 1

Introduction

Nowadays, IT technologies such as communication architectures are constantly improving. Operators and users are looking for more reliable and with higher bit rate networks, SDN (Software Defined Networking) architecture is being studied in order to achieve both goals. The main ideas of these networks are:

- To split data plane and control plane
- Decide which nodes perform the function of controllers.
- To connect each switch to one controller at least. This one will decide the optimal routing the packets from one switch to another one.

The problem that is tried to be solved in this study is to find the optimal controller placement, i.e., where and how many controllers have to be deployed in a network in order to ensure the fastest and most reliable communication. This problem is called CPP (Controller Placement Problem) and will be studied in this thesis.

When the optimal allocation placement for controllers is considered, there are several criteria that must be taken into account and that must be optimized. The controller placement performance metrics used in this thesis are average and worst-case latencies from controllers to other switches, average and worst-case latencies between controllers, controller's load and reliability. Some of them, such as switch to controller latency and load balancing cannot be always optimized at the same time. Hence, a trade-off between these metrics has to be always studied.

Some studies about controller placement have already been done. They have found a specific solution for a single metric or have mixed two of them and have found the Pareto-Frontier between these two metrics. This one determines when both metrics have the best values. Hence, if one metric is improved the other one will get worse and vice versa.

One of the purposes of this project is to continue studying SDN architecture, develop the code capable to compute the controller placement problem (CPP) and analyse the obtained results in order to find the optimal solution for different metrics at the same time. In this thesis, industrial topologies will be used. Our study compares their behaviour with the one in other typical networks, like the ones in database of operational wind parks [11].

Controller placement problem is NP-hard and it requires a huge amount of time –hours or even days- for a normal computer, a heuristic method is developed in order to reduce this time. However, some points in the Pareto-Frontier may not be computed. This metaheuristic evaluation is named PSA (Pareto Simulated Annealing) and will be explained later.

The second goal of this project is to study how distances between Pareto-Frontier computed with an exhaustive evaluation and Pareto-Frontier computed with PSA method vary in order to fine tune the PSA parameters, such as the number of iterations and the number of placements evaluated in each iteration. A new algorithm that combines PSA and ILP problem is also studied. Then, its results are compared with results from the commonly used PSA algorithm in order to see which algorithm is better.

This project has been carried out at the department of Electrical and Computer Engineering at Technische Universität München (TUM). The thesis will be developed as a part of a bigger project already started at TUM. It is called VirtuWind and its main goal is to study some offshore Wind Parks with a central station, named SCADA, where the communication system will be controlled by SDN architecture. Networks presented in VirtuWind [10] are considered industrial topologies and some of them will be used in this thesis. As it is said in the title of the thesis, it focuses on industrial networks. This kind of networks is later explained in Chapter 2.2.

This project is structured as follows: Chapter 2 presents the background information of the main topics of the thesis; Software Defined Networking and industrial networks. In Chapter 3, the CPP problem is described jointly with its SoA, the explanation of some metrics used in this thesis and a description of the meaning and computation of the Pareto Frontier. Then, in Chapter 4, Integer Linear Programming (ILP) problem is presented and its computation is explained. Chapter 5 presents PSA algorithm, in a brief introduction and SoA is explained why this metaheuristic algorithm is used. The code structure and evaluation methodology are presented. Then, Chapter 6 contains the obtained results in this thesis and in Chapter 7 the main conclusions and also future work are presented.

Motivation

This project has been developed with the objective of studying and improving SDN architecture technology, trying to solve its CPP problem obtaining the best relation between elapsed time and accuracy of the obtained values - distance between the Pareto Frontier obtained from metaheuristic algorithm and the Pareto Frontier obtained from an exhaustive evaluation -. PSA algorithm and ILP problem will be studied together and by separate in order to achieve this goal. The main motivation is to find a good algorithm that can be later overextended and implemented in real networks.

Chapter 2

Background

2.1. Software-Defined Networking

Nowadays, network architectures have a control plane and a data plane working on the same level, i.e., all nodes can control where and how to send packets at the same time that they are forwarding them. This means that each switch has its own control logic. Due to increasing traffic and flexibility demands, new network architectures are required. SDN is an architecture that is still under study, but it seems to be the solution for these problems.

In SDN architecture, control and data planes are decoupled. The control plane is performed by one or more controllers. These SDN controllers are a software running on a host server, which is attached to one of the switches in case of in-band signalling. They have an overview of the network and they can decide for other switches the routes they have to forward and communicate it through OpenFlow Protocol. That's why SDN is considered as a logically centralized architecture. All deployed nodes in the network, which can contain a controller or not, are also called switches.

In the following figure 2.1 [12], a legacy switch is shown on the left and a SDN switch is shown on the right. The legacy switch decides the forwarding path and also forwards the data, i.e., it executes control plane and data plane functions. On the other hand, the externally controlled switch - a SDN switch - just forwards data, but does not decide the route.

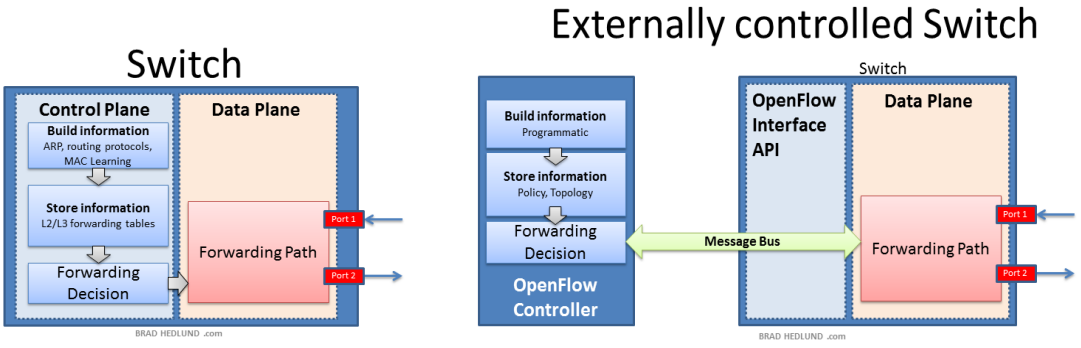

Figure 2.1 Traditional Switch vs SDN Switch [12]

A switch cannot send any packet until it receives the flow table provided by the controller. The larger latency is the time that the receiver is waiting for this flow table. Therefore, the propagation delay, proportional to the distance, between a switch and a controller is considered as the latency of the network.

SDN is known to be really fast and reliable compared with other architectures but it is still under study due to there are a lot of networks, metrics and variables that can modify the optimal controller placement and number of controllers. There is not a unique solution for the Controller Placement Problem, also known as CPP, i.e. there are different placements where controllers can be deployed, there is a trade-off between some metrics, named Pareto Frontier, which will be later studied in this thesis.

In the figure 2.2 an SDN architecture is shown. There are some switches connected to two different controllers. These two are also connected and have a general view of the network. Then, they can decide the best routes for the switches. As it is illustrated in figure 2.2 the communication protocol between controllers and switches that do not have a controller, is OpenFlow protocol [13].

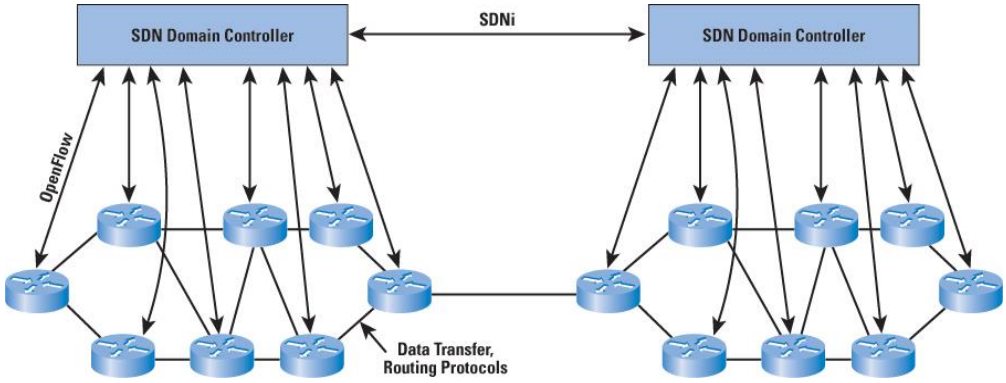

Figure 2.2 Example of SDN architecture [13].

2.2. Industrial networks

In this study, Industrial topologies, such as the ones presented in VirtuWind [10] are studied. These topologies are used in operational wind parks.

Industrial topologies present some special characteristics. They are normally well-distributed, i.e. nodes are distributed in a particular shape that can be classified in 4 big groups: Tree, ring, grid and Mesh. In this project, all of them are studied and they all have a central node named Supervisory Control and Data Acquisition (SCADA), which works as the control centre.

Industrial topologies also have strict and special QoS requirements; the response time must be fast and the reliability has to be five nines or better. E.g., as it is presented in table 2.1[10], the reliability requirements depend on the application. Latency values are obtained from IEEE 1646 standard [14] and reliability and packet loss rate are obtained from Energy Vertical Sector [15].

Service	Latency	Reliability	Packet Loss Rate
Analogue measurements	16 ms	99.99 %	< 10 – 6
Status information	16 ms	99.99 %	< 10 – 6
Protection traffic	4 ms	99.999 %	< 10 – 9

Table 2.1 Example of requirements in an industrial network

The following networks, all obtained from database of operational wind parks, are the ones thesis has focused on:

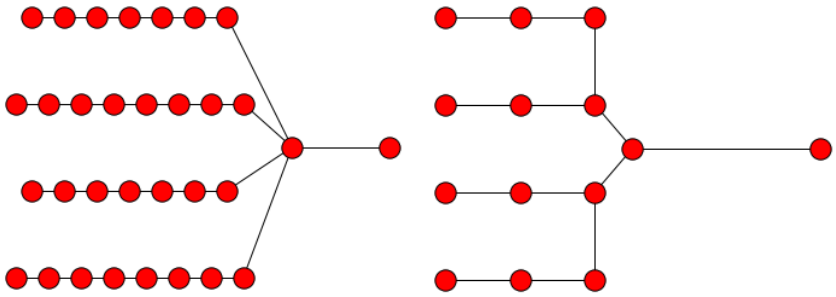

Figure 2.3 Barrow and AlphaVentus are considered as Tree Topologies [10]

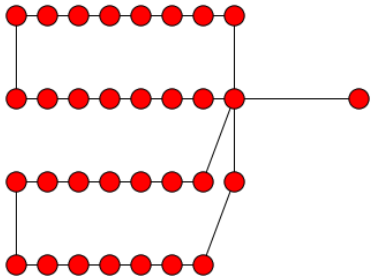

Figure 2.4 Ormonde is a Ring Topology [10]

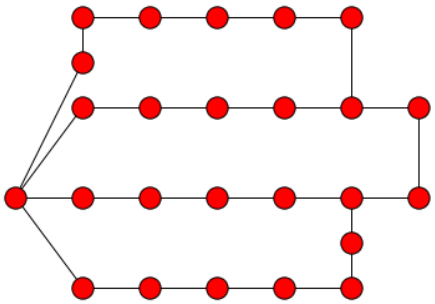

Figure 2.5 ThornonBank2 is considered as a Tree + Ring topology [10]

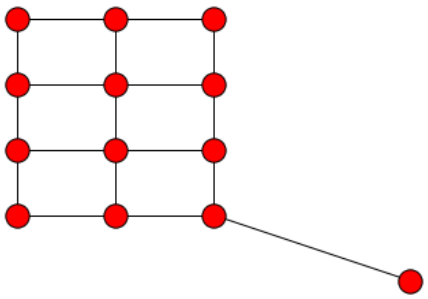

Figure 2.6 AlphaVentus2 is the classical grid topology [10]

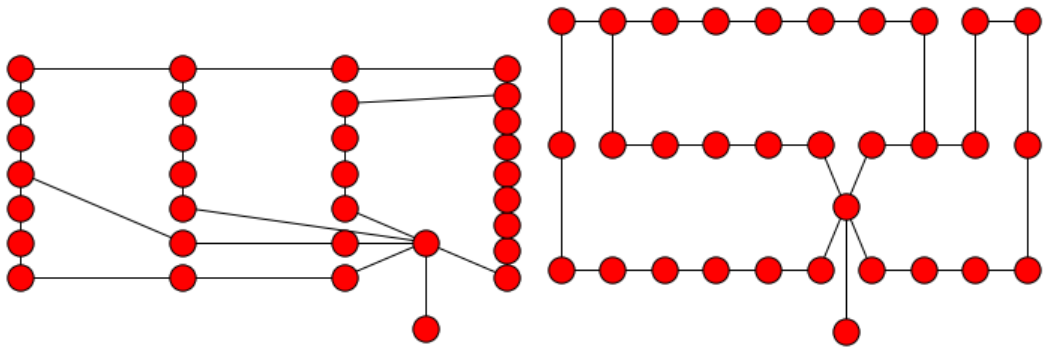

Figure 2.7 ThorntonBank1 and Riffgat are Meshed topologies [10]

The isolated node is named ‘SCADA’. It works as the central station in the wind park and as the control station where all information is sent and processed.

In some cases, like in the wind park topologies this thesis works with, links between nodes are constrained by the position of power cables. These ones are connected in a specific way that reduces power losses, but they do not necessarily provide an optimal distribution for communication links. I.e., in some industrial topologies, communication links are not always deployed in the optimal way, sometimes they have to be adapted to the distribution of the power cables.

In some cases during the thesis, the obtained results using the Industrial Networks defined previously will be compared with the results obtained from a European topology named cost266, formed by 37 nodes and 57 links and that connects the main cities in Europe. It is provided by Zoo Topologies [11].

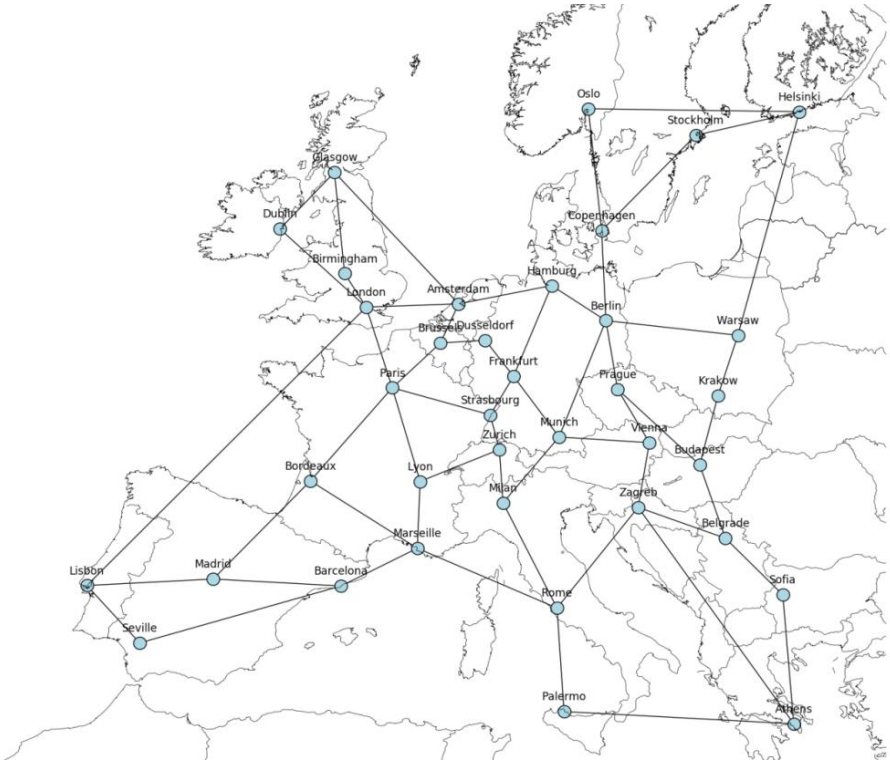

Figure 2.8 Cost266 topology

Chapter 3 Controller Placement Problem

3.1. Introduction

The main problem that this thesis is solving is the CPP (Controller Placement Problem). SDN networks split data and control paths. These control paths are handled by controllers deployed in some nodes of the same network. But here is where the problem appears: how many controllers are needed and where they have to be deployed.

Before talking about the CPP problem that is studied in this thesis, firstly all variables that take place in the calculus are introduced. This notation comes from [1] and it is also used in other papers like [2] and [3].

The network can be described as a graph named *G(V,E)* where *E* is the set of edges and *V* is the set of nodes. These nodes will be used as controllers and switches; *k* denotes the number of controllers that are deployed in the network and *n* denotes the total amount of nodes contained in a network. Matrix *d(i,j)* contains the shortest path distances from node *i* to node *j* and matrix *a(i,j)* contains the availabilities from node *i* to node *j* using the shortest path, i.e., the probability that a connection between *i* and *j* is working properly.

In order to know which allocations are the best ones, performance metrics that will be explained later are computed in all possible controllers' placements. Then, the obtained results are compared and the best controllers' placements are stored for a benchmark. Set *S* contains all the possible combinations of the controllers and the objective is to find a set *S'* that contains the best controllers' allocations. On now on, *S'* will be used as a particular subset of *S*. The total amount of evaluated placements, for which the metrics are computed, can be calculated with a factorial combination:

$$number\ evaluated\ placements \rightarrow \quad \binom{n}{k} = \frac{n!}{k!\,(n-k)!} \qquad (3.1)$$

3.2. State of the Art

In [1], the Controller Placement Problem is firstly introduced. It explains which are the traditional architectures used like BGP, OSPF and IS-IS and it says that their way of working is peer-to-peer. I.e., each device makes autonomous decisions hearing events from its peers. Hence, each forwarding device is working as its own controller. In [1] it is said that OpenFlow is the communication protocol used in SDN architecture between nodes and controllers.

In [1], it is explained that what the Controller Placement Problem, also named CPP, is looking for is the optimal number of controllers and the optimal allocation for them. CPP considers several performance metrics and also the network topology. There are several metrics that can be considered like latencies, load balance and reliability.

In [1] only two metrics are used: Average Latency and Worst-case Latency. It is considered that the highest latency is the propagation delay between nodes, which depends proportionally to the distance between nodes. Then, the average distance and the largest distance between controllers and their assigned nodes are computed. However, in other documents more metrics are also evaluated and compared. In [1], failures, load balancing and latencies between controllers are not taken into account. In this thesis, a total of six metrics will be used including the ones in [1].

In [2] and [10], Inter-Controller Latencies are also taken into account. In this case latency is also defined as the distance between two controllers. Therefore it can be also computed by analysing the average and the worst-case distance between controllers. In this thesis, these two metrics are computed as well.

In [10], [2] and [5], load imbalance contains the difference between the number of nodes connected to the controller with a higher load and the ones connected to the controller with the lowest load. In other words, a zero load-imbalance means that all controllers have the same number of assigned nodes. In both [10] and [5] is considered that each node counts as one load, but as it is said in [2], it could be also considered that different nodes have different load depending on their requirements, e.g., nodes that control big cities may have a higher load than nodes that control small ones. In this thesis, load imbalance metric will be used as it is defined in expression (5) in document [5].

In [2] and [3] some more metrics are introduced and studied such as resilience and failure tolerance of a network. These new ones are studied jointly with latencies presented in [1]. These new metrics let evaluate a network in function of two possible failures; the first one considers that an edge or a node has an outage and then the whole distribution of the network varies. Hence, distances from one node to another one may increase or even become disconnected. The evaluated metric in this case is $\pi^{\text{controller-less}}$. In the second case, a controller stops working and then, all nodes are automatically reconnected to their second closest controller. In this second case, the network is not altered and the studied metric is latency.

In this thesis, failure scenarios are not considered. However, metric *availability* is used. It reflects the probability that a connection between two switches or a switch and a controller using the shortest path to connect them is operational, taking into account the probability of failure of links and nodes that it has to traverse.

In [1], it first focuses on the best allocation for controllers for Internet2 OS3E network. The chosen placements will be those where the evaluated metrics are optimal, i.e., when metrics return a minimum value. In [1] the CPP problem is computed in two different ways; in the first one it chooses a random allocation for controllers and in the second it computes an exhaustive evaluation of all possible placements and it chooses the best one. Its conclusion is that if some controllers' placements are chosen randomly instead of being chosen through an exhaustive evaluation, both average latency and worst case

latency will increase too much. Therefore, controllers' placements cannot be selected randomly; a specific criterion must be used, in this case, an exhaustive evaluation.

