

Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering

Cloud Computing and Network Infrastructure track

A.Y. 2025/2026

Graduation Session: September 2026

Fault-Tolerant Placement of Phasor Data Concentrators in Next-Generation Smart Grids

Supervisors:

Fulvio Risso

Co-Supervisors:

Stefano Galantino

Elisa Albanese

Luca Zuanazzi

Candidate:

Alessio Giliberti

Abstract

Smart grids rely on Information and Communication Technology (ICT) infrastructures to collect measurement data from Phasor Measurement Units (PMUs) and deliver it, through intermediate Phasor Data Concentrators (PDCs), to a Control Center (CC) responsible for state estimation and grid monitoring. The reliability of this communication chain is critical: any disruption caused by a link or computational node failure can compromise the timely delivery of measurements and, consequently, the observability of the power system. This thesis addresses the problem of placing PDCs within an ICT network so that the resulting PMU-to-CC communication paths are resilient to at least a single failure, whether affecting a link or a node, anywhere in the network.

Building on and extending previous work on PDC placement, this thesis introduces a classification of PMUs into essential (ePMU) and redundant (rPMU) categories. This distinction stems from the observation that, for the purpose of state estimation, it is not necessary to collect data from every PMU in the network; a properly chosen subset is sufficient to guarantee observability. However, this subset cannot be selected arbitrarily, since discarding entire regions of the network would compromise the quality and completeness of the state estimate. The selection must therefore respect observability constraints while also satisfying the maximum latency requirements imposed on measurement delivery. To solve this problem, the thesis develops and compares two algorithmic strategies: an incremental, two-phase approach that improves upon an existing placement solution by introducing resilient paths, and a joint optimization approach that determines PDC placement and resilient routing simultaneously, drawing on state-of-the-art techniques.

Both approaches are validated on randomly generated ICT network topologies of varying size, allowing their behavior and scalability to be assessed across different network scales. For each topology, static evaluations measure aspects such as the number of PDCs required and the overall efficiency and redundancy of the resulting data flows, while dynamic evaluations simulate single-point failures, crashing individual nodes or links, to verify that data delivery to the CC remains stable even after such disruptions. Across the tested scenarios, the resilient placement consistently preserves the percentage of data reaching the CC despite the loss of (at least) a single network element, confirming that the proposed solutions maintain full observability under failure conditions. These results demonstrate the effectiveness of the proposed algorithms in guaranteeing resilient, low-latency PMU data delivery, and provide a solid basis for their application to real-world smart-grid ICT infrastructures of arbitrary scale.

Acknowledgements

Alla mia Chiara, a mia madre e a mio padre.
Grazie di aver creduto in me.

Table of Contents

List of Figures VII

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Problem . . . 2
- 1.3 Thesis Structure . . . 2

2 State of the Art 4
- 2.1 Cloud, Edge, and Resilience in Synchrophasor Systems . . . 4
- 2.2 Resilient Placement Approaches and Works . . . 5
  - 2.2.1 Disjoint Path Routing . . . 7
- 2.3 Foundation Works . . . 7

3 Problem Formulation 9
- 3.1 General Problem . . . 9
- 3.2 Intent-Based Orchestration and Use Cases . . . 10
- 3.3 Formal Problem Definition . . . 10
- 3.4 Constraints . . . 11

4 Placement Algorithms 12
- 4.1 Two-step approach . . . 12
  - 4.1.1 Variables . . . 12
  - 4.1.2 Observability coefficient . . . 13
  - 4.1.3 Recursively Resilient algorithm . . . 15
- 4.2 Single-step approach . . . 23
  - 4.2.1 Greedy Edge Penalty algorithm . . . 23
  - 4.2.2 Tiered Min-Cost algorithm . . . 23
  - 4.2.3 k-Shortest Candidates algorithm . . . 29
  - 4.2.4 Unsuitable algorithms after evaluation . . . 32

5 Evaluation 34
5.1 Evaluation Setup . . . 34
5.2 Static evaluation . . . 35
5.3 Dynamic evaluation . . . 38

6 Future Enhancements 42
6.1 Generalizing the ePMU/rPMU Classification via Partial Observability 42
6.1.1 Observability from PMU Measurements . . . 42
6.1.2 An Illustrative Example: IEEE 14-Bus System . . . 43
6.1.3 Why the ePMU/rPMU Classification is Only Locally Well-Defined . . . 43
6.1.4 Toward a Per-Bus PMU Decomposition . . . 45
6.2 Algorithmic and Implementation Optimizations . . . 45
6.3 Waiting-Time Coordination and Stability-Oriented Placement . . . 46
6.4 Coupling the Electrical and ICT Grid Layers in Simulation . . . 46
6.5 Shared Risk Link Groups . . . 47

7 Conclusions 49
7.1 Summary of the Proposed Approach . . . 49
7.2 Summary of the Evaluation . . . 50
7.3 Final Remarks . . . 50

Bibliography 52

List of Figures

4.1 A complete binary tree of depth three. Nodes 1, 2, and 3 (light blue) are candidate nodes, with node 1 being the CC; nodes 4–7 (orange) are PMUs. 15

4.2 The three scenarios obtained by connecting node X (dashed, equivalent to node 4, both outlined in green) to each of the candidate nodes 1, 2, and 3 in turn. The candidate node X is connected to in each scenario is highlighted, and the resulting absolute variation $\Delta\rho(n)$ is reported below the corresponding tree. 16

4.3 The tree extended with nodes 8 and 9: node 8 is a child of node 1 and is itself a candidate node, while node 9, a sensor, is child of node 8. Node X (dashed, equivalent to node 4, both outlined in green) is being evaluated for connection to either node 3 or node 8 (dashed edges). The relative variation $\Delta_r\rho(n)$ for each candidate is reported below the tree; node 8, the best-ranked candidate, is highlighted. . . 16

4.4 The flow diagram of the Recursively Resilient algorithm. Clouds are used to return from the current instance of the recursive `choose` function, and to get the return value from the exiting instance. Return by raising an exception is colored in red. 20