In [2], a study with resilience performance metrics is done. When considering possible controller failures, it is concluded that if no failures are considered, controllers are spread around the network and if failures are considered, then controllers must be deployed closer.

Furthermore, another conclusion in [2] is that if controllers are deployed taking into account a possible controller failure, then the latency is worse than the one for free failures case. But, on the other hand, in a failure scenario this latency will be better than the case in which we didn't consider any failure.

In [1], there is also a study about the number of controllers. A study of average latency and worst-case latency when increasing the number of controllers is done for Internet2 OS3E network. Its results are shown in figure 3.1. It is shown that worst case latency and average latency values are reduced when more controllers are used. However, in [1] is concluded that this reduction is not proportional with the number of controllers *k*. I.e., the relation between controllers used and latency reduction is not constant. In fact, the gradient is reduced every time a controller is added. I.e., the more controllers are used, the less difference there will be with the previous value. E.g. in figure 3.1 is seen that if a second controller is added, average latency is reduced a 30%, but if a third controller is added average latency is just reduced almost a 10% more.

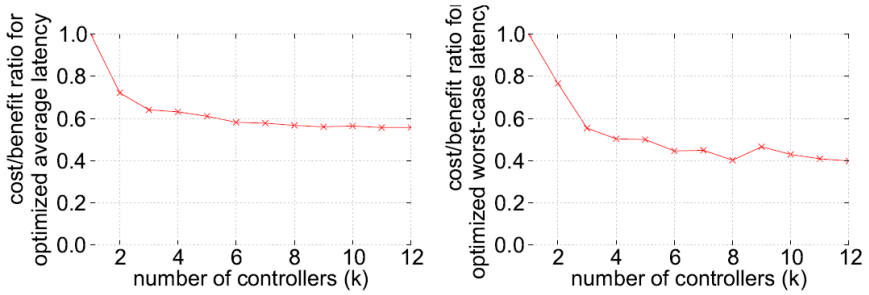

Figure 3.1 Latencies reduction when *k* is increased. Fig 4 in [1].

In [1] three time values –mesh restoration (200 ms), ring protection (50 ms) and switch processing (10 ms) - are fixed in order to see how many controllers are needed to be deployed to reduce average latency and worst-case latency below those thresholds. Each green line in figure 3.1 represents optimal average latency (left) or worst-case latency (right) value per each network when the number of controllers is increased. As it is seen in figure 3.1, in some cases one controller is enough due to with one controller, latencies are already bellow the most restrictive threshold. In this study, industrial networks are used instead of regional or wide area networks. This kind of networks have smaller distances, so it is supposed that if just Average Latency and Worst-case Latency were considered, one controller could be enough as well. However, this is not possible because, as it is shown in other documents such as [2] or [3], one controller is not enough due to a bad behaviour when evaluating other metrics.

In document [2], Zoo topologies are used and evaluated taking into account resilience and latency metrics. It is concluded that in order to reach the zero value for $\pi^{\text{controller-less}}$ metric when considering outages in a node, controller or edge, a high number of controllers must be deployed. Therefore, one controller is not enough when considering possible outages.

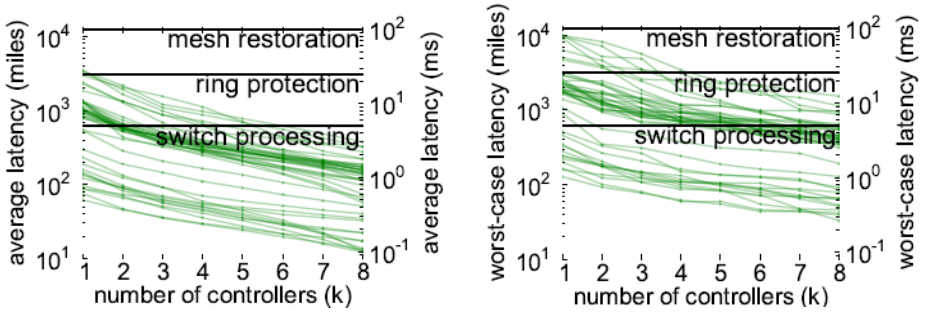

Figure 3.2 Latencies versus number of controllers. Figure 6 in [1].

The studies done in [3] and [4] take into account the reliability and it is concluded that in the large majority of topologies, the number of controllers that should be deployed is between *[0.035n - 0.117n]*. If there are not enough controllers used, there are longer paths between nodes and, therefore, more elements can fail in between. On the other hand, if there are a lot of controllers, there are more control paths. And then, there are also more paths that can fail.

The term Pareto Frontier is also used in [1]. It is described as a trade-off between two or more metrics. That means that a metric cannot be minimized in return for increasing another metric at the same time, i.e. when a metric is improved by choosing the best controller' allocation for it, another metric is getting worse.

The exact Pareto Frontier is obtained by an exhaustive search, but it can require a high time to compute it. That is why, in this thesis, a metaheuristic algorithm named Pareto Simulated Annealing is used. However, the Pareto Frontier obtained from this algorithm is just an estimation of the one obtained from an exhaustive evaluation. In this thesis is studied how this estimation differs from the real Pareto Frontiers and how it can be improved, i.e., how its accuracy can be optimized.

3.3. Performance metrics

As it is said in the State of the Art (SoA), there are a lot of metrics that can be considered in order to evaluate a controllers' placement. In this study, all those metrics that do not include node, link or controller failures are evaluated. I.e., metrics which take failures into account are not considered in this thesis. However, availability metric of the Network is also taken into account. Then, in this study, the following performance metrics that will be evaluated for each controller's placement:

- Average distance nodes-controllers
- Worst-case distance nodes-controllers
- Inter controller Average distance (controllers-controllers)
- Inter controller Worst-case distance (controllers-controllers)
- Load imbalance
- (un)availability

The objective is to minimize the obtained values for these metrics, with the exception of availability. For this reason in this thesis unavailability is used instead of availability.

It is considered that all nodes are connected to the closest controller and that there are not restrictions of number of nodes per controller, i.e., there is not maximum nor minimum load per controller.

Average switch to controller latency

First of all, the word latency has to be defined. It refers to the time interval between the source sending a packet and the destination receiving it. For SDN technologies, as it is said in [1], the most restrictive latency in this kind of communications is the propagation delay between nodes and controllers, which is proportional to the distance between nodes and controllers.

Then, for this metric the average distance between placed controllers and all nodes assigned to them is computed. Therefore, it has to be seen which nodes are connected to each controller and then obtain their distances.

In order to compute the average latency the following formula is used:

$$L_{avg}(S') = \frac{1}{n} \sum_{i \in V} \min_{(j \in S')} d(i,j) \qquad (3.2)$$

Worst case switch to control latency

The worst-case distance between controllers and nodes can also be computed. Then, it has to be seen which nodes are connected to each controller and obtain their distances. From all these distances, this metric focuses on the largest one, the worst-case latency. Its expression is:

$$L_{wc}(S') = \max_{(i \in V)} \min_{(j \in S')} d(i,j) \qquad (3.3)$$

Average latency inter-controllers

In SDN architectures the data and control paths are separated, i.e. there is data and a control plane. The control plane consists of a subnetwork created just by controllers. For this reason, the latencies between these controllers are also important. Therefore, the average distance and worst case distance between controllers must be also computed.

In order to compute this metric, *d* matrix must be modified. A new *d(ci,cj)* is created from d(i,j) that includes de shortest distance from controller c_i to controller c_j. It can be

created by removing all rows and columns from *d(ci,cj)* that do not contain a controller. Then, the following formula can be computed:

$$L_{av-IC} = \frac{1}{k} \sum_{c_i, c_j \in S'} d(c_i, c_j) \qquad (3.4)$$

Worst-case latency inter-controllers

In this metric the maximum distance that separates controllers is computed. First, the same procedure used for Average inter-controller Latency to obtain *d(ci,cj)* must be followed. But then, instead of computing the average distance from all controllers to other controllers, just the highest value is chosen. The regular formula to compute it is:

$$L_{wc-IC} = \max_{c_i, c_j \in S'} d(c_i, c_j) \qquad (3.5)$$

Load imbalance

In this metric, the difference between the maximum and minimum number of nodes connected to different controllers is computed, i.e. a 0 value would be obtained when all controllers have the same number of assigned nodes. As it is said in [16] the optimal situation is given when each controller has assigned between $\left\lfloor \frac{n}{k} \right\rfloor$ and $\left\lfloor \frac{n}{k} \right\rfloor + 1$ nodes. Considering n_c as the number of nodes assigned to the controller *c*, the Load Imbalance can be computed using the following formula:

$$\pi_{lb}(S') = \max_{c \in S'} n_c - \min_{c \in S'} n_c \qquad (3.6)$$

Availability

In this metric the probability of failure in the connection between a controller and the nodes assigned to it is calculated.

It is considered that every switch and edge that is used to communicate node A with node B has a probability of failure so, the more switches there are and the longer edges are, the higher probability of failure there will be. Firstly, it is computed matrix *a(i,j)* that contains the availability between node *i* and node *j*, using the shortest path to connect them. The node failure probability and link failure probability are represented with f_l and f_n, respectively.

In order to compute the availability between two nodes *i* and *j*, the following formula is used:

$$a(i,j) = a(j,i) = (1 - f_n)^{|path(i,j)|} = \prod_{x = path(i,j)} \left(1 - f_l \cdot d(x, x+1)\right) \qquad (3.7)$$

Where *path(i,j)* is a vector that contains all nodes that perform the shortest route between node *i* and *j*.

For a particular controllers' deployment, the value of the worst availability between a node and a controller is saved since it is the most restrictive. However, it is transformed to unavailability due to this is the metric that has to be minimized. Then:

$$\pi_{availability}(S') = \min_{i,j \in S'} a(i,j) \quad (3.8)$$

$$\pi_{unavailability}(S') = 1 - \pi_{availability}(S') \quad (3.9)$$

3.4. Pareto Frontier

The Pareto Frontier is made out one or more different points. If two metrics are considered it will be represented as a line and if three metrics are considered it will be represented as a plane. When two or more metrics are evaluated, normally a solution that satisfies all of them at the same time cannot be found. I.e., there is normally a trade-off between these two or more metrics. When one of them improves its value, the other one gets worse and vice-versa. As the objective is to minimize both metrics, values on the low-left side of the graphic are the important ones.

In [7], Pareto-Frontier is defined as:

"A placement *x* is considered Pareto optimal, if only if there is no placement *y* such that $\forall i\ f_i(y) \leq f_i(x)$ and $f_i(y) < f_i(x)$ for at least one *i.*" I.e., *x* is considered to be part of the Pareto-Frontier when there is not another placement *y* which its evaluated metrics $f_i(y)$ are lower or equal to $f_i(x)$, for at least one metric *i*. Then, the Pareto Frontier set is created by all these Pareto optimal placements.

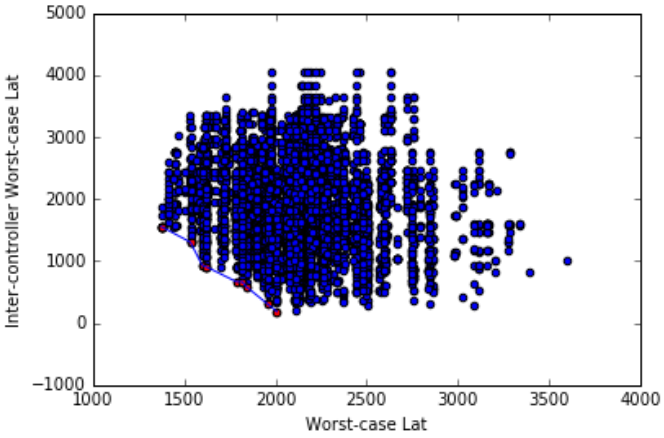

Figure 3.2 Example of Pareto Frontier

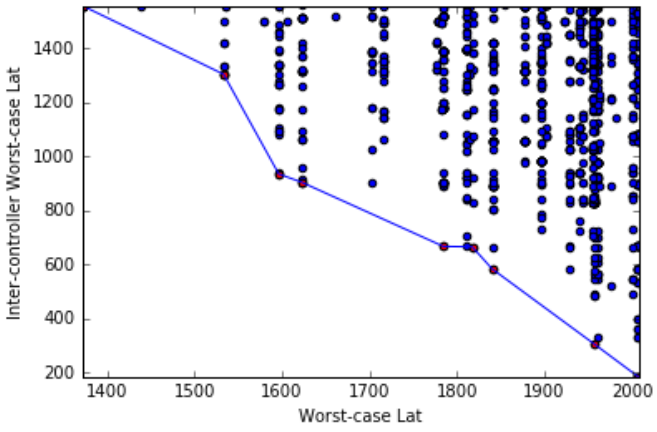

Figure 3.3 Example of Pareto Frontier (zoomed)

As it is seen in figures 3.2 and 3.3 all subset of controller's placements which values are in the Pareto Frontier have been colored in red and the blue line represents the Pareto Frontier.

Chapter 4 Integer Linear Programming

4.1. Introduction

In order to obtain the optimal value of a metric, all possible controllers' placements had to be computed and the best one had to be chosen, i.e., an exhaustive evaluation was computed. This method requires a lot of processing time due to in some networks there are a lot of possibilities to evaluate. For this reason, in this chapter Mixed Integer Linear Programming -MILP- problem is presented and defined.

On the one hand, this method can reach an optimal allocation consuming much less time than an exhaustive evaluation. But, on the other hand, it cannot compute a Pareto Frontier between two metrics. It will not be useful due to it can just provide the two limits that perform this trade-off known as Pareto Frontier, not obtaining the rest of points that are also included. Moreover, it can return an optimal placement that it is not even in the Pareto Frontier since there can be more than one optimal placement.

As it is explained later in chapter 6.1.2, there are just three metrics that must be taken into account: Average Latency, Inter Controller Average Latency and Load Imbalance. For each metric some variables, some constraints and, in our case, a function to minimize must be declared.

4.2. Problem formulations

Average Latency

The following expressions are all provided by [10] and with them an optimal controllers' allocation in our networks can be found. For the Average Latency the following expression 8 is used.

$$Minimize: \quad \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j} d_{i,j}$$

$$Subject\ to: \quad \sum_{j=1}^{n} x_{i,j} = 1, \forall i \in \{1,..n\} \qquad (4.1)$$

$$\sum_{j=1}^{n} y_j = k$$

$$x_{i,j} \leq y_{i,j}, \forall i,j \in \{1,..n\}$$

Where $x_{i,j}$ is 1 when node *i* is controlled by node *j*, otherwise its value is 0. Variable y_j is 1 when a controller is placed in node *j*. Finally, as it happens in CPP problem, $d_{i,j}$

contains the shortest distance between node *i* and node *j*, *n* is the total number of nodes and *k* is the number of controllers.

Inter-Controller Average Latency

Then, for optimizing the Inter-controller average latency the next expression can be used:

$$\begin{aligned} &Minimize: && \sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j d_{i,j} \\ &Subject\ to: && \sum_{j=1}^{n} x_{i,j} = 1, \forall i \in \{1,..n\} \\ & && \sum_{j=1}^{n} y_j = k \\ & && x_{i,j} \leq y_{i,j}, \forall i,j \in \{1,..n\} \end{aligned} \tag{4.2}$$

Load Imbalance

Finally, in order to compute the load imbalance between controllers it can be used:

$$\begin{aligned} &Minimize: && \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j} d_{i,j} \\ &Subject\ to: && \sum_{j=1}^{n} x_{i,j} = 1, \forall i \in \{1,..n\} \\ & && \sum_{j=1}^{n} y_j = k \\ & && x_{i,j} \leq y_{i,j}, \forall i,j \in \{1,..n\} \\ & && \sum_{i=1}^{n} x_{ij} \leq \left\lceil \frac{n}{k} \right\rceil, \forall j \in \{1,..n\} \end{aligned} \tag{4.3}$$

In this case, a maximum load value for the controllers is fixed. As it can be seen in the last expression, the maximum load per controller is $\left\lceil \frac{n}{k} \right\rceil$. E.g., if there are 32 nodes and 4 controllers, there would be a maximum of 8 controllers per node.

Chapter 5 Pareto Simulated Annealing

5.1. Introduction

When the CPP problem is solved using an exhaustive evaluation it is consuming a lot of time since there are a lot of possible controllers' placements to evaluate. As it is shown in the next table, when the number of controllers is increased, the required time increases exponentially.

As a lot of controllers' possible placements have to be evaluated, too much time is elapsed. In a real SDN network this time is not tolerable due to a fast response is required. That is the reason why a metaheuristic algorithm must be used. It will try to find the best controllers' placement computing just a small number of placements. I.e., the search space is reduced, so it spends less time.

5.2. State of the Art

In paper [3] different algorithms are compared; Random Placement, l-w-greedy, Simulated Annealing and Brute Force. Their conclusion is that Simulated Annealing (SA) is the he algorithm that obtains the best relation between elapsed time and distance from the exhaustive evaluation Pareto Frontier and the algorithm Pareto Frontier, named accuracy, i.e. in [3] it is concluded that Simulated Annealing is the best metaheuristic algorithm that can be used.

The objective in document [7] is to improve POCO, a program that computes an exhaustive evaluation, using a heuristic approach. PSA is the chosen and studied one. Firstly, the algorithm is introduced and it is explained how it works step by step, also including some formulas written in this thesis in Chapter 5.3. Hence, in our study, the PSA algorithm explained in [7] will be used and also some expressions from [8].

In order to compare and study the PSA algorithm, two variables named Relative Budget b^{rel} and Relative Time Consumption t^{rel} are defined in [7] as follows:

$$b^{rel} = \frac{b^{PSA}}{b^{POCO}} = \frac{m \cdot s \cdot \left\lceil -\frac{logTo}{\log p} \right\rceil}{\binom{n}{k}} \qquad (5.1)$$

$$t^{rel} = \frac{t^{PSA}}{t^{POCO}} \qquad (5.2)$$

Relative Budget and Relative Time Consumption compare number of evaluated placements and time between PSA algorithm and the exhaustive evaluation used in POCO. A conclusion obtained in [7] is that the larger size of the Search space in a network, i.e., the number of evaluated placements $\binom{n}{k}$ in an exhaustive evaluation, the more are reduced b^{rel} and t^{rel}. I.e., the relation between the number of evaluated placements when using PSA and when using exhaustive evaluation is lower when using large scale networks with high number of controllers and it happens the same with the relation of time.

In the following figure 5.1, 60 networks from Zoo Topologies have been evaluated with different number of controllers, obtaining different sizes for the Search Space. The PSA algorithm has been computed 40 times and if the accuracy of at least f_{min} % of repetitions are below the accuracy threshold (0.01 or 0.02), then the parameters of the PSA are accepted and the relative budget can be computed. If not, PSA parameters are increased in order to increase iterations in the PSA and reduce the accuracy. As it is seen in figure 5.1, the larger size of the Search Space, the smaller Relative Budget is needed. This result means that the bigger Search Space, the more profitable PSA is.

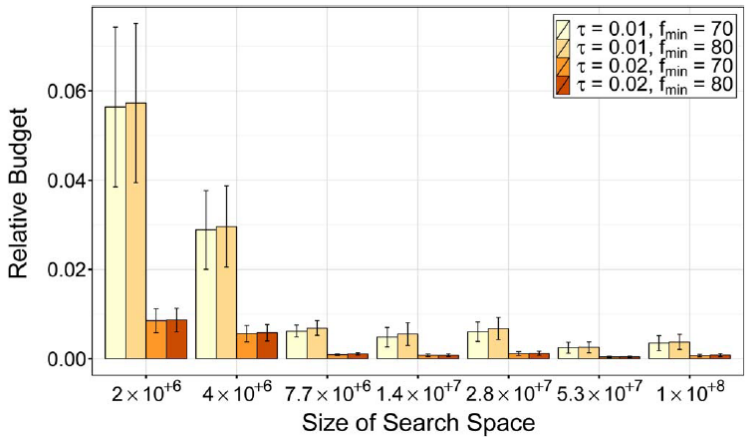

Figure 5.1 Relative budget for different Size of Search Space in order to achieve an accuracy level. Figure 5 in [7].