5.1 Execution time distribution as a function of graph size 35

5.2 Ratio between placed PDCs and candidate locations as a function of graph size 36

5.3 ePMU latency distribution as a function of graph size 37

5.4 rPMU latency distribution as a function of graph size 37

5.5 Node Jain index as a function of graph size 38

5.6 Number of failures required to lose observability 39

5.7 Number of node failures required to lose observability 40

5.8 Number of edge failures required to lose observability 41

6.1 The IEEE 14-bus test network, with PMUs installed at buses 2, 6, 7, and 9 (highlighted). Every other bus is observed indirectly, via a directly-measured neighboring bus. 44

Chapter 1

Introduction

1.1 Context

Wide-area monitoring of electrical power systems increasingly depends on infrastructures able to deliver accurate, timely, and dependable situational awareness to grid operators. A central role in this process is played by Phasor Measurement Units (PMUs), sensor devices that sample electrical quantities (i.e. voltage, current amplitude and phase, frequency, Rate-Of-Change-Of-Frequency (ROCOF)) at high frequency and tag each measurement with a common time reference obtained through satellite synchronization. This time alignment allows measurements collected at physically distant points of the network to be compared and interpreted as a coherent, real-time picture of the grid's dynamic state.

As the number of deployed PMUs grows, so does the volume of data that must be collected, synchronized, and delivered to the control infrastructure. Transmitting every individual PMU stream directly to a central location does not scale well, both in terms of bandwidth and processing overhead. Phasor Data Concentrators (PDCs) address this issue by acting as intermediate aggregation points: each PDC collects streams from a set of PMUs, checks and re-aligns their timestamps, and forwards a consolidated stream either toward another PDC or directly to the Control Center (CC), where the data is used for state estimation, protection, and control functions.

Because these functions are time-critical, the way PDCs are organized and connected within the underlying ICT network has a direct impact on the quality of grid monitoring. Added latency, dropped packets, or a loss of synchronization at any point along the path from a PMU to the CC can degrade the operator's real-time picture of the system, with consequences ranging from delayed disturbance detection to incorrect control actions. This concern is amplified by the fact that real communication networks are not fault-free: links can become congested or unavailable, and nodes can fail or be taken offline for maintenance. If the placement

of PDCs does not account for these possibilities, a single failure can disconnect entire portions of the sensing infrastructure from the CC, undermining the resilience that wide-area monitoring is meant to provide in the first place.

This work is part of a broader collaboration with **RSE (Ricerca sul Sistema Energetico)**, an Italian research center dedicated to innovation in the energy sector, and continues a line of research previously focused on the orchestration and scalability of grid-monitoring services through cloud-native and multi-cluster architectures, which also produced early algorithms for positioning PDCs within the ICT network.

1.2 Problem

This thesis builds on this foundation by placing resilience at the center of the PDC placement problem: rather than optimizing PDC placement solely for latency or bandwidth, the goal becomes ensuring that PMU data can still reach the CC even after a single node or link failure occurs anywhere in the network.

A complementary idea explored in this work is that full resilience does not necessarily require protecting every single PMU stream. For the purposes of state estimation, a carefully chosen subset of PMUs is often sufficient to preserve full observability of the grid, provided the subset is not chosen arbitrarily but respects the topological structure of the network. Building on this observation, PMUs are classified into essential (ePMU) and redundant (rPMU) categories, a distinction that is later used to simplify the resilient placement problem without violating the maximum-latency requirements imposed on measurement delivery.

Two algorithmic strategies are proposed to tackle the resulting placement problem. The first proceeds in two phases, starting from an existing placement solution and subsequently reinforcing it with resilient, failure-tolerant paths. The second instead optimizes PDC placement and path resilience jointly, in a single step, drawing on optimization techniques from the current state of the art. Both strategies are tested on randomly generated network topologies of different sizes, and their performance is assessed both under normal operating conditions and after simulating node or link failures.

1.3 Thesis Structure

The thesis is structured as follows:

- *Chapter 2* reviews the relevant literature, covering cloud and edge computing paradigms in synchrophasor systems, existing self-healing and optimization-based approaches to PMU-PDC resilience, and the disjoint-path routing techniques underlying the single-step algorithms developed later, before presenting

the foundation work this thesis builds upon.

- *Chapter 3* formally defines the resilient PDC placement problem, introducing the underlying graph model, the essential/redundant (ePMU/rPMU) classification of sensors, and the constraints – resilience, latency, and redundancy-aware coverage – that any valid solution must satisfy.
- *Chapter 4* presents the resilient PDC placement algorithms developed in this thesis, organized into a two-step approach based on the observability coefficient, and a family of single-step approaches – Greedy Edge Penalty, k-Shortest Candidates, and Tiered Min-Cost – that jointly optimize path placement and resilience.
- *Chapter 5* evaluates the proposed algorithms on randomly generated ICT network topologies of varying size, comparing them against baseline placement strategies through both static metrics and dynamic, failure-simulation-based metrics.
- *Chapter 6* discusses possible extensions of this work, including a more granular, per-bus notion of PMU redundancy, algorithmic and implementation improvements, the remaining intent-based orchestration use cases, tighter electrical-ICT co-simulation, and explicit support for Shared Risk Link Groups.
- *Chapter 7* summarizes the contributions and results of the thesis, and offers final remarks on the proposed approach.

Chapter 2

State of the Art

2.1 Cloud, Edge, and Resilience in Synchrophasor Systems

Synchrophasor-based applications have been implemented using either cloud or edge computing paradigms, depending on system requirements such as latency, scalability, and resilience. Early cloud-based approaches, such as those by Maheshwari et al. [1] and Luo et al. [2], demonstrated the scalability and computational benefits of offloading PMU data processing and state estimation to the cloud, later confirmed at industrial scale by deployments such as GridCloud [3]. However, the geographical distance between cloud data centers and field devices introduces latency that is often incompatible with real-time, high-resilience monitoring requirements. This limitation motivated a shift towards edge computing, where processing is moved closer to data sources: works such as those by Meloni et al. [4], Pau et al. [5], and Galantino et al. [6] show that edge-based architectures can significantly reduce latency, at the cost of increased management complexity as the number of distributed applications grows. A common limitation across this body of work is that cloud and edge resources are typically treated as two separate silos rather than as a single pool of computing resources, which limits both the seamless deployment of applications and the resilience benefits that could arise from orchestrating them jointly across domains.

Resilience in cyber-physical systems generally requires prevention, detection, and restoration capabilities, with self-healing mechanisms playing a key role in enabling autonomous recovery from faults. In PMU–PDC architectures, self-healing has mainly been explored through Software-Defined Networking (SDN), where a centralized controller with a global view of the network reconfigures switches to

reroute data flows after a failure. Lin et al. [7] proposed an ILP[1]-based self-healing scheme that reconnects unaffected PMUs to healthy PDCs by reconfiguring SDN switches, while Qu et al. [8] optimized the efficiency of this process, and Duan and Dinavahi [9] introduced a fast path-recovery algorithm for single link failures based on a simplicity criterion. These SDN-based solutions share a network-oriented view of resilience and rely on strong assumptions – full centralized control over network equipment, costly infrastructure upgrades, and the availability of nearby PDCs with sufficient spare capacity at the time of failure – that do not always hold in realistic, heterogeneous network environments. More recently, Tzanis et al. [10] shifted the focus from the network to the PDC itself, proposing an approach that relocates virtualized PDCs via Virtual Machine (VM) migration in response to latency degradation; however, the recovery times associated with VM migration, on the order of thousands of seconds, limit its applicability, and, like the SDN-based approaches, it leaves the functional recovery of the PDC application itself unaddressed.

To overcome the siloed treatment of cloud and edge resources, recent research has proposed the computing continuum – also referred to as the *cloud-to-edge continuum* – as a unifying paradigm that exposes cloud and edge resources as a single, transparent environment enabling applications to be deployed dynamically across heterogeneous infrastructures [11, 12]. Building on this paradigm, recent work has begun to design self-healing mechanisms that restore PMU–PDC measurement systems at the application level, addressing the functional recovery gap left open by SDN-based rerouting and VM migration.

Overall, the reviewed literature tends to approach resilience *reactively*, restoring connectivity or functionality after a failure occurs; this thesis instead adopts a *proactive*, placement-oriented perspective, aiming to determine in advance a PDC placement and set of communication paths that already tolerate a single failure, complementing rather than replacing the reactive recovery mechanisms discussed above.

2.2 Resilient Placement Approaches and Works

Beyond the network- and application-level self-healing mechanisms discussed in the previous section, resilient sensor selection and placement have also been studied from a purely combinatorial optimization perspective. In particular, Quadratic Unconstrained Binary Optimization (QUBO) formulations have recently gained

[1]Integer Linear Programming (ILP) is an optimization technique in which a linear objective function is minimized or maximized subject to a set of linear constraints, with decision variables restricted to integer values.

attention as a way to encode selection and placement problems – including adversarial or failure-resilient variants – as quadratic objectives over binary variables, potentially enabling the use of specialized solvers, including quantum and quantum-inspired annealers. A QUBO-based formulation of the resilient PDC placement problem addressed in this thesis was considered as a possible direction; however, it was not pursued further, as it fell outside the scope of the contractual activities agreed upon with RSE and did not align with the topics of interest of the RSE researchers involved in the project.

A closely related formulation is proposed by Laszka et al. [13], who address the resilient selection of a fixed-size subset of sensors under a strategic adversary capable of disabling an arbitrary subset of the selected sensors. Modeling sensor observations using a Gaussian-process-based regression model, the authors formulate attack-resilient sensor placement as the problem of selecting a subset of observations that minimizes the uncertainty, i.e. the posterior variance, of the resulting predictions under the adversary's worst-case attack. They show that both the design problem and the attack problem are NP-hard, and propose efficient greedy heuristics with theoretical approximability guarantees, validated on real-world datasets. While conceptually appealing, this formulation is not directly applicable to the problem addressed in this thesis: Laszka et al. [13] consider a *flat* selection problem, in which sensors are simply chosen from a candidate set with no further structure, whereas the problem addressed here additionally requires building a *tree* of PDCs connecting every selected PMU to the CC. Adapting their approach would therefore require not only a different modeling of the underlying random variables, but also multiple iterations of the selection algorithm to progressively construct the aggregation tree up to each PMU, rather than a single flat selection step.

Another work more directly related to the PMU–PDC domain informed the design choices of this thesis. Islam et al. [14] propose a scalable two-stage PDC placement technique aimed at minimizing end-to-end delay while ensuring resiliency against a single PMU, PDC, or communication link failure. In the first, prescreening stage, plausible PDC configurations are identified through a genetic algorithm with a computationally efficient meta-heuristic to address scalability; in the second stage, Suurballe's and Dijkstra's algorithms are used to select the best configuration, incorporating the role of each PMU in power grid observability directly into the routing policy. This work is particularly relevant to this thesis, as it distinguishes PMUs by their contribution to grid observability with the essential/redundant (ePMU/rPMU) classification adopted here, and shows how such a distinction can be exploited to achieve resilience without requiring every PMU to remain connected to the PDC hierarchy after a failure.

Pourramezan et al. [15] address the optimal coordination of PDC waiting times in hierarchical synchrophasor networks, building on the IEEE Standard C37.247-2019 [16] for PDCs. They propose a mathematical model of the synchrophasor

hierarchy that incorporates both the *absolute* and *relative* wait-time policies required by the standard, and optimally design these wait times to maximize application throughput and grid observability within given time constraints, validating their method on realistic PMU data. This work could have provided useful guidance on how waiting-time configuration interacts with end-to-end latency across multiple PDC levels, if the 2nd idea in Section 3.2 was developed further.