In [5], it is said that when large scale SDN networks are used, our networks often have to be adapted changing the controllers' position, and it has to be done quickly. The problem is that the elapsed time and memory requirements are too high if all metrics for all possible placements are evaluated, i.e. if an exhaustive evaluation is computed. In this paper [5], Pareto-Capacitated K-Metoids (PCKM) and SA are studied and compared. PCKM is a heuristic algorithm that improves two particular metrics; average latency and imbalance. On the other hand, PSA algorithm is not focused on any particular metric.

Its conclusions are that algorithms focused on particular metrics, like PCKM, work better than generic ones, like PSA, when simulating the Pareto Frontier for the metrics they are specialized for. However, this algorithm works worse for other metrics. As there are more metrics to compare, this thesis will consider the PSA instead of PCKM due to it computes several metrics.

5.3. Algorithm

In order to program a code capable to compute the PSA metaheuristic algorithm this thesis has followed steps used in [7] and [8]. In [7] PSA is presented as follows:

Algorithm: Pareto Simulated Annealing
1: **Input**: G = (V,E),k,s,m,To,p
2: n = |V|
3: S = generateRandomPlacements(n,s,k)
4: Λ = generateRandomWeights(S)
5: M = paretoFrontier(evaluatePlacements(S))
6: T = To
7: **while** T > 1 **do**
8: Y = drawNeighbours(S,n,$\lceil \frac{kT}{2To} \rceil$)
9: updateParetoFrontier(M,Y)
10: Λ = updateWeights(S)
11: S:= accept y∈ Y with probability P(S,Y,T, Λ)
12: **if** m iterations were performed at T
13: T = T ρ
14: **end if**
15: **end while**
16: **return** M and corresponding placements

Table 5.1 PSA Algorithm

Input values

There are several parameters that must be given to the PSA algorithm and that determine its way of working and will influence the obtained result. The first parameter that must be given is the network and the metrics that are going to be evaluated. Then, there are several PSA parameters that will be used in the performance of the algorithm:

- *s*: number of initial random placements from where PSA algorithm starts.
- *m*: number of iterations per temperature level.
- T_o*:* initial temperature level, it also controls the probability of accepting movements to worse values.
- *p*: temperature decrease with the number of iterations. It's a value ranged from 0 to 1 that reduces the temperature level every *m* iterations.

Depending on these parameters some characteristics can be increased or decreased like the running time, repetitions of the experiment, dispersion and probability of acceptance of worse values.

Generate random placements and compute random weights

The next step is to generate some random placements from where our code will start working and assign them some random weights. As explained in [8], in order to compute these random weights it just hast to be taken into account that they all have to

be positive and sum 1, i.e. $\forall j\ \lambda_j \geq 0\ and\ \sum_j \lambda_j = 1$. The random placements are saved in a variable named *S*.

Create the Pareto-Frontier

The next step is to evaluate the randomly generated placements. Then, the optimal values are added to the Pareto-Frontier. E.g., if *s*=3, three random placements are generated and then evaluated, those values that contain minimums will be added to the Pareto-Frontier. This Pareto-Frontier will be later updated with new obtained values for other placements. At the end of the PSA algorithm a Pareto-Frontier containing all the best evaluated placements is returned.

Inside the loop

Right after that, a loop that will be repeated as many times as we want depending on the variables *m*, *To* and *p* starts. The following expressions provided in [7] can be used to know how many times the loop will be computed and the maximum of different placements that will be evaluated:

$$number\ of\ loops = \left\lceil -\frac{\log T_o}{\log p} \right\rceil \qquad (5.3)$$

$$maximum\ evaluated\ placements = s \cdot m \cdot \left\lceil -\frac{\log T_o}{\log p} \right\rceil \qquad (5.4)$$

Generate random neighbours

In each loop iteration what is firstly done is to generate new random neighbours named *Y* for *S* placements. In this study, a new controllers' placement *Y* is considered neighbour of *S* if at least they share $kT/2T_o$ (rounding up) controllers as it is done in [7]. However, as it is said in [7], $k/2$ (rounding up) elements of difference could be also considered. Thus would increase the dispersion according to [7].

Update Pareto-Frontier

The following step is to evaluate these new placements, i.e., the neighbours *Y*, for the two metrics that are compared in this experiment and update the Pareto Frontier with the obtained results. In order to update it, old values in the Pareto Frontier and the new ones obtained from the evaluation are compared. If one of the two metrics has a lower value than other points that are actually in the Pareto Frontier, it is included as a new point. If the new value is better in both metrics than other old value in the Pareto Frontier, then this old value is removed from the Pareto Frontier.

Accept neighbours as new placements or not

If the new placements have better values than the old ones, they will be accepted, i.e. *S* will become *Y*. If they are worse, there is still a probability to accept *Y* placements. It is needed to be like that because if not, our evaluation could get stacked in a local minimum and probably the global minimum would never be reached as it is shown in

the next figure 5.1. After figure 5.1 it is explained how it is decided if a worse placement is accepted or not.

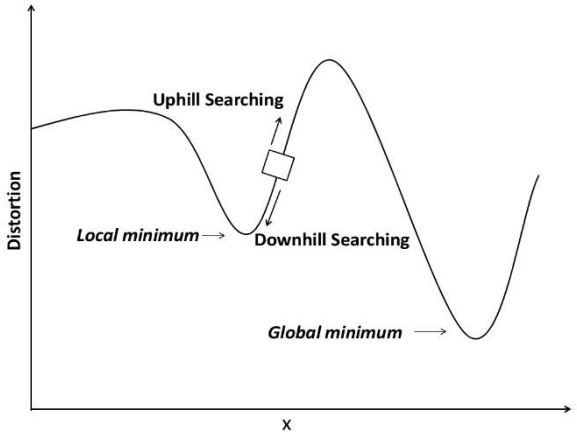

Figure 5.2 Accepting worse values to reach the global minimum

What is also repeated in the loop is an update of the weights. They are computed using an expression provided in [8], where $\lambda_j^{\boldsymbol{x}}$ is the previous weight and $\boldsymbol{x}$ contains the new values from the previous evaluation:

$$\lambda_j = \begin{cases} \alpha\, \lambda_j^{\boldsymbol{x}} & if \quad f_j(\boldsymbol{x}) \geq f_j(\boldsymbol{x}') \\ \lambda_j^{\boldsymbol{x}}/\alpha & if\ f_{\ j}(\boldsymbol{x}) < f_j(\boldsymbol{x}') \end{cases} \qquad (5.5)$$

In order to compute the probability of acceptance of placements which have a worse value than the previous one, the following formula provided in [8] must be used, where $\boldsymbol{y}$ are the new obtained values when Y is evaluated:

$$P(\boldsymbol{x}, \boldsymbol{y}, T, \Lambda) = \min\left\{1, \exp\left(\sum_{j=1}^{J} \lambda_j \left(f_j(\boldsymbol{x}) - \frac{f_j(\boldsymbol{y})}{T}\right)\right)\right\} \qquad (5.6)$$

Depending on probability P, the new placement Y will substitute placement S or not. It ranges from 0 to 1, where 1 means that there are 100% of chances to be changed. Probability P will be 1 when the new placement is better than the old one and will be between 0 and 1 depending on the obtained values, weights and the current temperature T.

Decrease temperature level

When m iterations have been computed with the same temperature level T, then this one is reduced by a p factor. The loop is repeated until the temperature level is equal or below 1.

Return

What is returned from the PSA algorithm is the last update of the Pareto Frontier. An estimation of the real Pareto Frontier that is obtained from an exhaustive evaluation.

Algorithm diagram:

In the following figure 5.3 is also explained how PSA algorithm works:

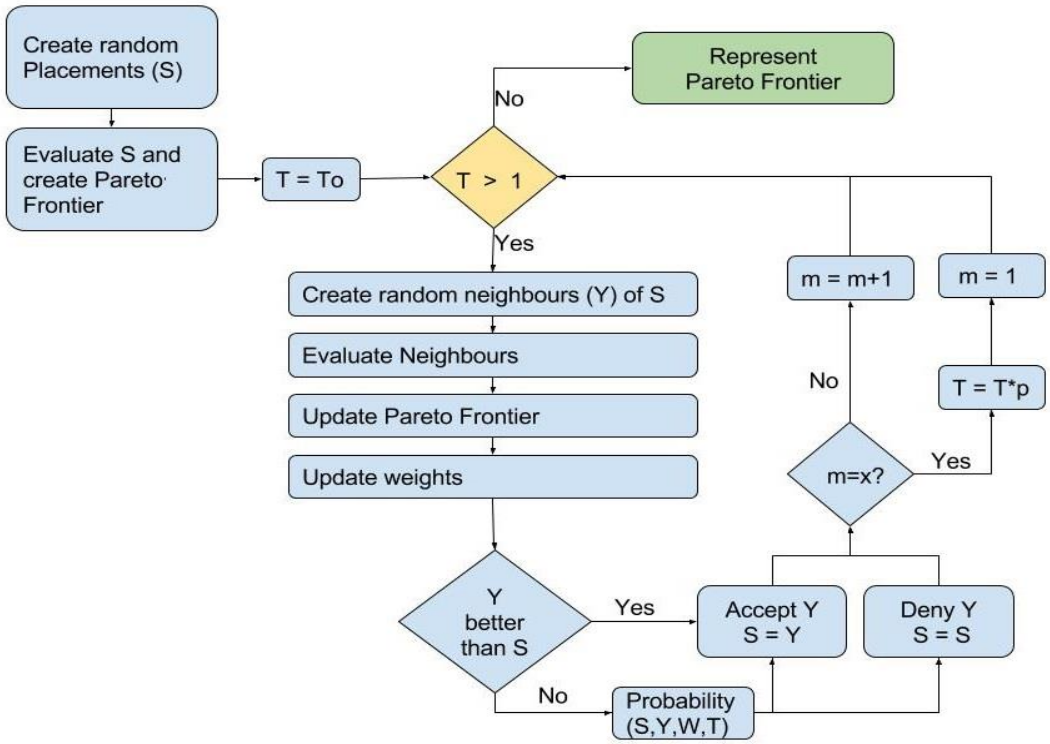

Figure 5.3 Representation of the PSA algorithm

5.4. Pareto Frontier Distances

In this study, the word accuracy will be treated as the distance between two Pareto-Frontiers. Generally, a Pareto Frontier obtained from an exhaustive evaluation, named *R*, and a Pareto Frontier obtained from PSA algorithm, named *M*. In document [7] some expressions are provided in order to compute accuracy. Firstly, distances between both Pareto Frontiers must be computed with the following expression:

$$c(x,y) = \max_{j=1,\dots,j}\{0, w_j(f_j(x) - f_j(y))\} \qquad (5.7)$$

In (5.7), w_j is the weight of each metric, i.e., the range between the maximum and the minimum value of each metric since *j* means each of the used metrics. Once computed the distances between Pareto Frontiers with 5.7, the average distance and the worst-case distance can be computed as follows:

$$\delta_{average}(R,M) = \frac{1}{|R|}\sum_{y\in R}\left\{\min_{x\in M}\{c(x,y)\}\right\} \qquad (5.8)$$

$$\delta_{worstcase}(R,M) = \max_{y\in R}\left\{\min_{x\in M}\{c(x,y)\}\right\} \qquad (5.9)$$

However, in this thesis just $\delta_{average}$ is computed and used.

Chapter 6

Implementation/Results

6.1. Exhaustive evaluation

In this first study, all 6 metrics described in chapter 3.3 have been computed for all possible controllers' placements varying the number of controllers *k* and the studied network. In order to compare the obtained results for those metrics, 2D Pareto Frontiers explained in chapter 2.1 have been computed. As there are 6 metrics and they are compared 2 by 2, there are a total of 15 possible combinations of metrics. The objective is to see how the behaviours of the Pareto Frontiers change for different networks and different *k*. As it is said in Chapter 3.1, there are a total of $\binom{n}{k}$ placements evaluated per network and per metric. In the following table, table 6.1, it is shown the total amount of placements computed per network.

Network	**number of nodes**	**k**				
		2	**3**	**4**	**5**	**7**
Barrow	32	496	4960	35960	201376	3365856
AlphaVentus	14	91	364	1001	2002	3432
Ormonde	32	496	4960	35960	201376	3365856
ThorntonBank2	25	300	2300	12650	53130	480700
AlphaVentus2	13	78	286	715	1287	1716
ThorntonBank1	32	496	4960	35960	201376	3365856
Riffgat	32	496	4960	35960	201376	3365856

Table 6.1 Number of evaluated placements per network and per *k*

As it can be seen, when *k* number is increased the number of evaluated placements also increases. In the next table, table 6.2, the total amount of evaluated placements per each *k* is shown, also taking into account the six metrics the study works with. I.e., shown values in 6.2 are a sum of the computed placements of all networks and then, this result is multiplied by 6 since there are 6 evaluated metrics.

	k				
	2	3	4	5	7
Number of evaluated placements	14718	136740	949236	5171538	83695632

Table 6.2 Total amount of evaluated placements per *k*

Taking into account that each computation requires a certain amount of time, it can be seen that the higher *k* value the code is working with, the higher elapsed time it will require.

Before starting the evaluation, availability performance parameters must be fixed. In this thesis, the following values have been used to compute availability metric:

Node failure probability f_n: 0.001 per node
Link failure probability f_l: 0.0001 per kilometre

Once all possible placements have been computed for all metrics and all networks, then the Pareto Frontier explained in chapter 3.4 can be computed. In order to select those values which are contained in the Pareto Frontier a short algorithm has been created and it works as follows:

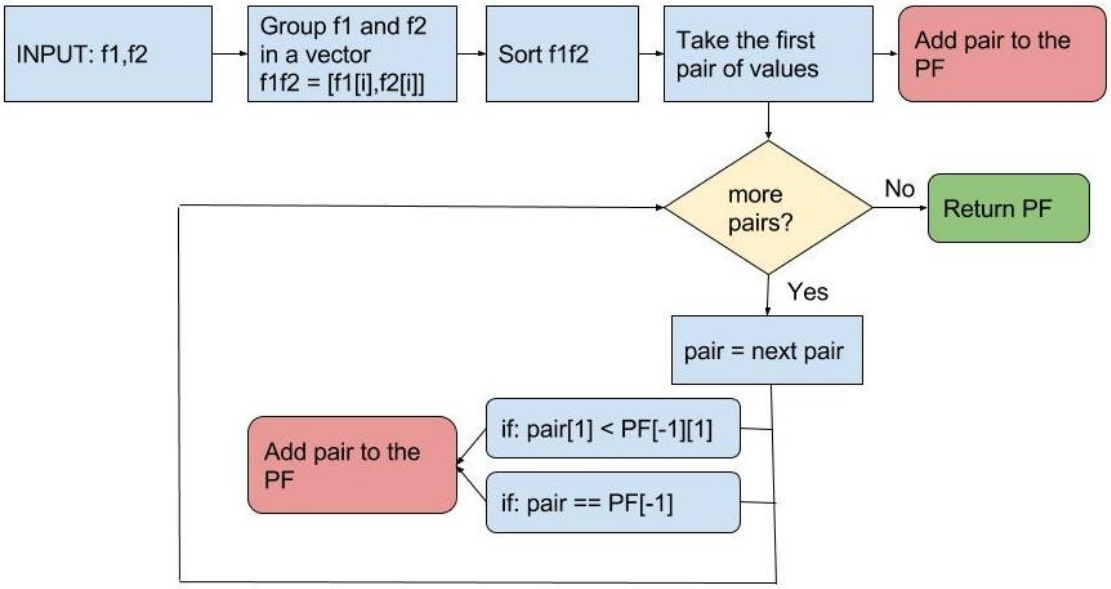

Figure 6.4 Algorithm that finds the Pareto optimal values

In the previous figure 3.4, all values previously obtained in a metrics' evaluation, e.g., an exhaustive evaluation, are grouped. Then, they are sorted and, therefore, the first value can be added to the Pareto Frontier set. Then a loop starts, for each pair of values, it is checked if the second value of the pair is smaller than the second value of the last Pareto Frontier solution. If it happens, this pair of values is added to the Pareto Frontier. Some pairs of values can be repeated in the Pareto Frontier, i.e., different controllers' placements can share pairs of values when they are evaluated.

From now on, in the plotted figures, all evaluated placements with their own values are represented in blue colour and obtained Pareto Frontiers are represented with a red line. In some cases, there is not a red line; it means that there is just a pair of optimal values and, therefore, the Pareto Frontier is just a single point instead of a line.

The elapsed times on the performance of the exhaustive evaluations have been saved and they are shown in table 6.3. The execution has been done by an Intel Core i7-3610QM CPU at 2,3GHz with 8 GB of RAM. The number of evaluated placements can be computed with expression 3.1 in chapter 3.1.

K	Elapsed time	Number of evaluated placements
2	1,96 s	666
3	22,199 s	7.770
4	206,21 s -> 3,5 min	66.045
5	1411,01 s -> 23,5 min	435.897
6	7439,3 s -> 2,06 h *	2.324.784
7	9,15 h *	10.295.472
8	34,31 h *	38.608.020

Table 6.3 Elapsed time for different number of controllers

-Values with an asterisk have been estimated.

In the following chapter 6.1.1 obtained Pareto-Frontiers from the exhaustive evaluation when varying number of controllers *k* are shown. Then, in 6.1.2 correlation between some metrics is discussed and some conclusions are achieved.

6.1.1 Impact of the number of controllers (k)

In this part of the study, the exhaustive evaluation is computed for each network and each pair of metrics modifying *k* value. The objective is to see how Pareto Frontiers variate when modifying this parameter. In the following presented results, elapsed time is not shown and not taken into account due to it will be studied later, when the thesis focuses on a metaheuristic algorithm called PSA.

As the images are too big to be plotted in this part of the document, they are fully plotted in Annex 1. Instead of showing the whole figure here, just three comparisons over fifteen are shown; worst-case Latency versus Average Latency, worst case Latency versus Inter-Controller worst-case latency and worst-case latency versus Inter-controller average Latency. X-axis always correspond to the first metric written on the left and Y-axis correspond to the second one. E.g., in the first line X-axis are Worst-case latency metric and Y-axis are average Latency metric. Moreover, in order to have a general idea, networks are plotted in the first line.

In the following figures, *k = (3, 5, 7)* has been used in order to compare the obtained results when using different number of controllers.

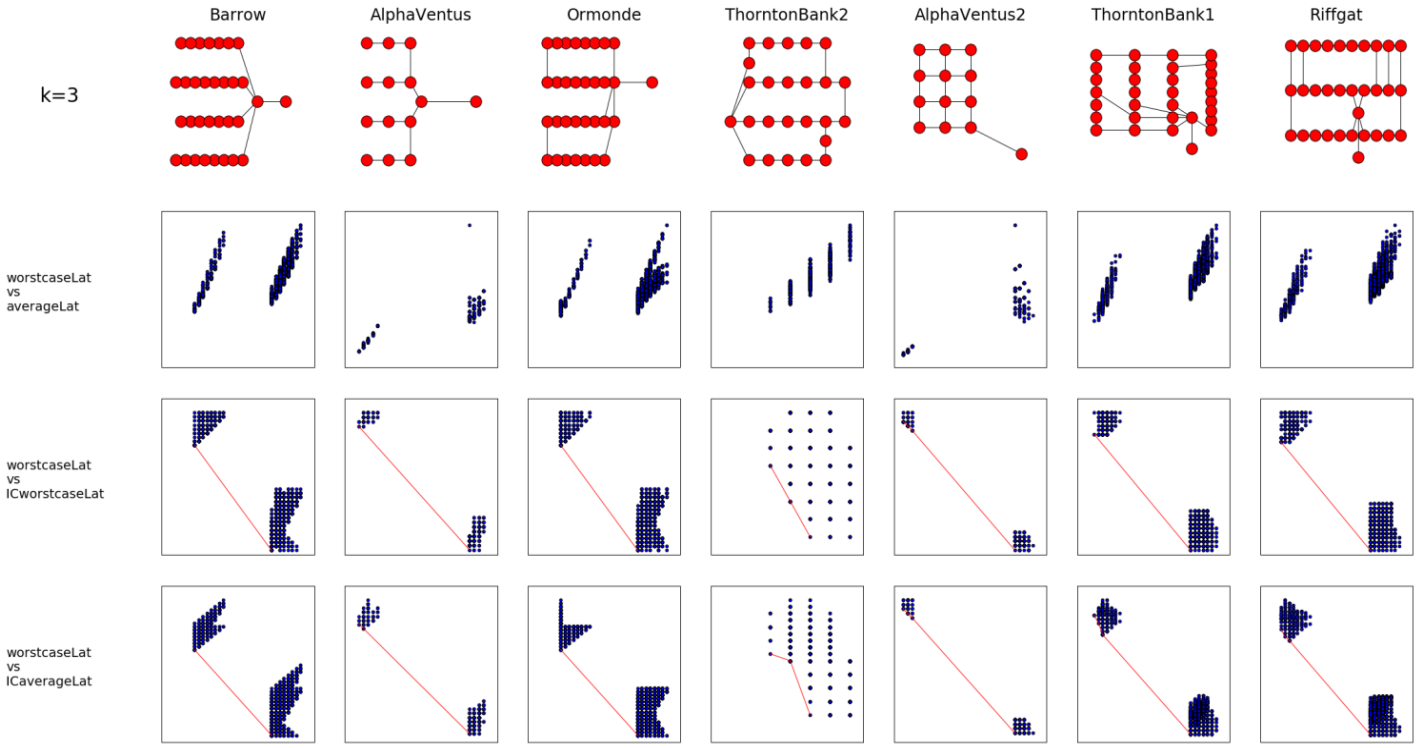

Figure 6.1 WC Lat vs Av Lat, WC Lat vs IC WC Lat and WC Lat vs IC Av Lat for k =3

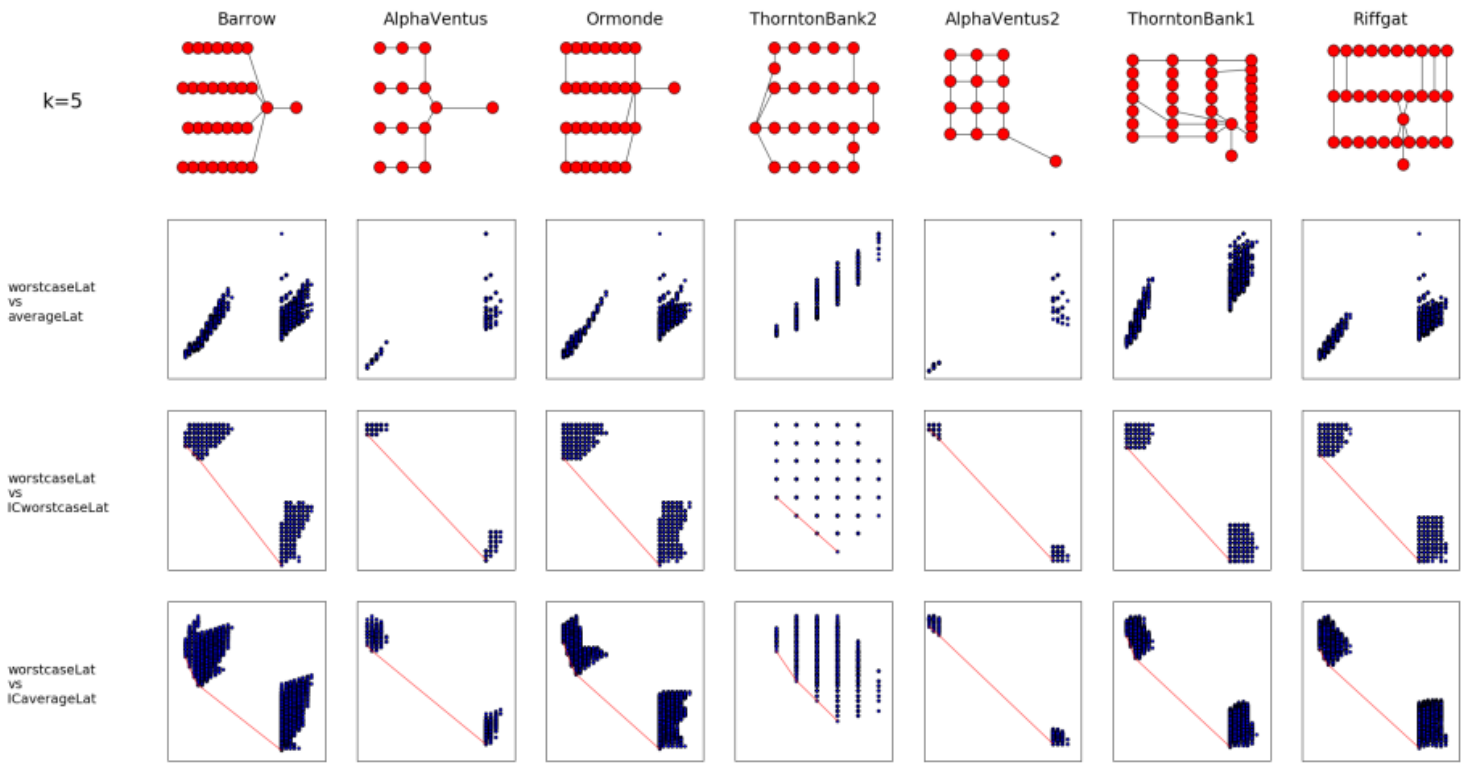

Figure 6.2 WC Lat vs Av Lat, WC Lat vs IC WC Lat and WC Lat vs IC Av Lat for k =5

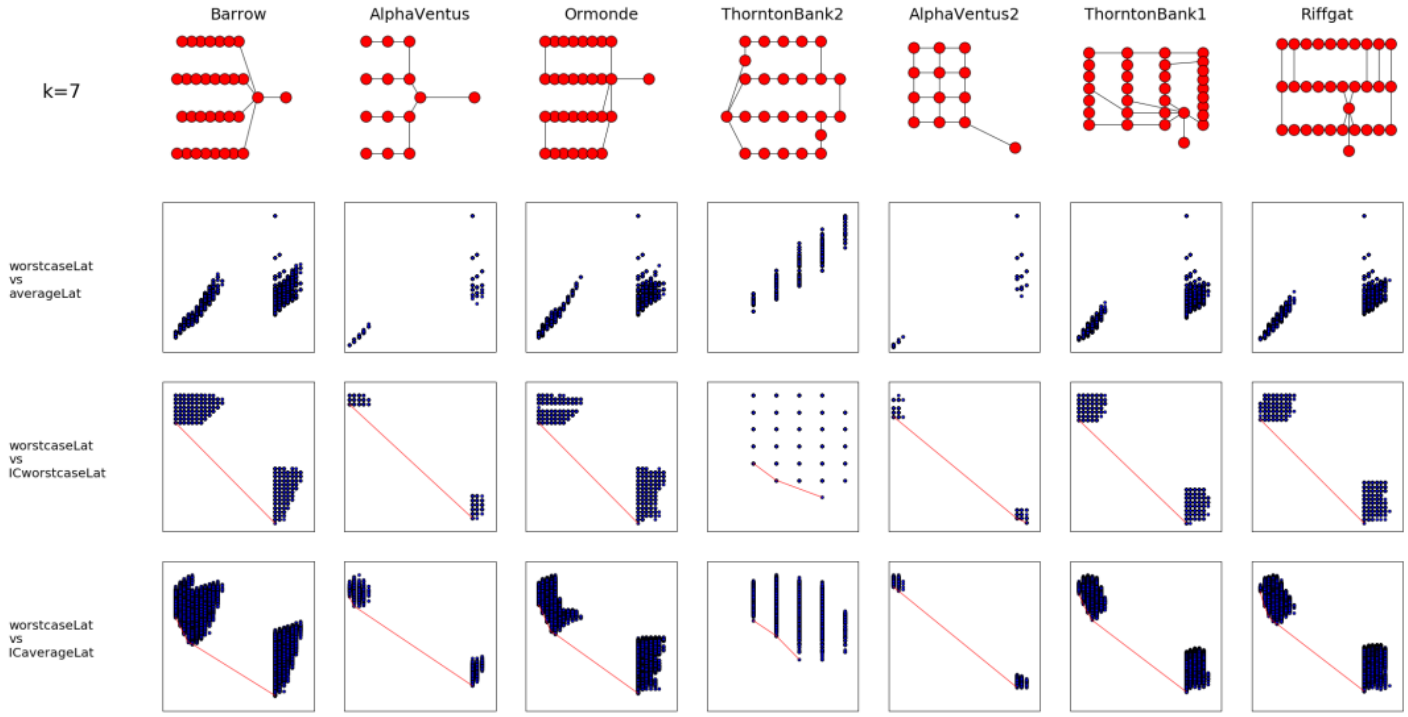

Figure 6.3 WC Lat vs Av Lat, WC Lat vs IC WC Lat and WC Lat vs IC Av Lat for k =7

What it can be seen in figures 6.1, 6.2 and 6.3 is that when the number of controllers *k* is increased, the shape of the Pareto Frontier remains being the same one. E.g., in worst-case Latency versus IC average Latency for Barrow topology, two big blocks of points appear in all three cases, *k*=3, 5 and 7. Each network has its own shape and size, but they are either separated by two big blocks or growing in a correlated way, excepting ThorntonBank2 that has just one block of values. As it can be seen in figure 1 in Annex 1, this behavior is repeated also for other metric comparisons, not just for the three comparisons shown above.

6.1.2 Correlation between different performance metrics

In this chapter another conclusion about the fifteen computed comparisons is explained; it is about correlation between metrics. All comparisons are plotted in Annex 1, figure 1, figure 2 and figure 3. As it can be seen, there are some metrics that have a correlated behaviour, i.e., they both improve in the same way. Therefore, they can be considered as 'equals' henceforth. In our study is considered that if a pair of metrics ends up with the same pair of values or a short Pareto-Frontier, both metrics are correlated.

In figure 6.4 represented bellow, both metrics share an optimal solution or they have a thick Pareto-Frontier. Moreover, it can be said that they are correlated due to they both grow in a linear way, i.e., when one metric gets worse, the other one gets worse too.

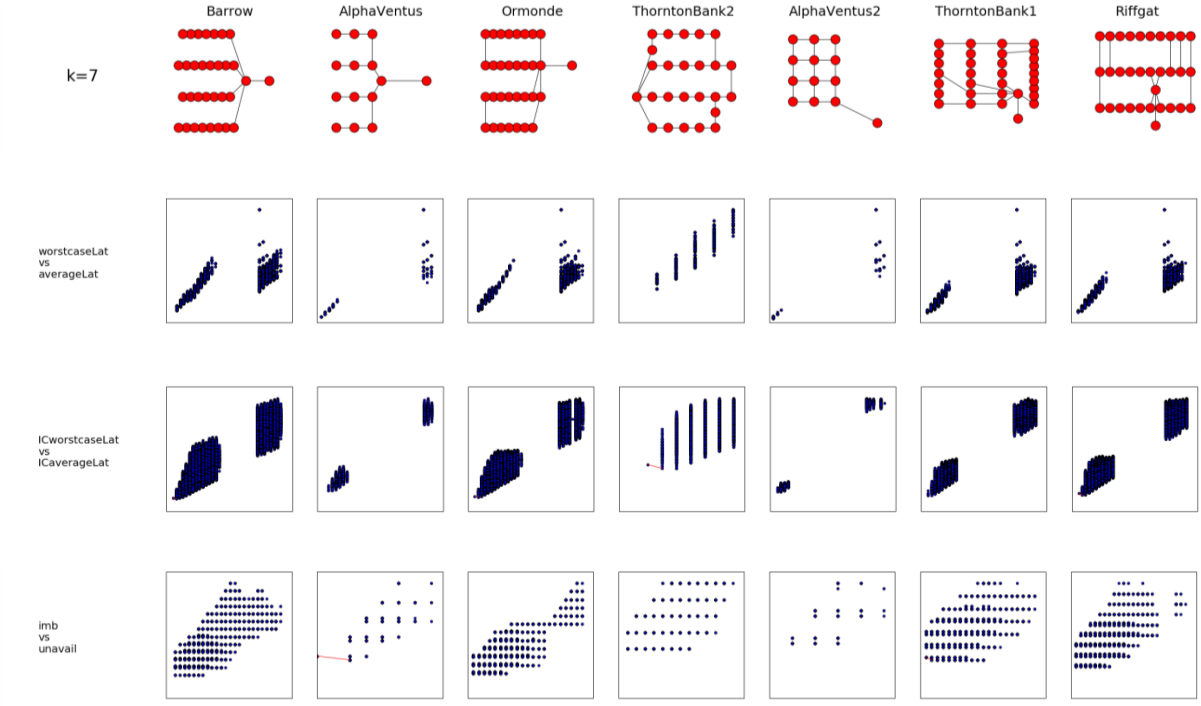

Figure 6.4 Correlated metrics

Conclusions obtained from the previous figure 6.4 are:

1) There are some pairs of metrics that clearly correlate for any number of controllers *k* and they have a similar behaviour when compared with other metrics. The correlated metrics are worst-case latency with average latency and inter-controller worst-case latency with inter-controller average latency. I.e., latencies between nodes and controllers and also from controllers to controllers are both correlated. Furthermore, imbalance and unavailability metrics are also correlated themselves.

2) Therefore, from now on, instead of considering 6 metrics, there will be considered just the following ones: average latency between nodes and controllers, average latency between controllers and load imbalance.

3) The correlation of these metrics is equal for all number of controllers, i.e. the correlation of these metrics has no dependence with k.

4) The correlation of these metrics is similar for all industrial networks this thesis is working with, i.e. the correlation of these metrics has no dependence with the network.

6.2. ILP

In this part of the thesis, results when computing ILP are shown. In order to show how the ILP method works, the code firstly evaluates all possible controllers' placements, i.e., an exhaustive evaluation, using three controllers and for all metrics; Average Latency, Inter Controller Latency and Imbalance. Then, ILP optimal placements are computed using Gurobi as a MILP solver and the explained expressions provided in chapter 4. All the code has been written in Python.

In the following figure, 6.5, just one example of two metrics, Average Latency and Inter-controller Average Latency, and for one network, ThorntornBank1, is shown. Blue points represent each pair of the metrics' values. The red line represents the Pareto-Frontier of the exhaustive evaluation and cyan dots represent the edges of this Pareto-Frontier. The red dot represents the ILP optimal placement when optimizing Average Latency and the yellow dot represents the ILP optimal placement when optimizing Inter-Controller Average Latency.

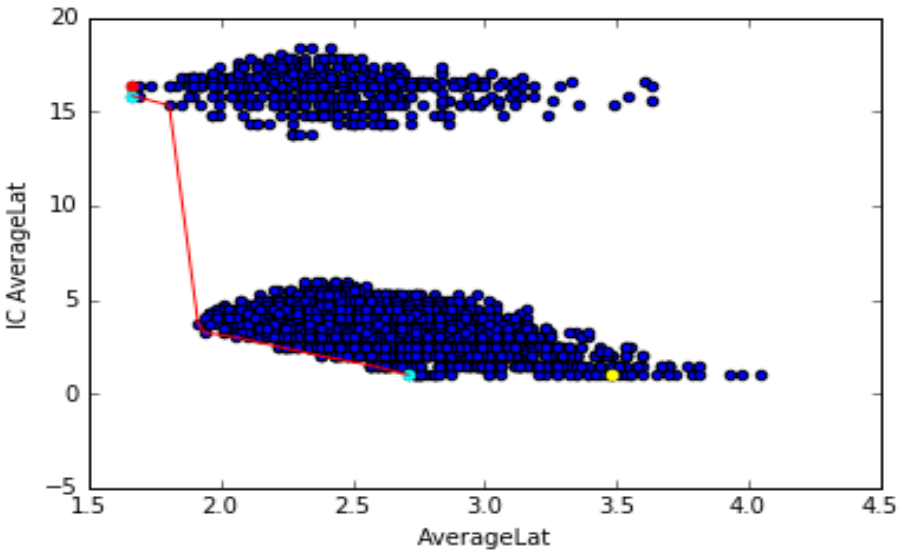

Figure 6.5 ILP method applied for Average Latency and IC Average Latency

As it is seen in figure 6.5, there are several optimal placements for each metric; all dots included in the lowest row for Inter-Controller Average Latency and all dots included in the leftmost column for Average Latency. However, there is just one dot included in the Pareto Frontier. It actually corresponds to one of the edges of the Pareto Frontier.

As there are several optimal placements for each metric, i.e., several controllers' placements that can optimize a metric, when ILP returns an optimal placement it does not have to be necessarily included in the Pareto-Frontier. As it is seen in figure 6.5, there is a certain distance between the yellow and cyan dots. It does not mean that the result is not correct; it just means that it will not always return the placement in the Pareto-Frontier edge. Later, when the metaheuristic algorithm PSA is studied jointly with ILP, this result can give unexpected problems.

6.3. PSA

As it can be seen in table 6.3 in chapter 6.1, elapsed times by exhaustive evaluations are too large and unfeasible for real networks since they need fast reaction times. In chapter 6.3.1, some PSA parameters will be tuned in order to see which one is better to be increased to reduce accuracy of the PSA Pareto Frontier with the real Pareto Frontier. In chapter 6.3.2 a new algorithm that combines PSA algorithm and ILP problem is studied in order to see if it is better or worse than the common PSA used in [7].

6.3.1 Tuning PSA parameters

In this chapter the accuracy of the PSA algorithm when modifying its parameters is studied. The goal is to see which of its values is worthier to increase. As it is explained in chapter 5.4, *m*, *s*, T_o and *p* are the parameters that can be modified. When any of them is increased, the number of evaluated placements in the PSA algorithm also increases.

In this case, the study begins with the following parameters: *m*=2, *s=2*, T_o=50 and *p*=0.9. I.e., the study starts computing a total of 152 placements every time the PSA algorithm is executed. The idea is to increase each parameter, compute the PSA algorithm 100 times and compute the accuracy explained in chapter 5.4 for all the 100 experiments. Then, an average accuracy is obtained. In order to compute this accuracy, an exhaustive evaluation per each network and for the three metrics used – Average Latency, Inter-Controller Average Latency and Imbalance - must be done.

Firstly, the number of evaluated placements has been computed when *m* is increased from 2 until 8, keeping other parameters with the same starting values, i.e., *s=2*, T_o=50 and *p*=0.9. Using (5.4) expression in chapter 5.3, the following values: [152, 228, 304, 380, 456, 532, 608] have been obtained, this array is named *numEvPl* and its values represent the maximum number of placements that will be evaluated in a PSA computation. The next step is to adapt other variables to these values. In order to do so, other variables must keep the initial value. I.e., just one parameter is modified in each case to reach the same number of evaluated placements. Parameter *s* has to increase in the same way that parameter *m* increases due to it has the same function in expression (5.4), i.e., *s* and *m* values are: [2,3,4,5,6,7,8]. In order to compute T_o and *p*, they have been isolated from (5.4) obtaining:

$$T_o(i) = 10\hat{}(-\frac{numEvPl(i)}{m \cdot s} \cdot log10(p)) = 10\hat{}(-\frac{numEvPl(i)}{2 \cdot 2} \cdot log10(p0.9)) \quad (6.1)$$

$$p(i) = 10\hat{}(-\frac{log10(T_o)}{\frac{numEvPl(i)}{m \cdot s}}) = 10\hat{}(-\frac{log10(50)}{\frac{numEvPl(i)}{2 \cdot 2}}) \quad (6.2)$$

After computing (6.1) and (6.2), it is concluded that values for T_o and p will be:

T_o	Evaluated placements
50	152
406	228
3003	304
22231	380
164571	456
1218278	532
9018588	608

Table 6.4 To values when m=2, s=2 and p=0.9

p	Evaluated placements
0.9	152
0.9336	228
0.9498	304
0.9596	380
0.9662	456
0.9710	532
0.9745	608

Table 6.5 p Values when m=2, s=2 and To= 50

The expected behavior is that the more evaluated placements are computed, the better accuracy will be obtained. The intention of this study is to see which parameter reduces more the accuracy between the exhaustive PF and the PSA PF when this parameter is increased.

Firstly, the accuracy for Barrow Network has been computed comparing these three metrics: Average Latency, Inter-controller Average Latency and Load Imbalance. The following represented figures, 6.6, 6.7 and 6.8 are split in four different graphs. Each graph shows the obtained accuracy when increasing a specific parameter; either *m*, *s*, T_o or *p*, keeping other parameters with their initial value, i.e., *m*=2, *s=2*, T_o=50 and *p*=0.9.