2.2.1 Disjoint Path Routing

The single-step optimization approach developed in this thesis draws on the literature on disjoint path routing, which provides algorithmic tools for computing multiple paths between a source and a destination that do not share failure-prone elements. Iqbal and Kuipers [17] survey the disjoint paths problem, covering its basic formulation, complexity results, and several variants – including availability-based, maximally disjoint, domain-disjoint, shared risk link group (SRLG)-disjoint, region-disjoint, and disjoint path pair problems – all aimed at making network traffic more robust to node or link failures. Bhandari [18] addresses the related problem of optimal diverse routing in telecommunication fiber networks, proposing an algorithm for computing the shortest pair of physically disjoint paths between two nodes, which can be used either as a primary/backup pair or to split traffic so that a single node or link failure does not cause a complete loss of data. Bermond et al. [19] study SRLG-diverse routing in networks where SRLGs satisfy the so-called star property – i.e., all links subject to a given shared risk group are incident to a common node – and, after disproving a previously proposed polynomial-time algorithm for this setting, show the general problem to be NP-complete while providing polynomial-time algorithms for several practically relevant subcases, such as a constant number of SRLGs or bounded vertex degree. These works provide the algorithmic foundations for the path-diversity constraints used in the single-step placement algorithms presented in Chapter 4.

2.3 Foundation Works

A solid foundation for the work carried out in this thesis was provided by Bagnara [20]. The author developed a substantial collection of Python scripts, built on top of the NetworkX library, to generate random network topologies, execute placement algorithms on them, and visualize the resulting solutions. These scripts are, in turn, used by his orchestrator to translate a given graph into a simulated network, in which every node is realized as a separate k3s cluster; these clusters receive directives specifying which pods to instantiate – corresponding to PMUs or PDCs – and how these pods should connect to one another in order to exchange synchrophasor data.

Crucially, this orchestrator interacts with the underlying placement logic exclusively through a well-defined API, which takes as input the network topology together with the positions of the PMUs and the CC, and returns the resulting PDC placement and the corresponding paths, independently of the specific algorithm used to compute them. This exposes the placement algorithm as an interchangeable component rather than a fixed part of the system. As a result, the work presented in this thesis is purely algorithmic in nature: by conforming to Bagnara's placement function API, the algorithms developed here can be plugged directly into his existing orchestrator, without requiring any modification to the underlying infrastructure, the cluster management logic, or the graph-generation utilities. The only substantive change is in the objective guiding the placement itself: rather than positioning PDCs so as to minimize end-to-end latency, as Bagnara's original algorithms do, the algorithms introduced in this thesis position them so as to maximize resilience to a single failure, within the same latency constraint.

Chapter 3

Problem Formulation

3.1 General Problem

In a distributed monitoring architecture, multiple services are responsible for collecting data from sensors deployed across a network and delivering it to a final application for processing and decision-making. This general scenario, of which the smart-grid monitoring infrastructure described in Section 1.1 is a specific instance, gives rise to four main challenges:

- ensuring that the data-collection system remains resilient to failures occurring within the underlying ICT infrastructure;
- meeting the latency requirements imposed by the final application, which determine how quickly data must reach it to remain useful;
- enabling efficient management and maintenance of the nodes composing the distributed architecture, as well as of the applications running on them;
- supporting the scalability of data streaming and processing as the number of sensors, and consequently the volume of measurements, grows over time.

These challenges are not independent: a design choice that improves resilience, for instance by introducing redundant communication paths, may increase latency or complicate maintenance, while a configuration optimized purely for scalability may leave the system exposed to single points of failure. A sound design of the data-collection infrastructure must therefore reconcile these competing requirements rather than optimizing for one of them in isolation.

3.2 Intent-Based Orchestration and Use Cases

Given this general problem, the collaboration this thesis is part of has identified an intent-based orchestration approach as a promising direction: rather than manually configuring the placement of intermediate aggregation nodes (PDCs), a network administrator specifies a high-level intent – for example, a maximum acceptable end-to-end latency and the system automatically determines a PDC placement that satisfies it. Three complementary use cases, or intents, have been identified around this idea:

1. Intent of latency and stability. Given a graph of PMU and PDC nodes, characterized by processing latency, and links, characterized by transmission latency, place PDCs so that end-to-end latency and the reconfiguration effort required after a failure is minimized, thus maximizing the stability of the placement.

2. Intent of scalability and availability. Given a fixed deployment of PMUs and a telecommunication network described in terms of link latency and bandwidth, find the optimal position of a set of low-level PDCs and a single high-level PDC, together with their waiting-time configuration, so as to minimize latency and thus maximize the availability of phasor data at the final application, including under dynamic conditions such as an increase in the number of PMUs or in channel latency.

3. Intent of resilience. Given the same graph model as above, place the PDCs so that the network is resilient to at least one failure among PMUs, (low-level) PDCs, or links, accounting for the fact that part of the PMUs are redundant (rPMU) and provide information that is equivalent, or correlated, to that of their corresponding essential PMUs (ePMU) for the purpose of state observability.

While these three intents share the same underlying graph model and address complementary aspects of the same general problem, this thesis focuses specifically on the third one – resilience to single failures – building on the ePMU/rPMU classification it introduces, while remaining a natural extension of the reconfiguration and stability concerns raised by the first intent.

3.3 Formal Problem Definition

The infrastructure is modeled as a graph in which nodes represent PMUs and PDCs, each characterized by a processing latency, and edges represent communication links, each characterized by a transmission latency. PMUs are further partitioned

into essential PMUs (ePMU) and redundant PMUs (rPMU): an rPMU provides information that is equivalent to, or strongly correlated with, that of one or more corresponding ePMUs, so that, for the purpose of state estimation, it is not strictly necessary for data from every PMU to reach the Control Center (CC), provided a sufficient and appropriately chosen subset does.

Given this graph and the fixed positions of the PMUs, the problem addressed in this thesis is to determine communication paths connecting each PMU to the CC, and the resulting placement of PDCs, such that the system remains resilient to at least one failure – affecting a PMU, a PDC, or a link – occurring anywhere in the network, while the end-to-end latency experienced by the data of every ePMU, along at least one surviving path, does not exceed a maximum latency threshold defined by the application.

3.4 Constraints

The problem is subject to the following constraints:

- **Single-failure resilience**. For every ePMU, at least one path to the CC must remain available after the failure of any single node (PMU or PDC) or link elsewhere in the network. Simultaneous multiple failures are not considered.
- **Latency bound**. Along at least one available path from each ePMU to the CC, the sum of the processing latencies of the traversed nodes and the transmission latencies of the traversed links must not exceed the maximum latency $L_{\max}$ required by the final application, both under normal operation and after a single failure.
- **Redundancy-aware coverage**. It is not required that data from every PMU reach the CC after a failure; however, the subset of PMUs whose data is guaranteed to arrive must be sufficient to preserve full observability of the grid state, which in practice requires that the classification into ePMU and rPMU – and hence the selection of which PMUs may be excluded – cannot be arbitrary, but consistent with the topological and observability structure of the underlying sensor deployment.
- **Genericity of the model**. Although the problem is formulated, and later evaluated, in the context of PMU and PDC infrastructures, none of the algorithms developed in this thesis rely on properties specific to phasor measurement systems. The underlying model – sensor nodes generating data, intermediate aggregation nodes, and a final collection point, connected by a network subject to latency constraints and single-point failures – is generic, and the results obtained apply more broadly to other distributed data-collection architectures with analogous requirements.

Chapter 4

Placement Algorithms

When first considering how to maximize resilience, the most straightforward answer was to have every low-level PDC (i.e., every PDC directly connected to one or more PMUs) flood its data toward every PDC connected to it: since data would then be duplicated to the greatest extent possible, this would yield maximal resilience. This approach, however, does not scale at all, and moreover ignores a major concern for grid operators, namely the available bandwidth: the data (measurements) transmitted must therefore be kept to a minimum.

A more refined idea was to duplicate the measurements of each PMU along two opposite directions, reaching the CC via paths that are as disjoint as possible. This is not directly feasible, since duplicating data is precisely what we want to avoid; the underlying idea, however, is not without merit. Indeed, recalling that not all PMUs give different information (for the state estimation), but that each ePMU is associated with at least one rPMU, it suffices to route the paths of these two associated PMUs so that they are maximally disjoint, allowing the pair to survive a single failure. This idea closely resembles the one underlying the work of Islam et al. [14].

4.1 Two-step approach

To understand how the Recursively Resilient algorithm works, it is important to first formalize the quantities it operates on.

4.1.1 Variables

The considered variables are:

- $N \in \mathbb{N}_+$ is the number of ePMUs.

- $M \in \mathbb{N}$ is the number of rPMUs.

- For a node n in the network, $\underline{x}(n) \in \{0,1\}^{(N+M)\times 1}$ is the binary column vector representing which PMU data flows pass through n: the i-th entry $x_i(n) = 1$ if and only if n receives data originating from PMU_i.

- $\underline{v} \in \mathbb{R}_+^{(N+M)\times 1}$ is the real column vector in which each entry v_i represents the "value" of the corresponding PMU, allowing some sensors[1] to be regarded as more important than others. Unlike $\underline{x}(n)$, $\underline{v}$ is a fixed, node-independent quantity.

- $\mathbf{R} \in \{0,1\}^{(N+M)\times(N+M)}$ is the redundancy matrix, whose elements are defined as

$$r_{ij} = \begin{cases} 0, & \text{if } i = j \\ 1, & \text{if } \text{PMU}_i \equiv \text{PMU}_j \\ 0, & \text{otherwise} \end{cases}$$

  In the remainder of this thesis, the redundancy matrix $\mathbf{R}$ will be treated as binary. It could, however, be generalized to the case where sensors are not strictly equivalent but only correlated to some degree, by allowing $r_{ij} \in [-1,1]$ to represent the strength of this correlation; this generalization is left outside the scope of the present work.
  Note that the redundancy matrix $\mathbf{R}$ is symmetric by construction, since $\text{PMU}_i \equiv \text{PMU}_j$ if and only if $\text{PMU}_j \equiv \text{PMU}_i$ (i.e., $r_{ij} = r_{ji}$, $\forall i, j$). Moreover, since equivalence is a transitive relation, if $\text{PMU}_i \equiv \text{PMU}_j$ then PMU_i and PMU_j are equivalent to exactly the same set of sensors, and hence the i-th and j-th rows (equivalently, columns) of $\mathbf{R}$ coincide.

4.1.2 Observability coefficient

The redundancy matrix $\mathbf{R}$ naturally induces a partition of the PMU set, which is formalized by the following definition.

[1] Throughout this thesis, the terms *sensor* and *PMU* are used interchangeably. PMUs represent the specific use case considered in this work, but the definitions and algorithms presented here are not inherently tied to synchrophasor measurements, and can in principle be generalized to other measurement systems composed of sensors organized in a similar hierarchical structure.

Definition 1: Sensor Partition into Equivalence Classes

Let S be the set of sensors, with $|S| = N + M$, and let $\mathbf{R}$ be the redundancy matrix associated with S. Let $S^* = \{E_1, E_2, \ldots, E_k\}$ be the partition of S induced by $\mathbf{R}$, that is, a collection of pairwise disjoint, non-empty subsets of S such that

$$E_i \cap E_j = \emptyset \quad \forall i \neq j, \qquad \wedge \qquad \bigcup_{i=1}^{k} E_i = S,$$

where two sensors $s, s' \in S$ belong to the same subset $E \in S^*$ if and only if $r_{ss'} = 1$. Each subset $E \in S^*$, referred to as an **equivalence class**, therefore contains sensors that are mutually equivalent, in the sense that any sensor in E can be substituted by any other sensor in E without loss of information.

Definition 2: Observability Coefficient

Let n be a node of the network, let $\underline{x}(n) \in \{0,1\}^{|S|}$ be the selection vector defined above, and let $\underline{v} \in \mathbb{R}_+^{|S|}$ be the value vector assigning a non-negative value v_s to each sensor $s \in S$. The **observability coefficient** of node n, associated with $\underline{x}(n)$, $\underline{v}$, and the partition S^* of Definition 1, is the non-negative real number

$$\rho(n) \;=\; \sum_{E \in S^*} \max_{s \in E} \; v_s \, x_s(n).$$

For each equivalence class $E \in S^*$, the term $\max_{s \in E} v_s \, x_s(n)$ selects the value of the sensor of maximum value among those in E whose data reaches n (i.e., with $x_s(n) = 1$); if no sensor in E reaches n, the corresponding term is 0, since $x_s(n) = 0$ for all $s \in E$. The observability coefficient of n is then given by the sum, over all equivalence classes, of these maximal achieved values.