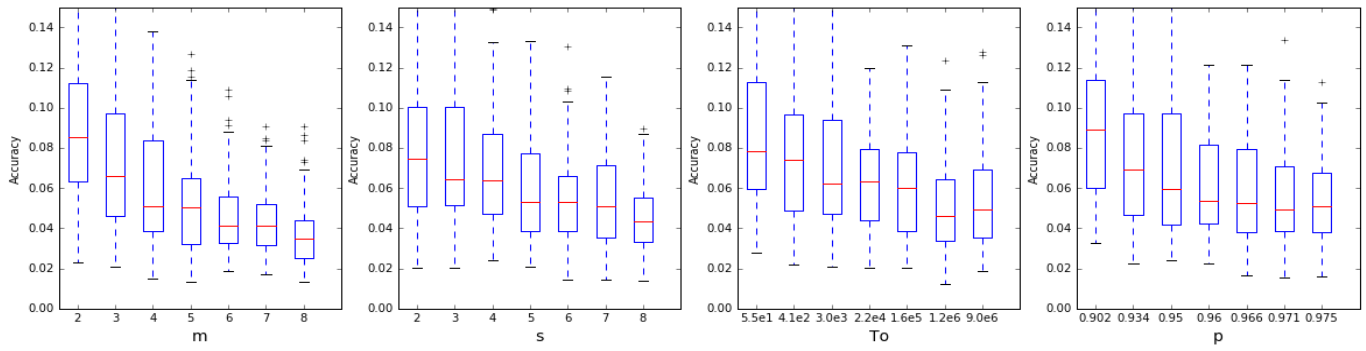

Figure 6.6 Accuracy for Barrow Network Av Lat vs IC Av Lat

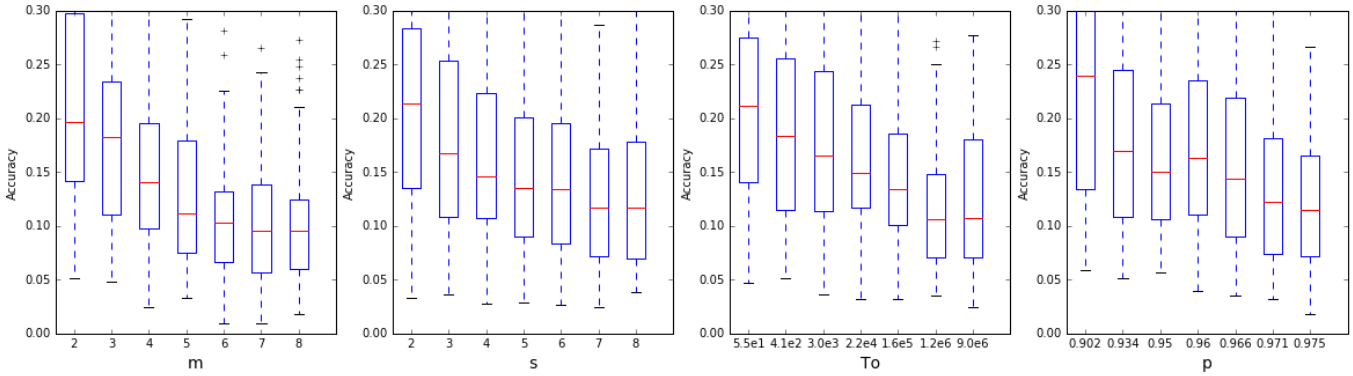

Figure 6.7 Accuracy for Barrow Network Av Lat vs Imb

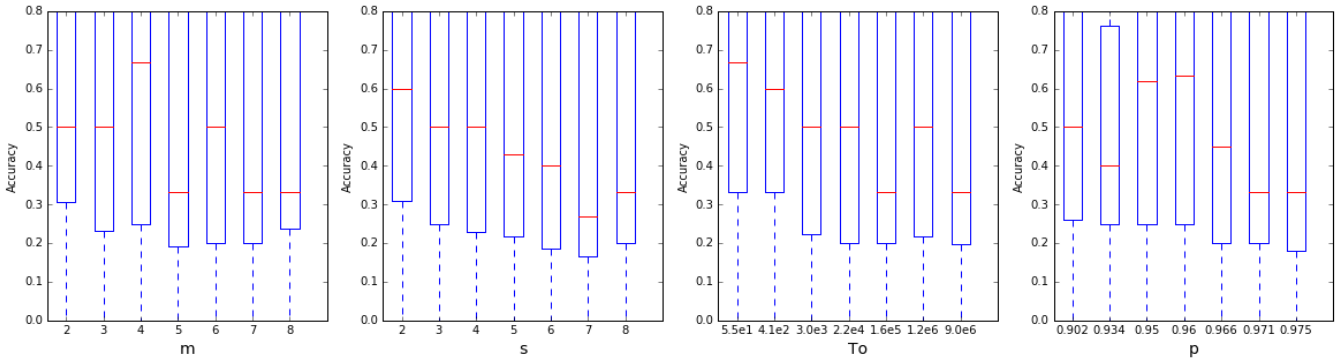

Figure 6.8 Accuracy for Barrow Network IC Av Lat vs Imb

As it can be seen in figures 6.6, 6.7 and 6.8, the obtained results are similar to the expected ones. When increasing the number of loops in the PSA algorithm, i.e., the number of maximum evaluated placements, accuracy takes a lower value, in other words, it improves. Each PSA has been repeated 100 times in order to obtain a more precise result.

Relationship between the PSA parameters and PSA accuracy

The next objective for this study is to see which metric has improved more when increasing the number of iterations in the PSA algorithm. In order to do so, the difference between the first value obtained and the last one has been computed per each parameter, e.g., the difference of accuracy when using m=2 and m= 8. The obtained result is named improvement. Each pair of compared metrics and each parameter has its own improvement. All improvements for each network and for each pair of compared metrics can be seen in figure 6.9.

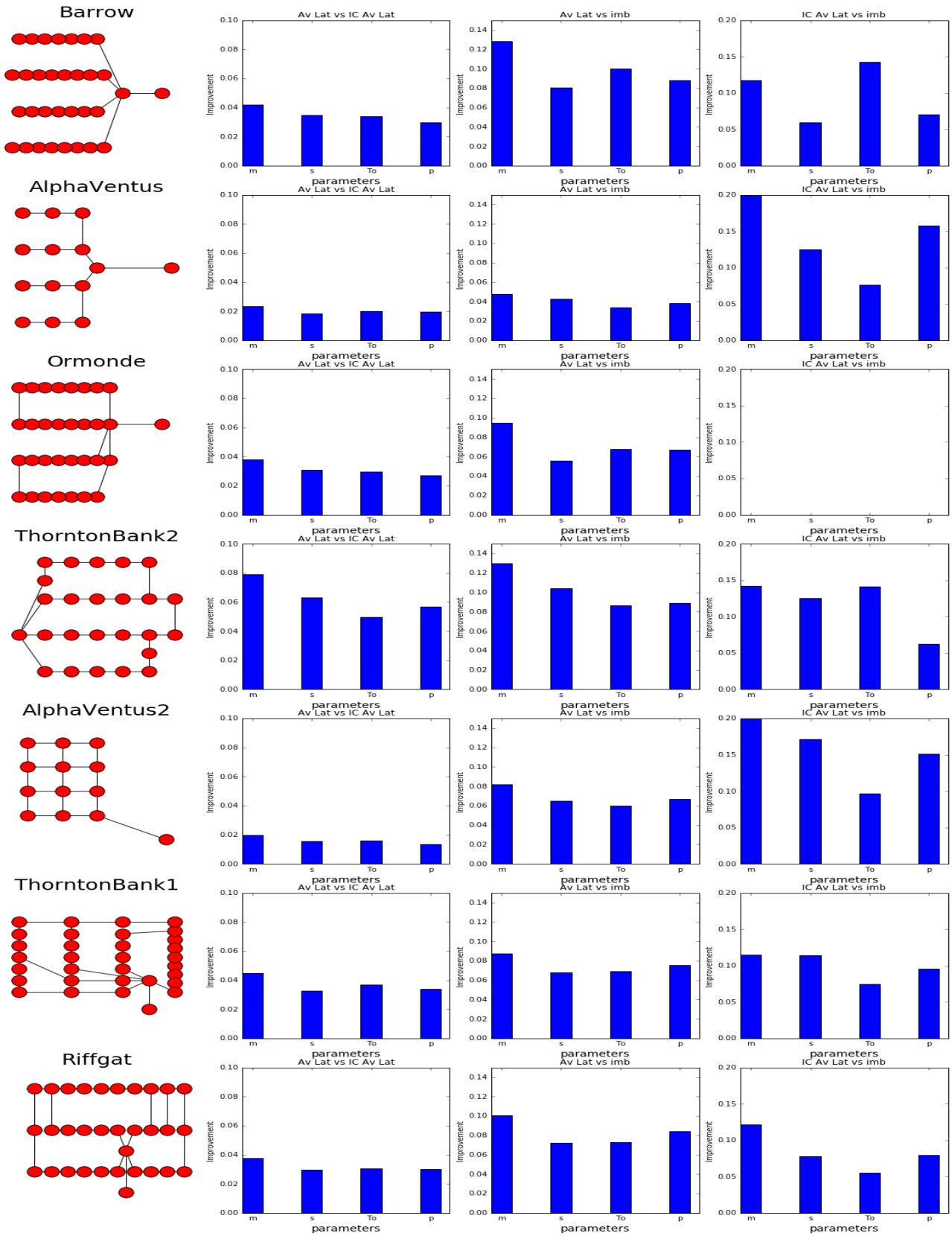

Figure 6.8. Improvement of each parameter in all cases

In figure 6.9, blue bars represent the improvement of each metric, i.e., the difference between the first and the last evaluation. Taking into account that the first case for each couple of metrics has similar results due to it uses the same parameters; the study wants to prove which one got a lower value when increasing the number of evaluated placements. I.e., the higher blue bar, the more it has improved.

In the following figure 6.10, the average improvement for each PSA parameter and each pair of metrics has been computed. It can be seen that *m* parameter has the greatest value in all three metric comparisons. Then, an important conclusion has been reached. The parameter that reduces more quickly the accuracy for all metric comparisons when increasing the number of iterations, and jointly the elapsed time, is *m* parameter. I.e., it reduces the accuracy between the exhaustive Pareto Frontier and the PSA Pareto Frontier faster than other parameters when considering the same number of PSA iterations, and therefore, the same elapsed time. Hence, the conclusion is that if accuracy is wanted to be reduced, the parameter that must be increased is *m*.

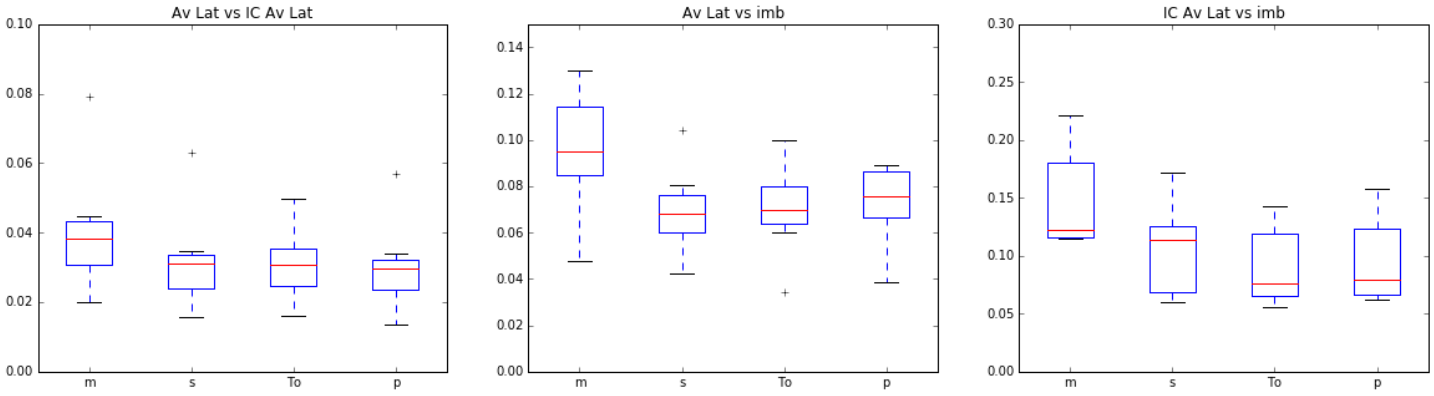

Figure 6.9 Average improvement between the first and last accuracy obtained for each parameter for k=4

Values shown above in figure 6.10 were computed with k = 4, it has also been computed with k=3 and the obtained values are the same ones.

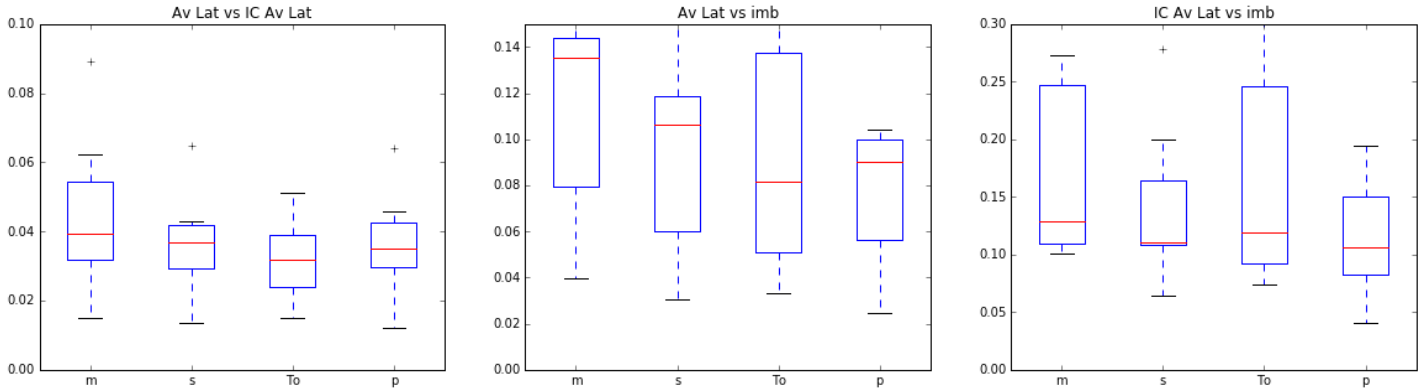

Figure 6.10 Average improvement for each parameter for k=3

6.3.2 PSA Random vs PSA ILP

In this study two different ways to compute the PSA algorithm have been studied, the first way to do it is by computing PSA algorithm as it is described in [7], i.e., starting from *s* random placements. The second way to compute it is starting from ILP best placement solutions. These two algorithms are named as *case 1* and *case 2*, respectively. In order to compare these two procedures, the accuracy between the original Pareto Frontier, i.e., the one obtained from an exhaustive evaluation, and the Pareto Frontier obtained from either *case 1* or *case 2*.

The chosen parameters for this evaluation are the following ones: *s* has to be equal to 2 because *case 2* can just start from two different placements, so it has to be the same for *case 1*. In order to see how the other parameters affect results, they have also been modified and the experiment has been repeated with the new ones. The first set of values used is m=2, T_o =50, p=0.9 and k =3.

In order to see if results had coherence, the whole calculous has been repeated three times. The obtained results in these three times are similar, but not equals. In this document the three of them are shown. In each case, the PSA evaluation has been computed 50 times in order to obtain a more exact accuracy value, computing the average of accuracy from the 50 obtained values.

In the following figure 6.12 an example of how the fifty computed Pareto-Frontiers are distributed in Barrow topology when using Average Latency and IC Average Latency metrics is shown. As it can be seen here and in more figures shown below, the accuracy in *case 2* is better than in *case 1* due to its Pareto Frontiers are closer to the one computed by an exhaustive evaluation, shown in red. Green lines represent the obtained Pareto Frontiers using the PSA algorithm.

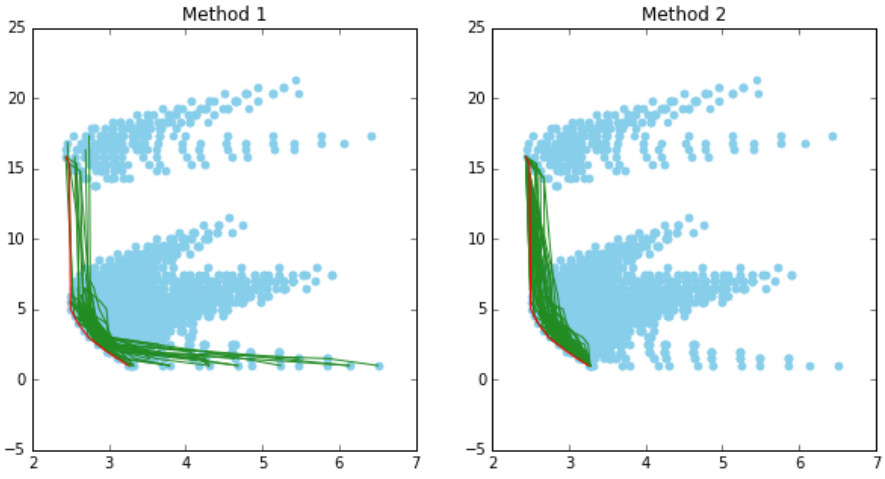

Figure 6.11 Pareto Frontiers when using method 1 and method 2

In figure 6.13 the average accuracy in *case 1* and in *case 2* is shown. If the accuracy in *case 2* is better, i.e., it has a lower value than accuracy in *case 1*, a green line that represents an improvement has been printed. On the other hand, if it is worse, a red line

has been printed. Moreover, a yellow line has been printed in case that the improvement or worsening of *case 2* in relation to case 1 is equal or below 3%.

In the next figures the seven topology distributions, their exhaustive evaluation when comparing two metrics and the accuracy obtained for *case 1* and *case 2* have been printed. PSA parameters used for this experiment have been m=2, T_o=50, p=0.9 and k=3. Firstly, the common PSA algorithm - *case* 1 - has been computed fifty times and then, using Gurobi, two optimal values have been obtained and used as starting point for the PSA, computing *case 2.* In figures 6.13, 6.14 and 6.15 is shown the same experiment repeated three times with the same parameters.

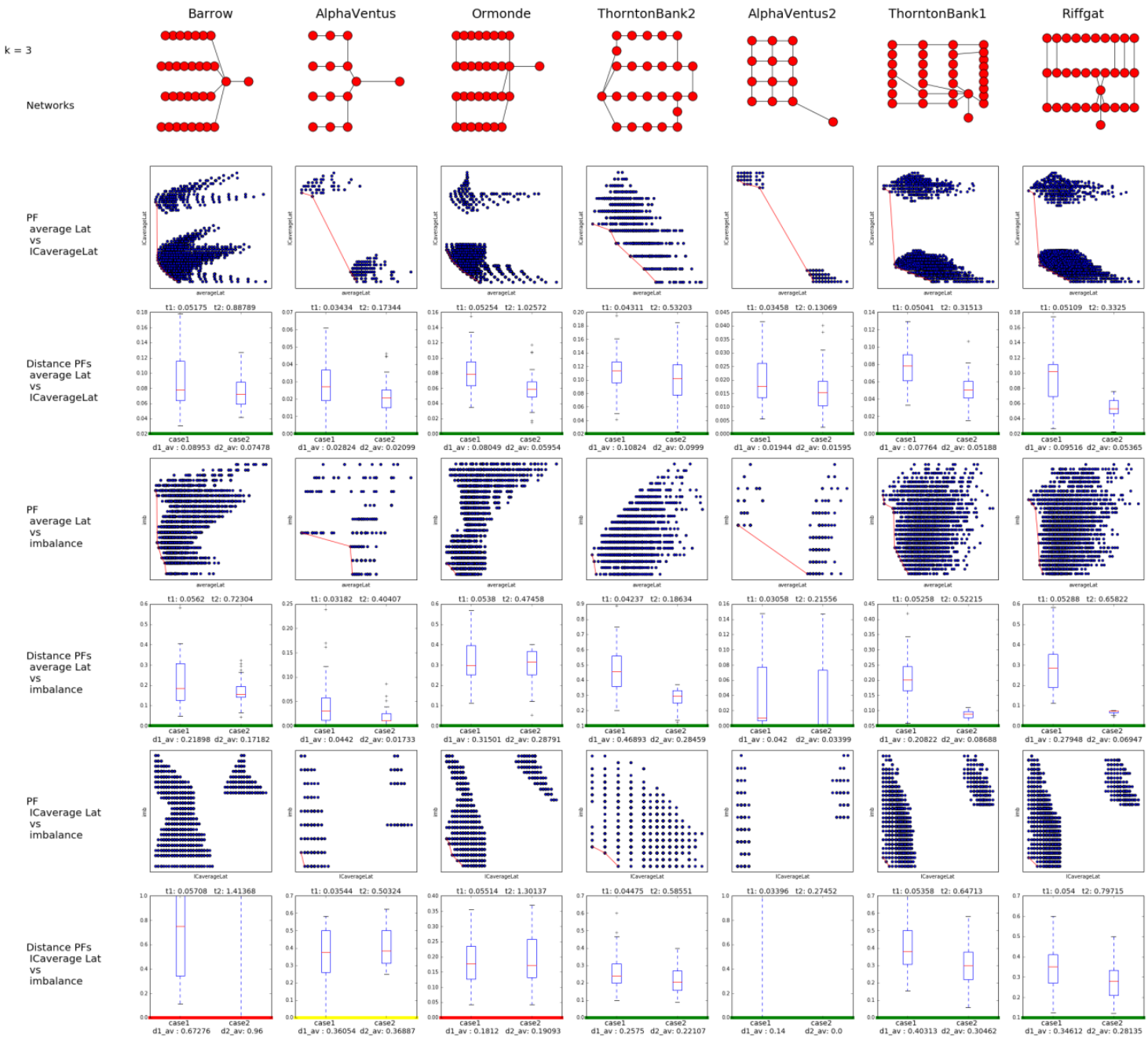

Figure 6.12 Accuracy in Case 1 vs Accuracy in Case 2. First computation.