Equivalently, since each equivalence class contributes to $\rho(n)$ through at most one of its sensors, the observability coefficient $\rho(n)$ of a node n can be interpreted as (a weighted count of) the number of *independent* PMU data flows passing through n, where two flows are considered independent if and only if they originate from sensors belonging to different equivalence classes of S^*.

This interpretation motivates the choice of name for the coefficient. When selecting a path for a PMU's data flow, this flow is routed through a sequence of intermediate nodes (i.e., PDCs). The variation $\Delta\rho(n)$ induced at a node n by adding a flow to its incoming paths indicates whether doing so genuinely improves the observability of the system at n: if the added flow originates from a sensor belonging to an equivalence class not yet represented among the flows already

reaching n, or from a sensor of higher value than the one currently selected for its class, then $\Delta\rho(n) > 0$; otherwise, if the flow is redundant with respect to the flows already reaching n, then $\Delta\rho(n) = 0$.

The observability coefficient, as defined above, relies on the redundancy matrix $\mathbf{R}$ being binary. The coefficient is, however, generalizable to the case where $\mathbf{R}$ is not binary.

4.1.3 Recursively Resilient algorithm

In what follows, the rationale behind the algorithm is presented before its actual implementation, so as to clarify how the algorithm follows naturally from the observability coefficient.

Rationale

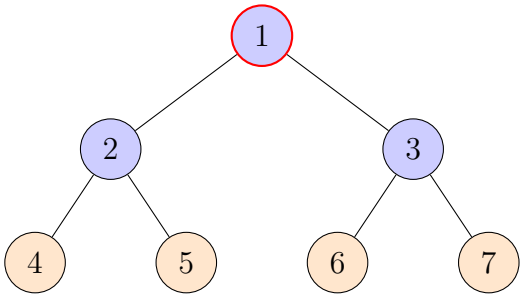


Figure 4.1: A complete binary tree of depth three. Nodes 1, 2, and 3 (light blue) are candidate nodes, with node 1 being the CC; nodes 4–7 (orange) are PMUs.

Consider a complete binary tree of depth three, having 7 $(4 + 2 + 1)$ nodes, as shown in Figure 4.1. Suppose further that only the first three nodes can host computation (i.e., they are *candidate* nodes), that node 1 is the CC, and that nodes 4, 5, 6, and 7 are sensors.

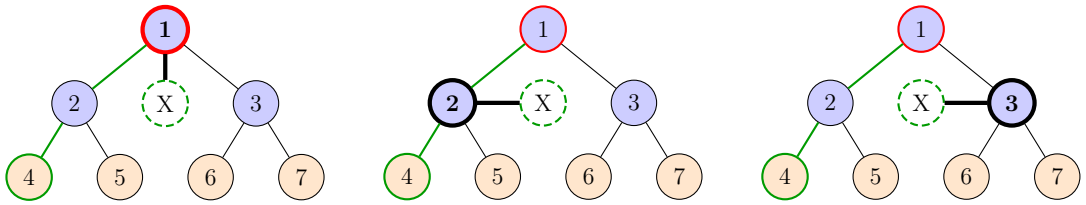
Suppose one wishes to attach an additional node, denoted X, to this tree, knowing that it is equivalent to node 4: how can this be decided objectively? The observability coefficient provides an answer. By computing the variation

$$\Delta\rho(n) = \rho_{\text{after}}(n) - \rho_{\text{before}}(n),$$

or, more formally, using the state of the node n:

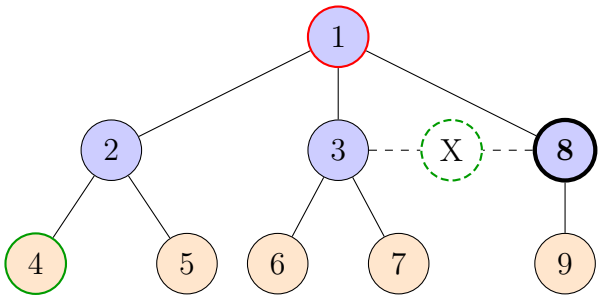
$$\Delta\rho(n) = \underbrace{\rho\left(\begin{bmatrix} \underline{x}(n) \\ x_X(n) = 1 \end{bmatrix}\right)}_{\rho_{\text{after}}(n)} - \underbrace{\rho\left(\begin{bmatrix} \underline{x}(n) \\ x_X(n) = 0 \end{bmatrix}\right)}_{\rho_{\text{before}}(n)}$$

using Definition 2, before and after connecting node X to each of the three candidate nodes, one obtains the values shown in Figure 4.2.



$\Delta\rho(\mathbf{1}) = 0$ $\Delta\rho(\mathbf{2}) = 0$ $\Delta\rho(\mathbf{3}) = 1$

Figure 4.2: The three scenarios obtained by connecting node X (dashed, equivalent to node 4, both outlined in green) to each of the candidate nodes 1, 2, and 3 in turn. The candidate node X is connected to in each scenario is highlighted, and the resulting absolute variation $\Delta\rho(n)$ is reported below the corresponding tree.



$\Delta_r\rho(1) = 0$ $\Delta_r\rho(2) = 0$ $\Delta_r\rho(3) = 0.5$ $\Delta_r\rho(\mathbf{8}) = 1$

Figure 4.3: The tree extended with nodes 8 and 9: node 8 is a child of node 1 and is itself a candidate node, while node 9, a sensor, is child of node 8. Node X (dashed, equivalent to node 4, both outlined in green) is being evaluated for connection to either node 3 or node 8 (dashed edges). The relative variation $\Delta_r\rho(n)$ for each candidate is reported below the tree; node 8, the best-ranked candidate, is highlighted.

However, the absolute variation $\Delta\rho(n)$ alone is not always sufficient to identify the best candidate node. To see why, consider extending the tree with two further nodes, 8 and 9: where node 8 is a child of node 1 and is also made a candidate node, and node 9, a sensor, is child of node 8. Suppose that node 9 already routes a single flow through node 8, so that $\rho_{\text{before}}(8) = 1$, while node 3 already routes two distinct, non-equivalent flows, so that $\rho_{\text{before}}(3) = 2$. If connecting node X to either node 3 or node 8 contributes a new, non-redundant flow to both, then

$$\Delta\rho(3) = 3 - 2 = 1 \qquad \text{and} \qquad \Delta\rho(8) = 2 - 1 = 1,$$

that is, the two candidates are indistinguishable in terms of absolute variation, despite node 8 already carrying more traffic than node 3 before the addition of X. Using the relative variation instead yields

$$\Delta_r\rho(3) = \frac{\Delta\rho(3)}{\rho_{\text{before}}(3)} = \frac{1}{2} = 0.5, \qquad \Delta_r\rho(8) = \frac{\Delta\rho(8)}{\rho_{\text{before}}(8)} = \frac{1}{1} = 1,$$

correctly favoring node 8 over node 3, since the latter is already handling more flows and would therefore benefit less, in terms of resilience and bandwidth usage, from the addition of a further one. Figure 4.3 shows the extended tree, with the value of $\Delta_r\rho(n)$ reported below the figure.

Therefore, in order to also obtain a traffic-distribution benefit in the implementation, the *relative variation*

$$\Delta_r\rho(n) = \frac{\Delta\rho(n)}{\rho_{\text{before}}(n)}$$

is used instead of the absolute one, so as to penalize nodes that already carried more flows than others *before* the addition of the new one.

Using the relative variation, it is then possible to rank the candidate nodes from best (highest $\Delta_r\rho(n)$) to worst, and to route the flow of node X through the best-ranked one.

There is, however, still a missing piece: computing observability coefficients presupposes the existence of a starting tree over which they can be evaluated. This is precisely where the work of a previous colleague, Bagnara [20], proves useful, providing an algorithm for constructing such an initial tree.

The actual execution of the approach is therefore split into two phases. In the first phase, a base tree connecting the ePMUs to the CC is constructed using one of the algorithms proposed by Bagnara [20], by minimizing end-to-end latency, which also ensures the best possible paths are used under normal, fault-free operation. In the second phase, the Recursively Resilient algorithm is executed on top of this base tree to determine the paths for the rPMUs, with the objective of maximizing the resilience of the overall tree, subject to the constraint that the maximum latency requirement is still satisfied, rather than being further minimized.

Flow chart

The resilient placement procedure, illustrated in Figure 4.4, extends the two-phase approach outlined above to account for redundant PMUs (rPMUs), i.e., PMUs that are not strictly required to satisfy the observability requirement, but whose inclusion can improve the network's resilience. The algorithm is organized into two cooperating procedures, represented in the flowchart as two nested swimlanes: an outer driver, `place_pdcs_resiliently`, and an inner recursive routine, `choose`,

responsible for identifying a feasible, latency-compliant path from the Control Center (CC) to a given rPMU.

Outer procedure. The outer procedure first partitions the set of PMUs into the N ePMUs and M rPMUs introduced in Section 4.1.1. PDC placement for the ePMUs is delegated to the existing placement function (e.g., the greedy heuristic of Bagnara [20]), producing an initial set of PDC locations together with the corresponding CC-to-PMU paths. These paths are used to build a prefix tree rooted at the CC, which compactly encodes the topology already induced by the essential placement and serves as the search space explored by the recursive procedure. For each rPMU, the algorithm first computes, as a fallback, the shortest end-to-end path between the CC and the rPMU via Dijkstra's algorithm on the full graph. It then attempts to find a preferable path by invoking `choose` on the prefix tree, starting from the CC. If the recursive search fails to return a valid path – signalled by an exception – the algorithm falls back to the previously computed shortest path, ensuring that a solution is always produced for every rPMU, possibly at the cost of not reusing any existing PDC infrastructure.

Recursive procedure `choose`. The core of the algorithm is the recursive procedure `choose`, which, given the current node under consideration (initially the CC) and the target rPMU, attempts to extend the prefix tree towards the rPMU while satisfying the latency and observability criteria discussed below.

At the top of the recursion, an optional constraint – `cc_successors_constraint` – determines the candidate set of successor nodes to explore: when disabled, and the current node coincides with the CC, the search is not restricted to the children already present in the prefix tree, but is instead extended to all of the CC's neighbours in the graph, widening the search space at the root of the recursion. Being configurable through a boolean flag, this behaviour is depicted with a dashed decision block.

If the rPMU is directly reachable from the current node – either through the prefix tree or through a direct graph edge – the base case of the recursion is reached and the corresponding trivial path is returned. Otherwise, for every candidate node (i.e., a node capable of hosting a PDC) among the children of the current node, the algorithm computes the end-to-end latency of the path obtained by concatenating

i. the path from the CC to the candidate, as encoded in the prefix tree, and,

ii. the shortest path from the candidate to the rPMU.

Candidates whose end-to-end latency exceeds the maximum admissible latency $L_{\max}$ are discarded. A second, configurable constraint – again a dashed decision block – additionally discards candidates whose latency exceeds that of the parent

node, so as to restrict the search to children that are, in principle, latency-improving with respect to the current position in the tree.

For each surviving candidate node n, the algorithm evaluates the relative variation $\Delta_r\rho(n)$ introduced in the Rationale above, instantiated here as

$$\Delta_r\rho(n) = \begin{cases} \rho_{\text{after}}(n), & \text{if } \rho_{\text{before}}(n) = 0, \\ \dfrac{\rho_{\text{after}}(n) - \rho_{\text{before}}(n)}{\rho_{\text{before}}(n)}, & \text{otherwise,} \end{cases}$$

where $\rho_{\text{after}}(n)$ and $\rho_{\text{before}}(n)$ denote the observability coefficient of n computed with and without the rPMU under consideration routed through n, respectively – the special case $\rho_{\text{before}}(n) = 0$, left undefined by the general formulation of $\Delta_r\rho(n)$, being resolved here by directly returning $\rho_{\text{after}}(n)$. Candidates are first filtered to retain only those achieving the best relative variation, within a numerical tolerance, and the resulting set is then sorted by increasing end-to-end latency. The procedure subsequently attempts, in order, to recurse into each of these candidates.