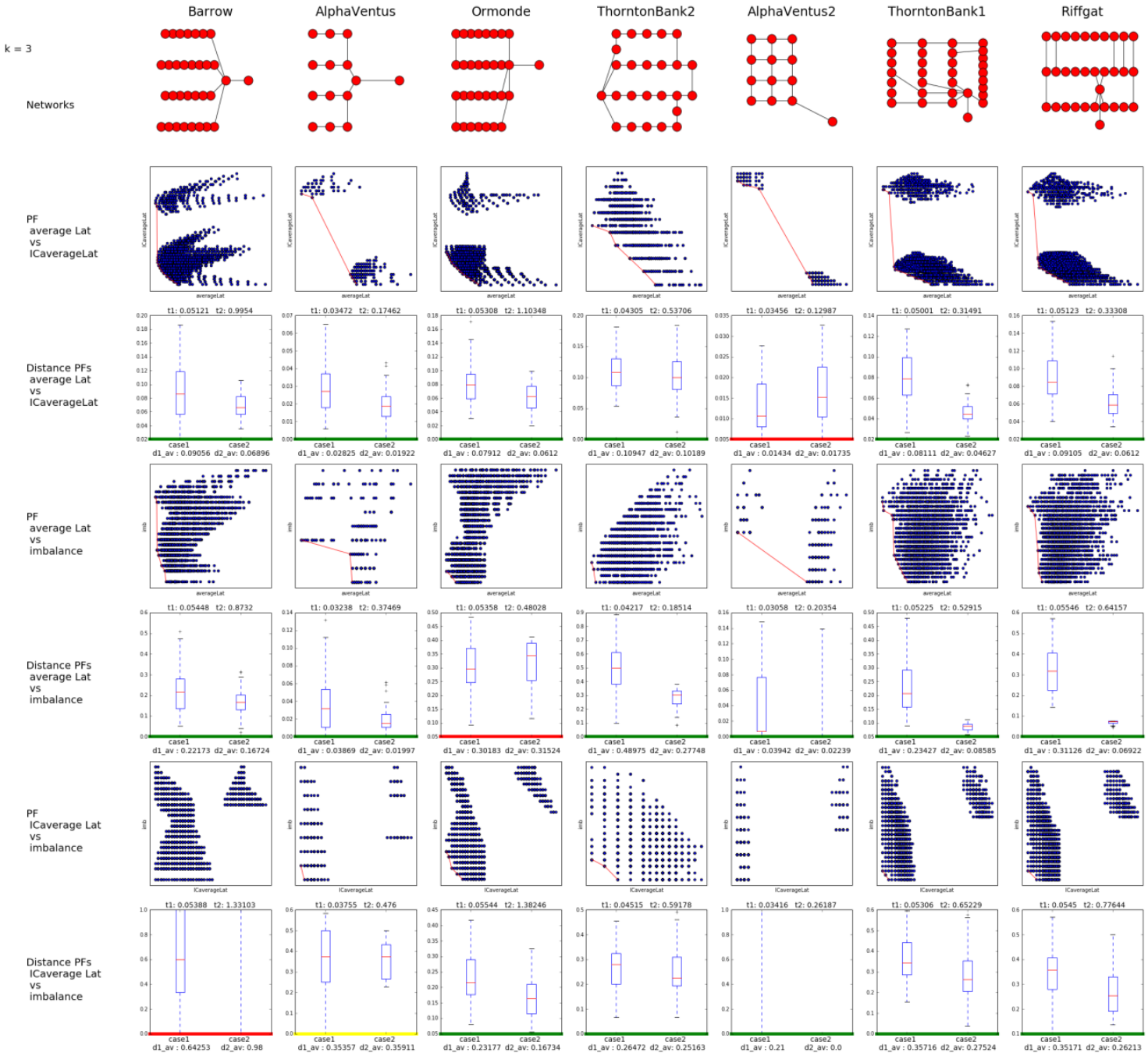

Figure 6.13 Accuracy in Case 1 vs Accuracy in Case 2. Second computation.

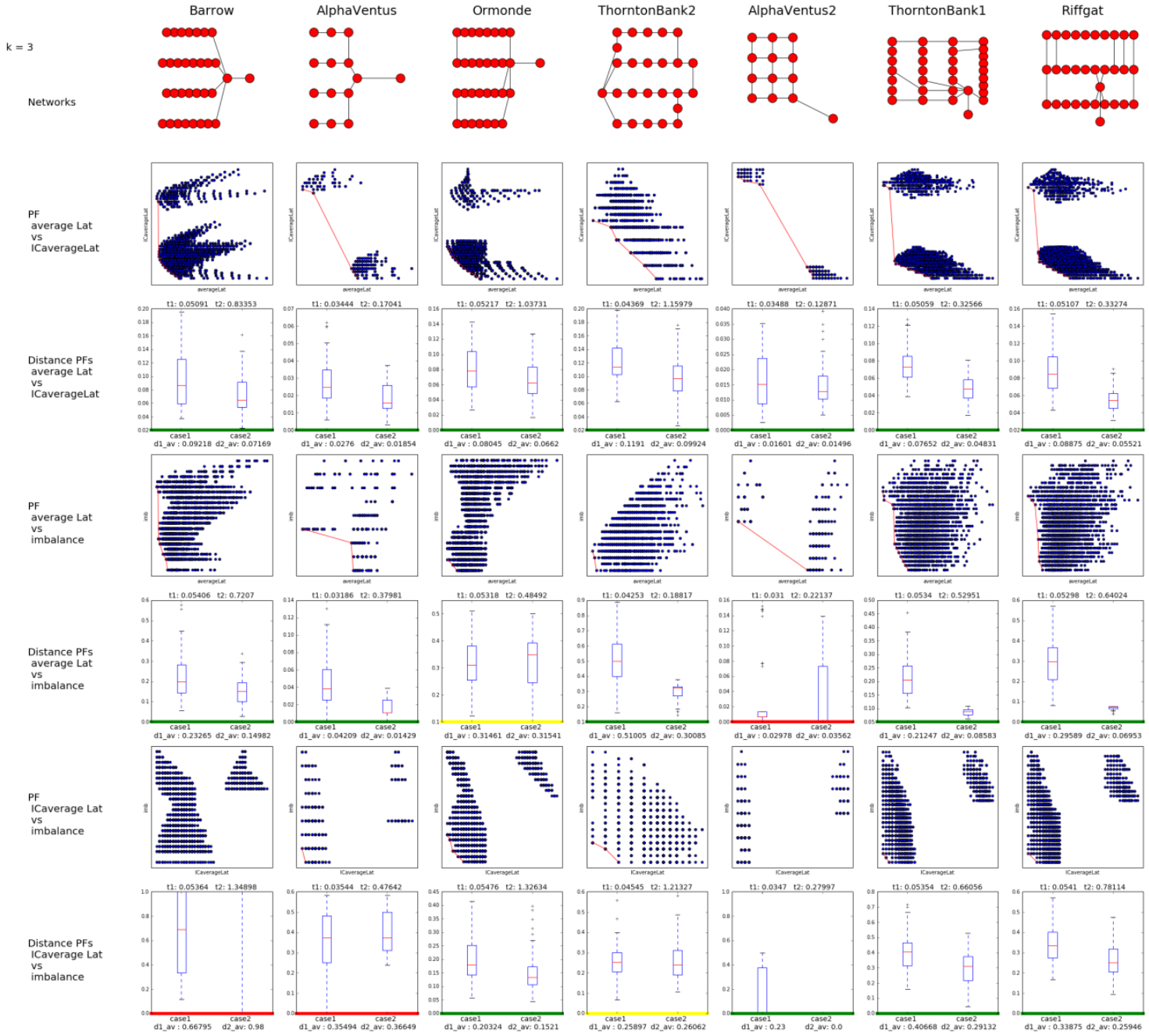

Figure 6.14 Accuracy in Case 1 vs Accuracy in Case 2. Third computation.

As it is shown in figures 6.13, 6.14 and 6.15, in general accuracy improves for different metrics and for all industrial networks when using *case 1* and *case 2* with the same PSA parameters. However, for Barrow and AlphaVentus accuracy does not improve when using metrics ICaverage Latency and Imbalance. Furthermore, elapsed time in case 2 is significantly higher than *case 1*, due to the computation time of the ILP.

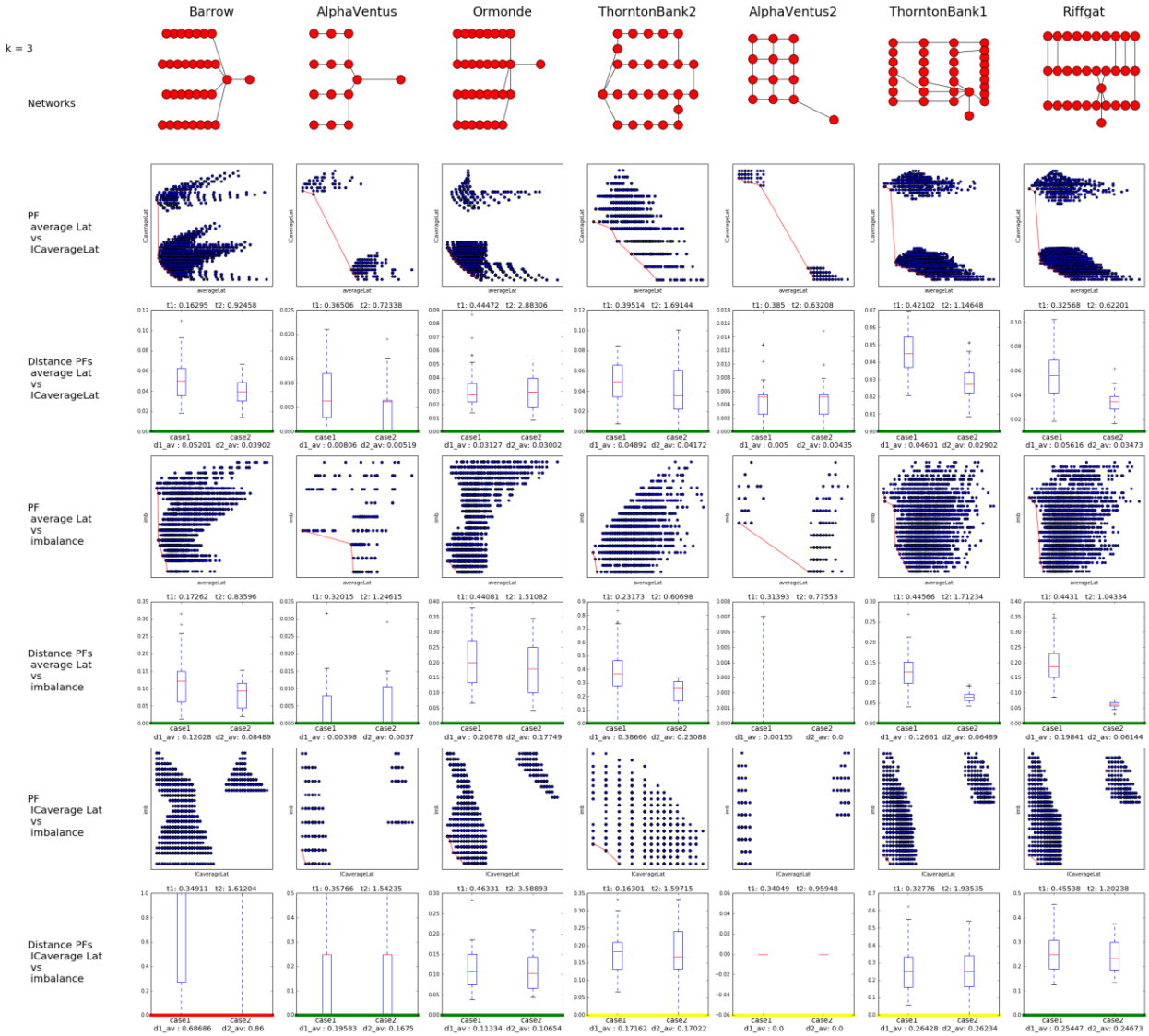

Figure 6.15 Accuracy in Case 1 vs Accuracy in Case 2. With m=5.

This figure has also been computed for m=5, T_o=50, p=50, k=3 and 50 repetitions of the PSA evaluation. When increasing m from 2 to 5, the number of times that the loop in the PSA is repeated is also increased. Therefore, more placements can be evaluated and accuracy is supposed to be smaller in both cases.

As it is seen in figure 6.16, *case 2* still improves the accuracy when using m equals to 5. It has the same behaviour.

Nonetheless, if elapsed time is taken into account, it can be seen that *case 2* takes much more time than *case 1*. When using the same parameters for the PSA, in some situations *case 2* spends around 20 times more time than *case 1*. Therefore, in the next section it is going to be studied which case is faster when they both have a similar accuracy value.

Elapsed time: ILP starting points for PSA versus Random Placements for PSA

In order to see which method is the fastest one to reach the same accuracy value, *case 2* is firstly computed, where PSA algorithm starts from an optimal placement obtained by an ILP computation. Then, the experiment has been repeated using *case 1* increasing parameter *m*, which controls the number of PSA algorithm repetitions, until a lower accuracy than the one obtained in case 2 is reached.

For the next study PSA variable *m* starts from 2 and is increased by 1 at a time until 10. Parameter *k* is also modified, using k = 3, 4 and 6. Other variables have remained constant like T_o = 50, p=0.9 and s=2. Average Latency, Inter-Controller Latency and Load Imbalance have been compared by separate.

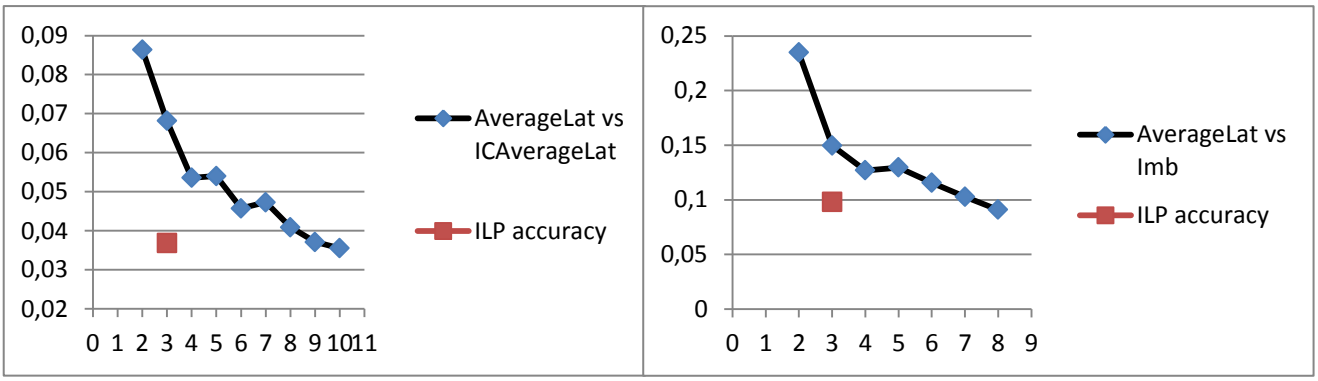

Figure 6.16 (Left) Accuracy for Average Latency vs IC Average Latency. (Right) Accuracy for Average Latency vs Imbalance

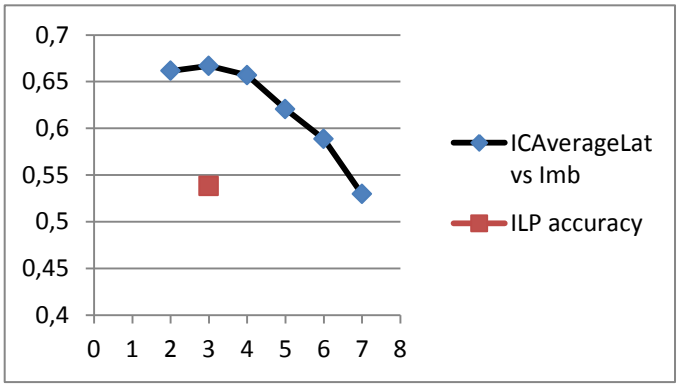

Figure 6.17Accuracy for IC Average Latency vs Imbalance

Figures 6.17 and 6.18 plotted above show the accuracy obtained when using ILP method combined with PSA, i.e., *case 2*, with a red square, using m = 3. Moreover, a black line with blue dots is used to represent the accuracy of *case 1* when modifying *m*. This graphics have been obtained from Barrow Network for 3 controllers. Just this network is plotted due to the behaviour with other networks is similar.

As it is seen in figures 6.17 and 6.18, *case 2* has a lower accuracy than *case 1* when using the same value of *m*. But, when *m* is increased, *case 1* accuracy becomes equal or smaller than the one obtained in case 2.

In table 6.5 and 6.6, accuracy and time average values from the fifty repetitions are shown for *case 1* and *case 2*. Also *m* value and the number of loops that are computed in the PSA algorithm are represented. As it can be seen in *m=3* row, *case 2* has a better accuracy when the same amount of placements is evaluated. However, when *m* is equal to 10, *case 1* accuracy is lower than *case 2* accuracy and elapsed time is 2.8 times faster for *case 1* (0,37829 seconds) than for *case 2* (1,04782 seconds), even if *case 2* is computing more placements than *case 1*. The problem with *case 2* is that it elapses too much time computing the ILP algorithm, specifically 0.0625 seconds for Average Latency and 0.8906 seconds for IC average Latency. That's why *case 1* is faster than *case 2* for reaching the same accuracy.

With figures 6.17 and 6.18 it can also be proved that; when the number of loops in the PSA algorithm is increased, the accuracy between the estimated Pareto Frontier and the real one is reduced. This is logical due to the more loops, the more evaluated placements, as it is said in expression 5.4 in chapter 5.3.

		case 2		case 1	
m	Evaluated placements	accuracy	time (sec)	accuracy	time (sec)
2	152			0,086400	0,05396
3	228	0,036850	1,04782	0,068147	0,09315
4	304			0,053557	0,13247
5	380			0,054023	0,17132
6	456			0,045738	0,21368
7	532			0,047301	0,25533
8	608			0,040882	0,29886
9	684			0,037100	0,33681
10	760			0,035513	0,37829

Table 6.6 Accuracy and time when changing m. Av Lat vs IC av Lat.

In the next table 6.6, the behaviour when two other metrics are studied is shown; Inter Controller Average Latency and Load Imbalance metric. Obtained results are similar, i.e., accuracy is reduced when *m* is increased. However, here time elapsed in *case 1* (0,26804 seconds) is 5.87 times faster than in *case 2* (1,57372 seconds). Again, the problem is that the elapsed time for the ILP algorithm is too heavy; 0.8906 seconds for IC average Latency and 0.5821 seconds for Imbalance.

		case 2		case 1	
m	Evaluated placements	accuracy	time (sec)	accuracy	time (sec)
2	152	0,53831	1,57372	0,66156	0,05872
3	228			0,66692	0,10031
4	304			0,65698	0,14125
5	380			0,62049	0,18374
6	456			0,58844	0,22557
7	532			0,52954	0,26804

Table 6.7 Accuracy and time when changing m. IC av Lat vs Imb

Then, the obtained conclusion is that our suggested method is not better than the common PSA algorithm. I.e., PSA algorithm starting from a random point is faster than PSA algorithm starting from ILP optimal placements. Furthermore, it is also demonstrated that the higher number of evaluated placements that are computed in PSA algorithm, the better accuracy is obtained, but also, the more time is required.

As it is talked about in Chapter 6.2 ILP implementation, this method returns one of the possible optimal solutions for one metric and there can be a lot, i.e., it does not always return the optimal value also contained in the Pareto Frontier. Our method could be much better if the ILP method returned us the optimal placement for a metric that was also located on the Pareto Frontier. Accuracy in *case 2* would be much lower and *case 1* would have to spend more time to reach it. However, this is not possible due to there is not a way that forces the ILP problem to return this optimal placement contained in the Pareto Frontier.

Chapter 7 Conclusions and Outlook

This project wanted to study the behaviour of some Industrial Networks when they were used as SDN Networks. It was wanted to see how they reacted to some different metrics when using different number of controllers. Moreover, the thesis wanted to study these networks for the CPP problem, trying to solve where and how many controllers had to be deployed. In order to solve it, a code capable to compute six different metrics and compute and show de 2D Pareto Frontier between them was developed. There were 15 possible combinations of those metrics per each topology. But then, this study realised that some metrics were correlated and, therefore, they could be treated as equals. That is why this project continued studying the Industrial Networks just comparing three different Pareto Frontiers instead of fifteen.

This thesis also focused its work on a metaheuristic algorithm called PSA due to exhaustive evaluations are not realistic to solve the CPP problem because their evaluation time is too large and this networks need a fast reaction time. This algorithm was studied varying some of its parameters and using some Industrial Networks. The objective was to know which PSA parameter was better to increase in order to obtain a better accuracy. By repeating more times the PSA loop, accuracy is improved, i.e., it gets a lower value. The number of loops can be increased by increasing *m*, *s*, T_o or *p* PSA parameters. The conclusion obtained here is that PSA algorithm improves faster when increasing *m* parameter.

Finally, this project wanted to improve the PSA algorithm with a new one, adding ILP problem to it. Instead of starting from a random value, this new algorithm was started from an optimal allocation for a metric, obtained by solving an ILP problem for each evaluated metric with Gurobi. Unfortunately, results were not good. This method computed a better accuracy when using the same number of iterations in the PSA algorithm, but, however, it spent more time to reach this accuracy than the common PSA algorithm. Therefore, even if the PSA algorithm that started from random placement needed to evaluate more placements, it was faster due to ILP computations were too slow.

This project could be improved by taking into account more industrial networks and also by comparing the obtained results with Zoo topologies. Also larger *k* number could be tested, maybe with higher number of controllers results would change. Moreover, more metrics could be also considered, including the ones that take into account failures. Another situation that could be studied is the load imbalance metric. In this thesis a homogeneous load is considered, i.e., each node has the same load but, in real scenarios some switches may have a higher demand than others.

As it is said in chapter 6.3, it would be useful to find a similar method to ILP that provided the two edges in the Pareto Frontier. This could decrease the accuracy for the proposed method in 6.3.2. It could be also tried, as it is said in [7], to change the way of how neighbours are assigned to the placements in the PSA algorithm. Finally, it could be also tried to modify α PSA parameter which it is used to assign and modify weights.

Chapter 8

Formatting

8.1. List of figures

Figure 2.1 Traditional Switch vs SDN Switch [12] 9

Figure 2.2 Example of SDN architecture [13]. 10

Figure 2.3 Barrow and AlphaVentus are considered as Tree Topologies [10] 11

Figure 2.4 Ormonde is a Ring Topology [10] 11

Figure 2.5 ThornonBank2 is considered as a Tree + Ring topology [10] 11

Figure 2.6 AlphaVentus2 is the classical grid topology [10] 11

Figure 2.7 ThorntonBank1 and Riffgat are Meshed topologies [10] 12

Figure 2.8 Cost266 topology 12

Figure 3.1 Latencies reduction when *k* is increased. Fig 4 in [1]. 15

Figure 3.2 Latencies versus number of controllers. Figure 6 in [1]. 16

Figure 5.1 Relative budget for different Size of Search Space in order to achieve an accuracy level. Figure 5 in [7]. 23

Figure 5.2 Accepting worse values to reach the global minimum 26

Figure 5.3 Representation of the PSA algorithm 27

Figure 6.1 WC Lat vs Av Lat, WC Lat vs IC WC Lat and WC Lat vs IC Av Lat for k =3 32

Figure 6.2 WC Lat vs Av Lat, WC Lat vs IC WC Lat and WC Lat vs IC Av Lat for k =5 32

Figure 6.3 WC Lat vs Av Lat, WC Lat vs IC WC Lat and WC Lat vs IC Av Lat for k =7 33

Figure 6.4 Correlated metrics 34

Figure 6.5 ILP method applied for Average Latency and IC Average Latency 35

Figure 6.6 Accuracy for Barrow Network Av Lat vs IC Av Lat 37

Figure 6.8 Accuracy for Barrow Network Av Lat vs Imb 38

Figure 6.9. Improvement of each parameter in all cases 39

Figure 6.10 Average improvement between the first and last accuracy obtained for each parameter for k=4 40

Figure 6.11 Average improvement for each parameter for k=3 40

Figure 6.12 Pareto Frontiers when using method 1 and method 2 41

Figure 6.13 Accuracy in Case 1 vs Accuracy in Case 2. First computation. 42

Figure 6.14 Accuracy in Case 1 vs Accuracy in Case 2. Second computation. 43

Figure 6.15 Accuracy in Case 1 vs Accuracy in Case 2. Third computation. 44

Figure 6.16 Accuracy in Case 1 vs Accuracy in Case 2. With m=5. 45

Figure 6.17 (Left) Accuracy for Average Latency vs IC Average Latency. (Right) Accuracy for Average Latency vs Imbalance 46

Figure 6.18Accuracy for IC Average Latency vs Imbalance 46

Figure 1.1 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=3 (1) 55

Figure 1.2 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=3 (2) 56

Figure 1.3 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=4 (1) 56

Figure 1.4 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=4 (2) 56

8.2. List of Tables

Table 2.1 Example of requirements in an industrial network 10

Table 5.1 PSA Algorithm 24

Table 6.1 Number of evaluated placements per network and per *k* 29

Table 6.2 Total amount of evaluated placements per *k* 29

Table 6.3 Elapsed time for different number of controllers 31

Table 6.5 To values when m=2, s=2 and p=0.9 37

Table 6.6 p Values when m=2, s=2 and To= 50 37

Table 6.7 Accuracy and time when changing m. Av Lat vs IC av Lat. 47

Table 6.8 Accuracy and time when changing m. IC av Lat vs Imb 48

8.3. Notation and Abreviations

This chapter contains tables where all abbreviations and other notations like mathematical placeholders used in the thesis are listed.

Av lat	Average Latency
Avail	Availability
CPP	Controller Placement Problem
E	Set of links
G	Network graph
Ic av lat	Inter-controller average latency
Ic wc lat	Inter-controller worst-case latency
ILP	Integer Linear Programming
Imb	Load imbalance
k	Number of controllers
MILP	Mixed-Integer Linear Programming
n	Number of nodes
PF	Pareto Frontier
POCO	Pareto-Optimal Controller Placement
PSA	Pareto Simulated Annealing
SDN	Software Defined Networking
SoA	State of the Art
V	Set of nodes
Wc lat	Worst-case latency

8.4. References

- [1] [Hel12] Heller, B., Sherwood, R., & McKeown, N. (2012). The controller placement problem. ACM SIGCOMM Computer Communication Review, 42(4), 473.
- [2] [Hock13] Hock, D. (2013). Pareto-optimal resilient controller placement in SDN-based core networks, 1–9.
- [3] [Hu13] Hu, Y., Wendong, W., &Gong, X. (2013). Reliability-aware controller placement for Software-Defined Networks. Integrated Network, 672-675.
- [4] [HuRel14] Yannan, H.U., Wendong, W., Xiangyang, G., Xirong, Q. U. E., & Shiduan, C. (2014). On Reliability-optimized Controller Placement for Software-Defined Networks, (February), 38-54.
- [5] [Lange] Lange, S., Gebert, S., Spoerhase, J., Rygielski, P., Zinner, T., & Kouvnev, S. (n.d.). Specialized Heuristics for the Controller Placement Problem in Large Scale SDN Networks.
- [6] Five Nines of Southbound Reliability in Software-Defined Networks.
- [7] [Lan15] Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks
- [8] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tram-Gia, „POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks, “ in Proc. INFOCOM, Toronto, ON, Canada, 2014, pp 115-116.
- [9] [Zhang11] Zhang, Y., Beheshti, N., & Tatipamula, M. (2011). On resilience of split-architecture networks. GLOBECOM – IEEE Global Telecommunications Conference.
- [10] VirtuWind
- [11] ZooTopology
- [12] http://bradhedlund.com/2011/04/21/data-center-scale-openflow-sdn/
- [13] http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html
- [14] IEEE Standard 1646-2004, “Communication Delivery Time Performance Requirements for Electric Power Substation Automotion”.
- [15] M. Goraj, Y. Epassa, R. Midence und D. Meadows, “ Designing and deploying Ethernet networks for offshore wind power applications – a case study,” 10th IET International Conference on Developments in Power System Protection, pp. 1- 5, 2010
- [16] Hock, D., Gebert, S., Hartmann, M., Zinner, T., & Tran-Gia, P. (2014). POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks. IEEE/IFIP NOMS 2014 – IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World.

Appendix A

In this appendix some exhaustive evaluations are plotted. In all cases the topology is shown within its 15 possible combinations of metric comparisons. The metrics used in these comparisons are listed and explained in chapter 3.3 Performance metrics.

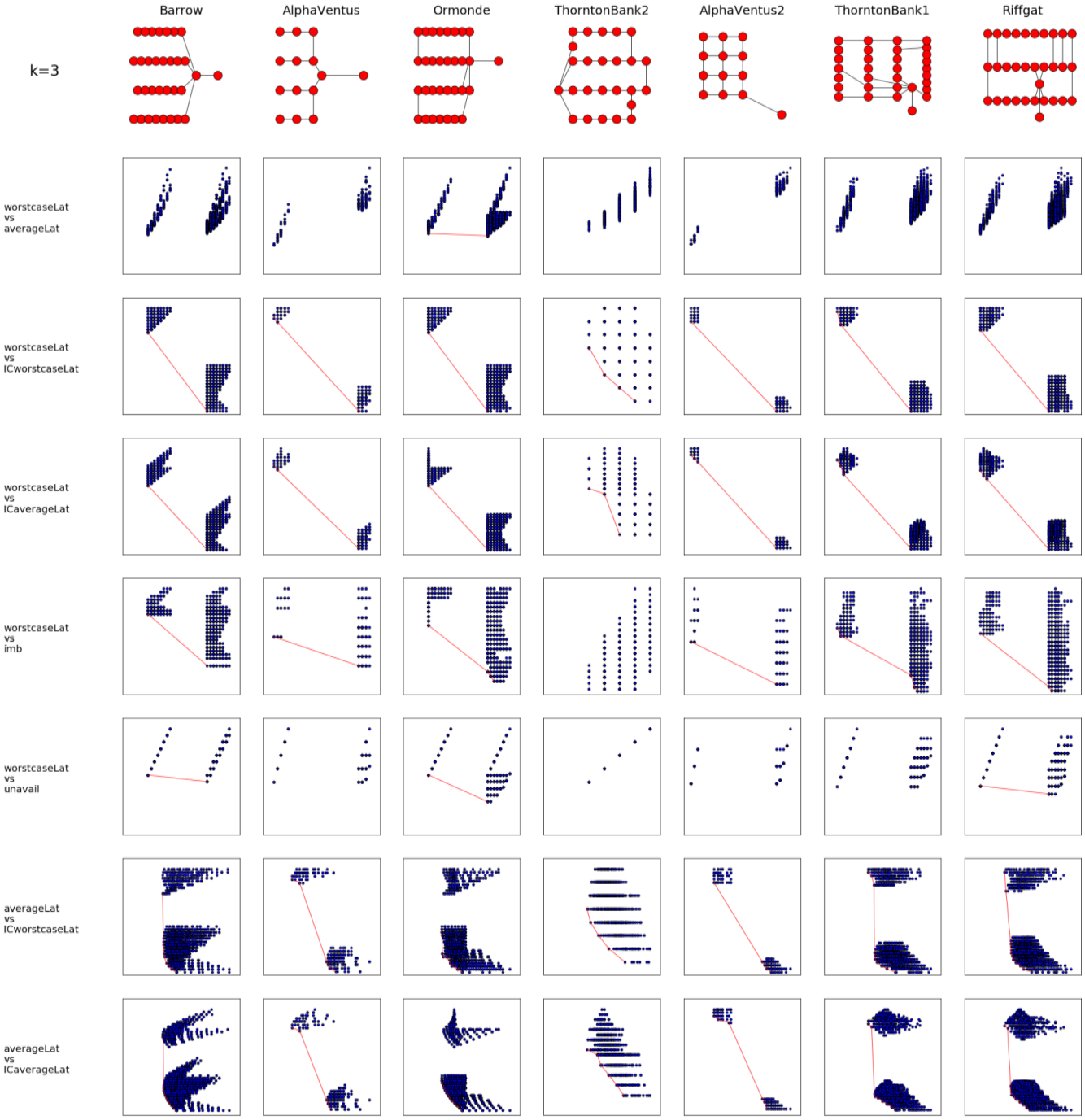

Figure 8.1 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=3 (1)

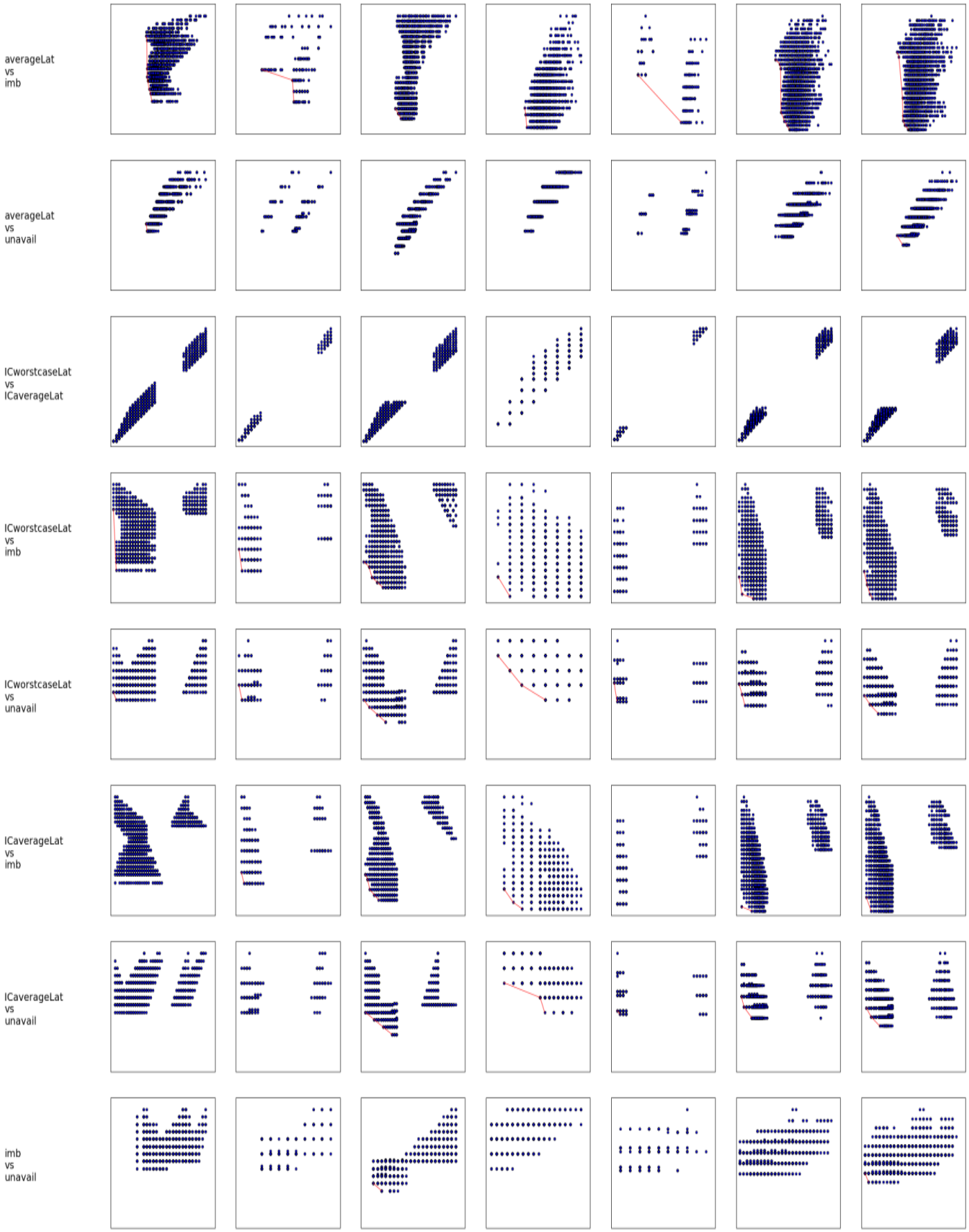

Figure 1.2 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=3 (2)

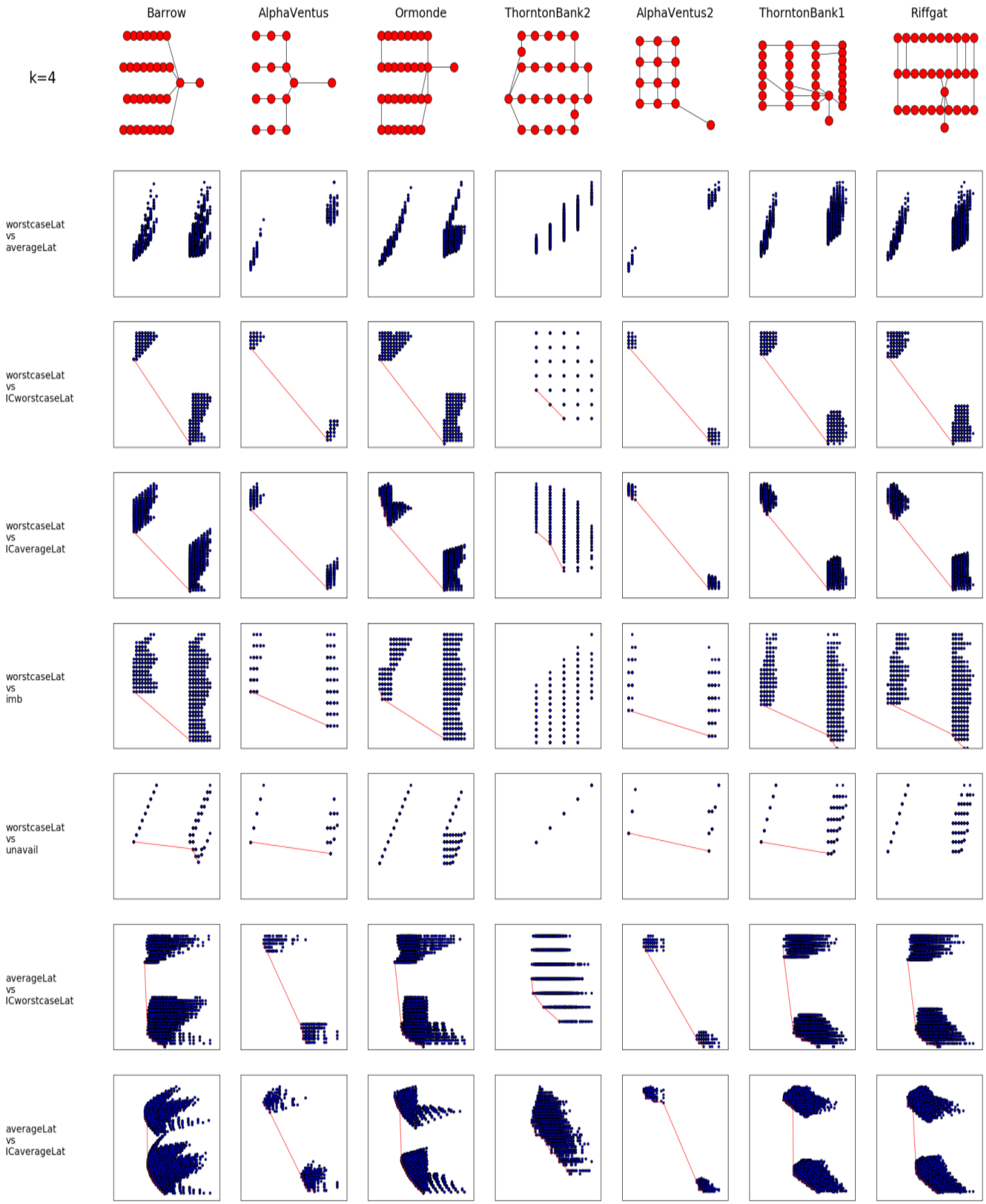

Figure 1.3 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=4 (1)

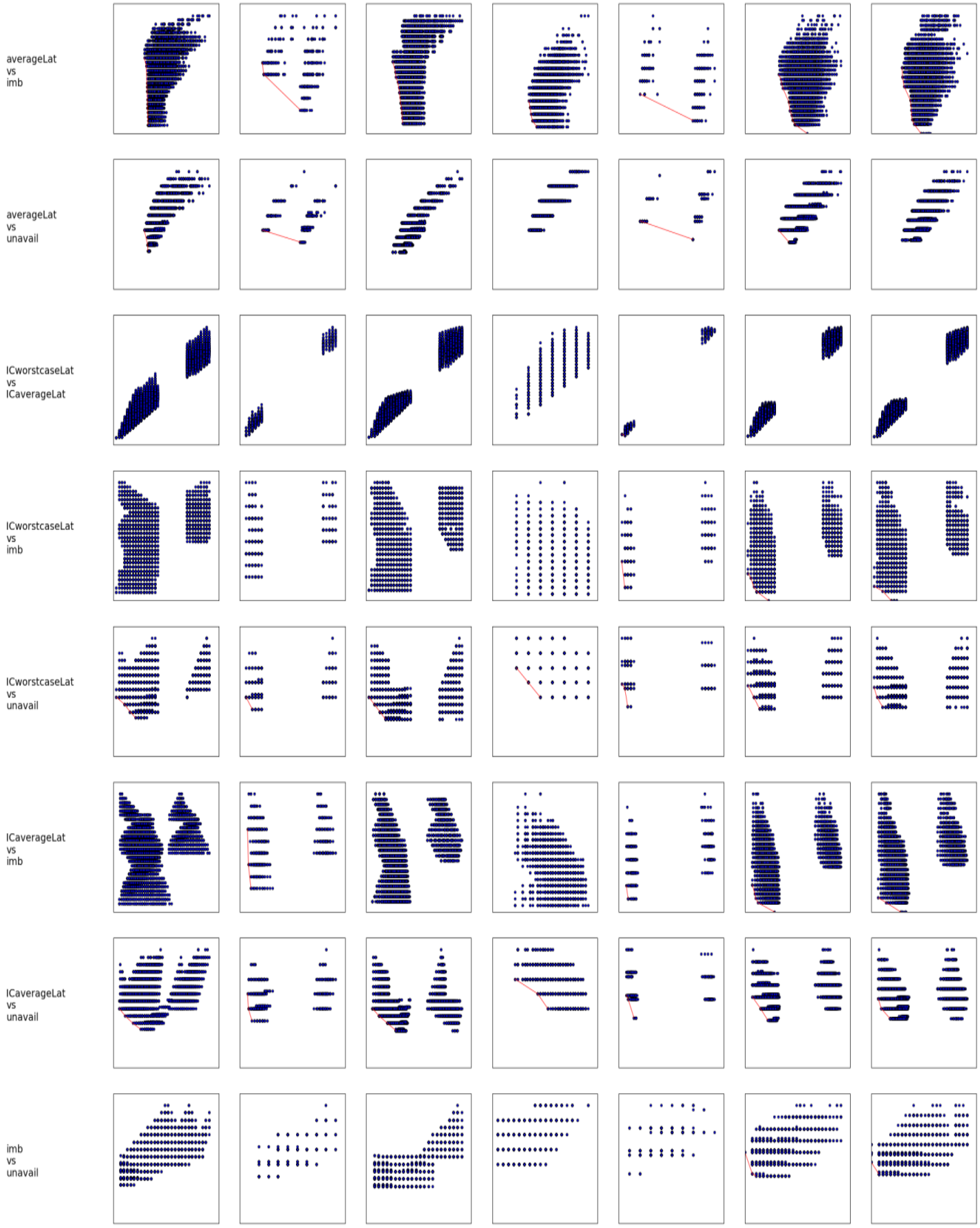

Figure 1.4 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=4 (2)

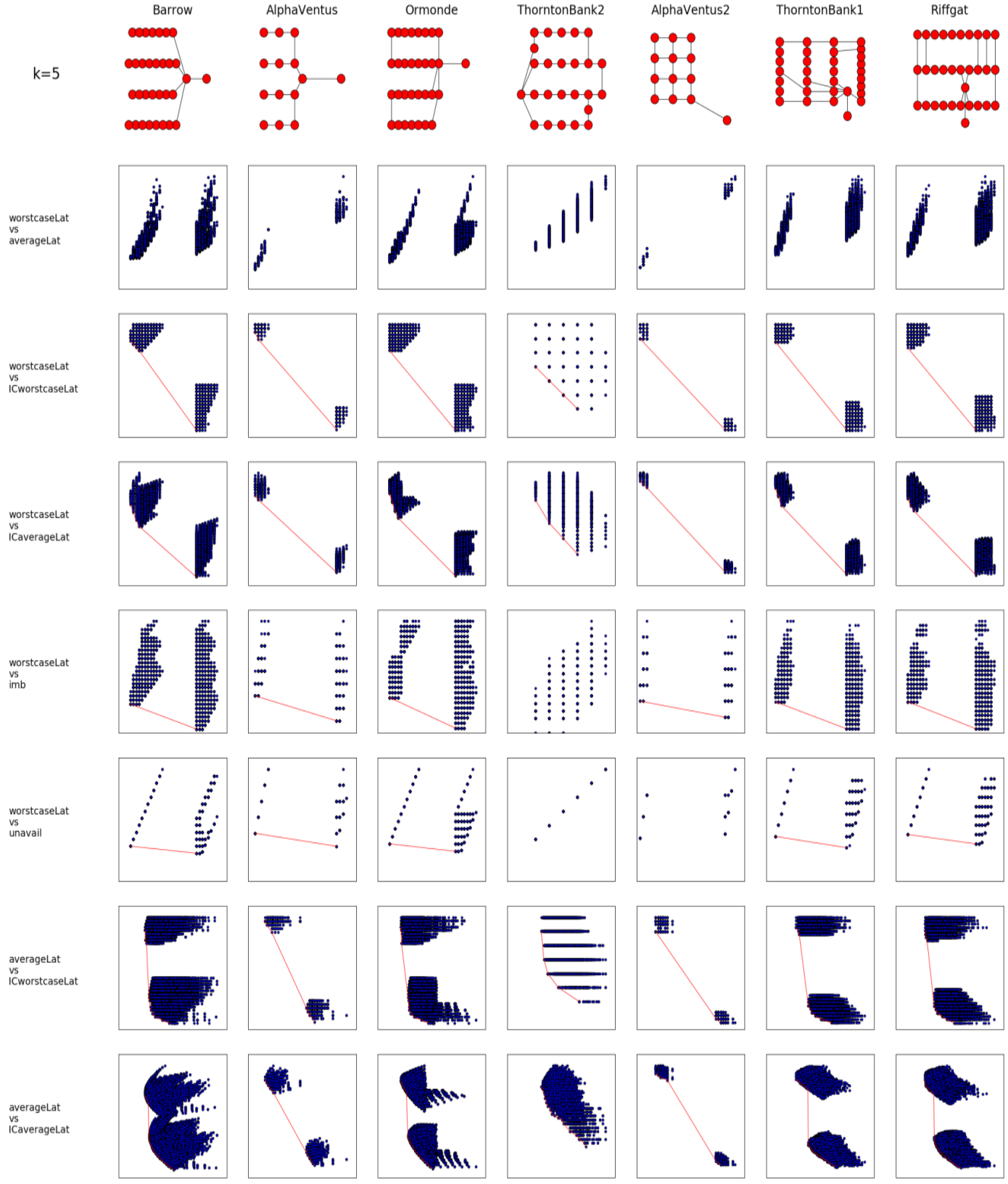

Figure 3 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=5 (1)

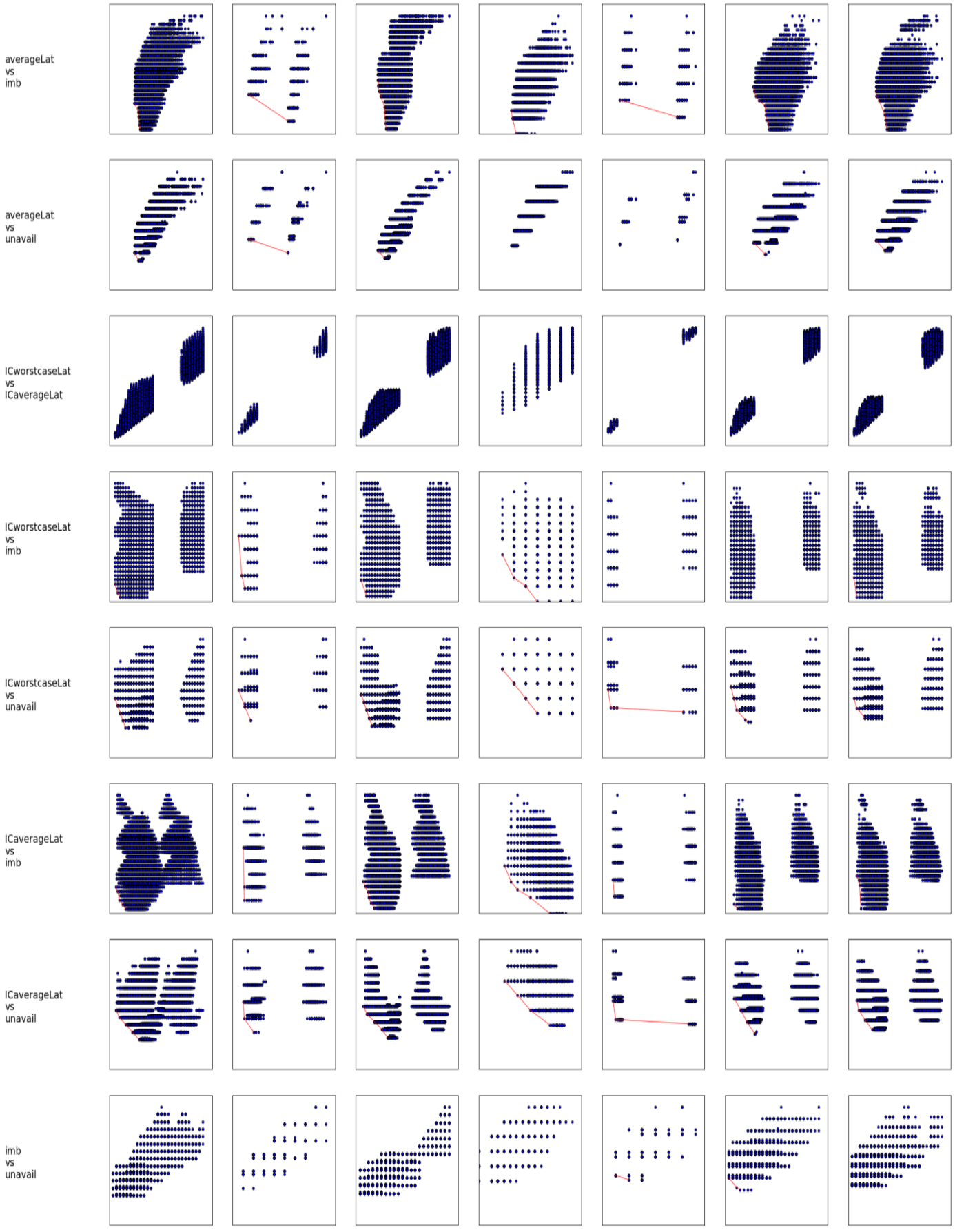

Figure 3 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=5 (2)

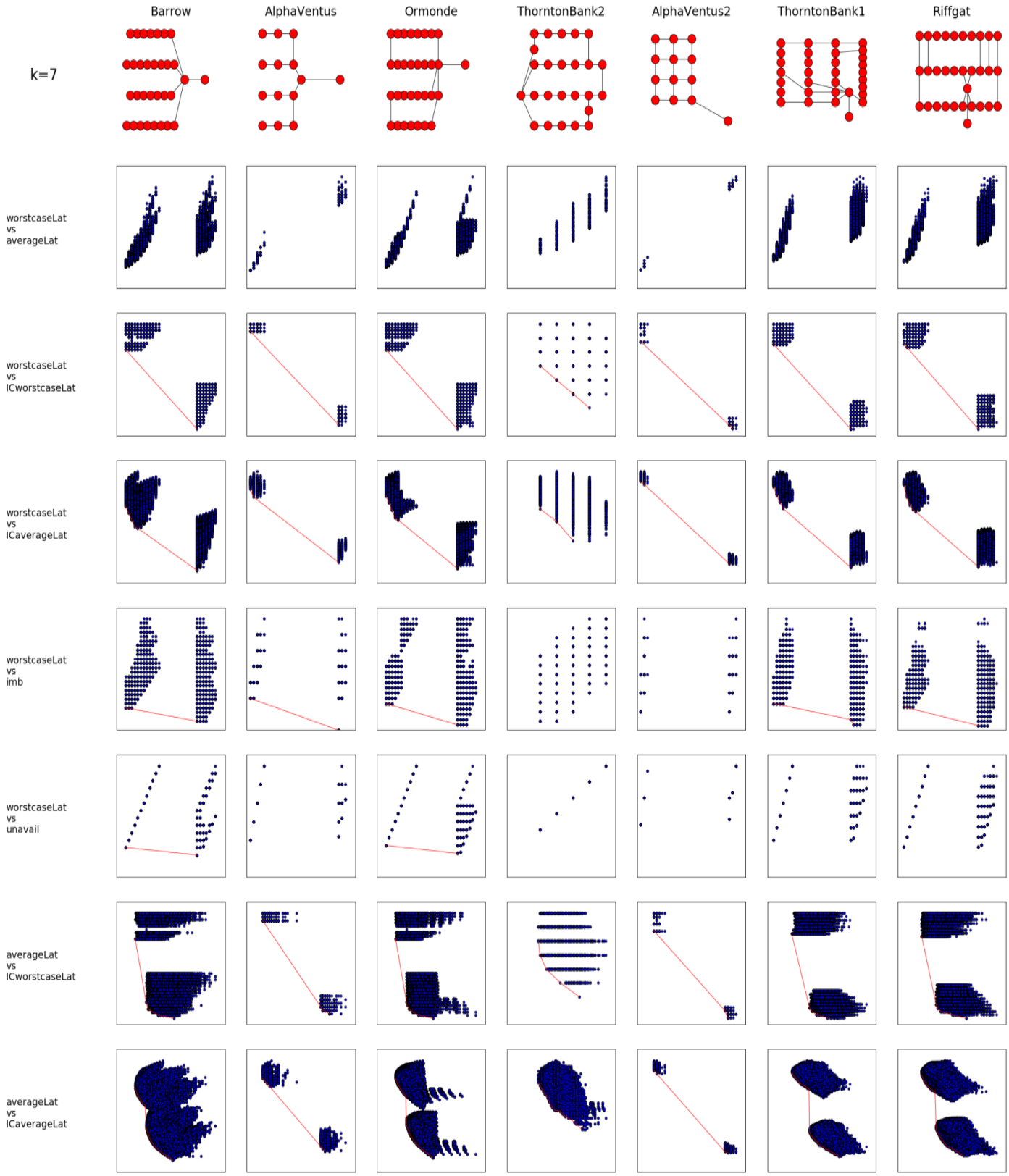

Figure 4 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=7 (1)

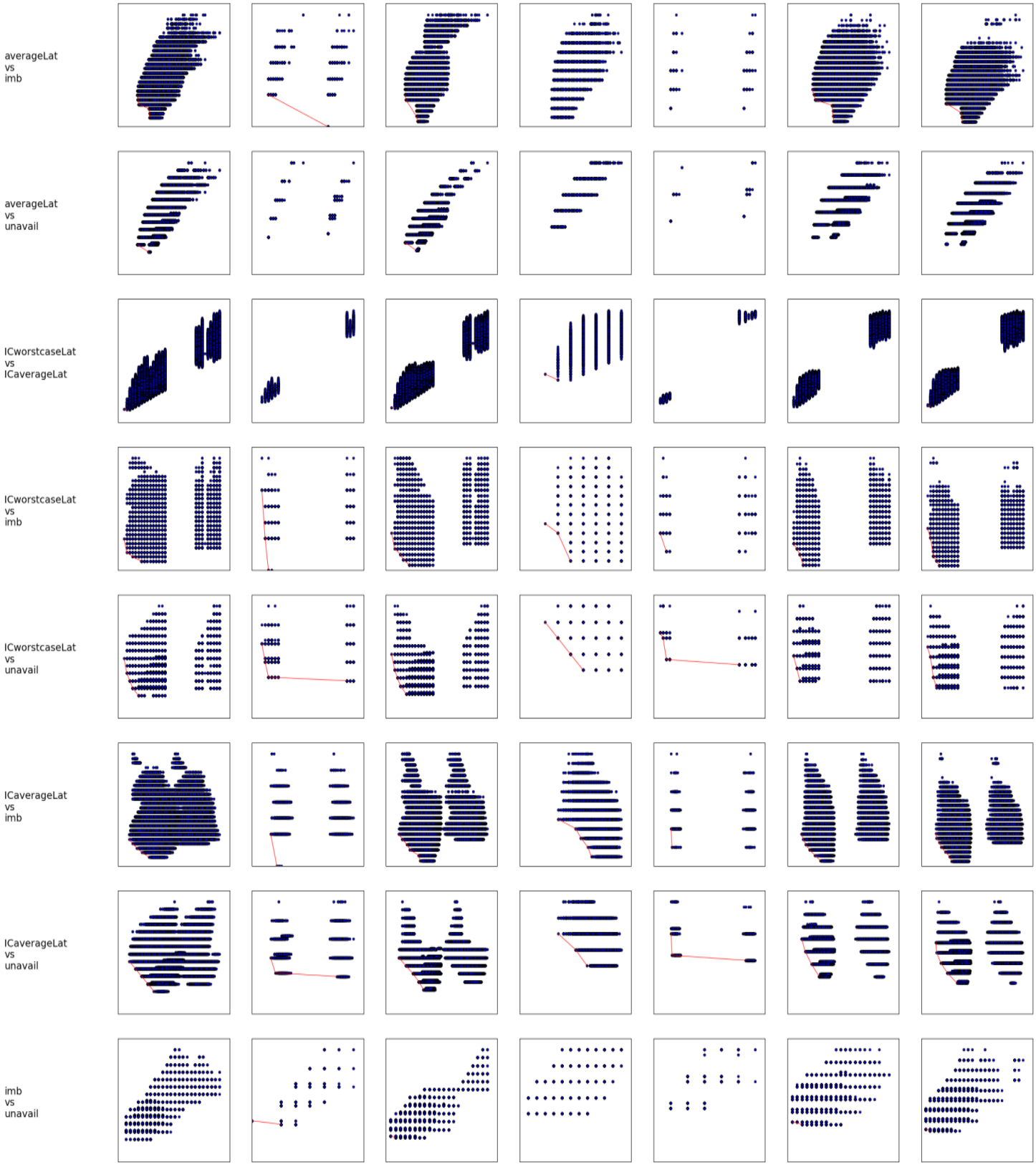

Figure 4 Exhaustive evaluation and Pareto Frontiers for all 6 metrics combinations K=7 (2)