Should none of the recursive calls succeed, the algorithm falls back to selecting the best candidate at the current level via its Dijkstra shortest path to the rPMU. Before finalizing this choice, however, a *cross-level consistency check* is performed: the best candidate identified at the current recursion level is compared against the best candidates discarded at shallower levels of the recursion (i.e., closer to the CC). If a superior candidate was available at an earlier level, the current selection is rejected, forcing the recursion to backtrack and allowing the calling level to explore alternative candidates instead. This check prevents the algorithm from committing to a locally optimal but globally suboptimal placement. Finally, an additional safeguard verifies that the selected fallback path does not loop back through the parent node – a situation that may arise only when the parent-latency constraint is disabled – and raises an exception in that case, again triggering backtracking at the caller level.

It is worth noting that the flowchart intentionally omits certain low-level implementation details that do not affect the overall control flow.

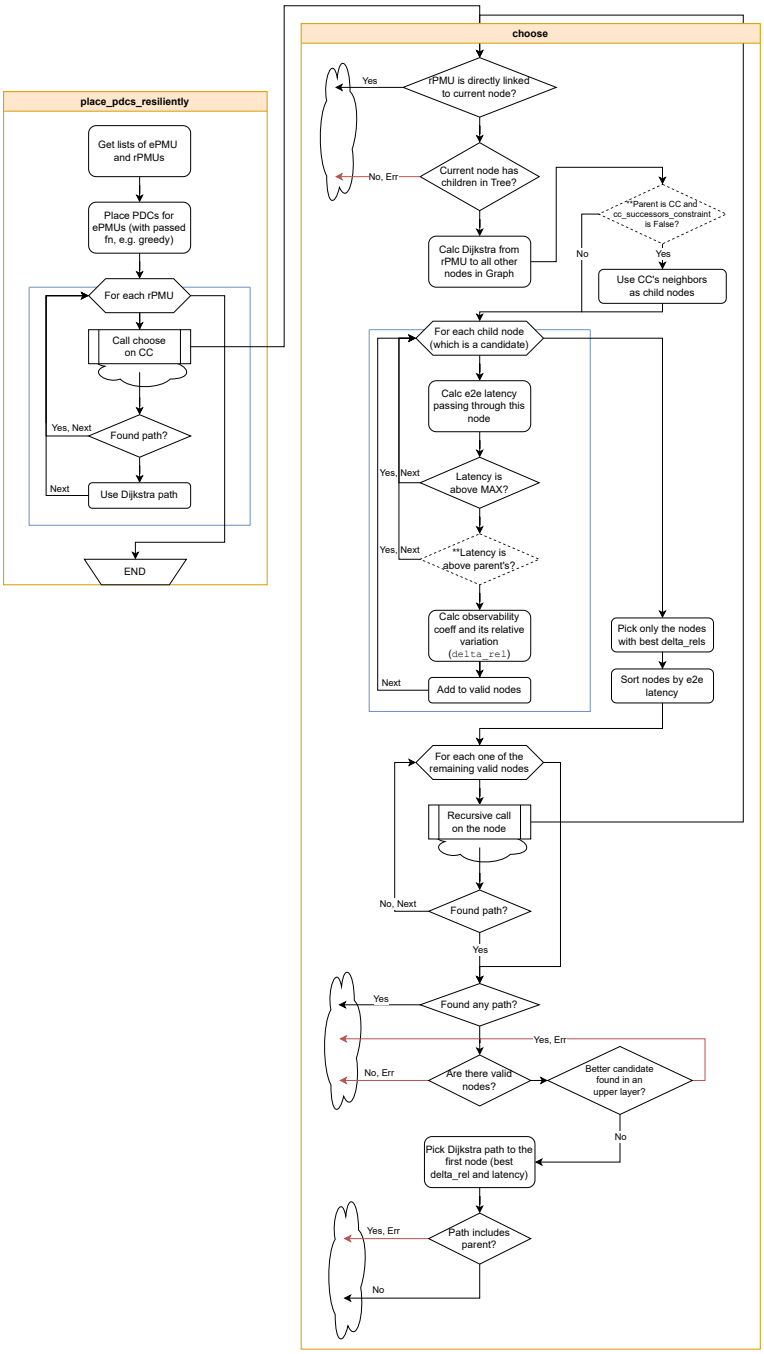


Figure 4.4: The flow diagram of the Recursively Resilient algorithm. Clouds are used to return from the current instance of the recursive `choose` function, and to get the return value from the exiting instance. Return by raising an exception is colored in red.

Computational complexity analysis

This section derives the time and space complexity of the Recursively Resilient algorithm presented before.

Notation. Let

- $V = |V(G)|$ and $E = |E(G)|$ denote the number of nodes and edges of the graph G;
- $P = |\text{PMUs}|$, with $P = N + M$, where N and M denote, respectively, the number of essential and redundant PMUs;
- T denote the prefix tree built from the essential-PMU paths, with $|V(T)| \leq V$;
- D denote the depth (height) of T, with $D \leq V$ in the degenerate case of a path-like tree.

The complexity of the first-phase placement routine, e.g. `place_pdcs_greedy`, is treated as an external black box, denoted $T_{\text{greedy}}(V, E)$, and is not re-derived here.

Outer procedure. The only significant part of the outer procedure is constructing the prefix tree from the N essential-PMU paths which costs $O(N \cdot V)$ in the worst case, since each path has length at most V and every node of every path is inserted into the tree exactly once. But this cost is dominated by the per-rPMU processing analyzed below and are therefore not further discussed.

Cost of a single invocation of `choose`. Consider one call to `choose` for a fixed target `rpmu`, with a given `parent` node and a candidate set `nodes` of size k. Ignoring the cost of the recursive calls themselves, this invocation performs:

1. **Single-source shortest paths.** Computes Dijkstra's shortest paths from `rpmu` to *every* node of G, at a cost of $O((V + E) \log V)$ using a binary-heap implementation.
2. **Ancestor retrieval.** For each of the k candidates, `all_predecessors` walks up the tree to the root, at a cost of $O(D)$ per candidate, i.e., $O(kD)$ overall.
3. **Observability coefficient evaluation.** For each candidate, the algorithm computes the PMU-reachability vector twice (with and without the candidate rPMU). This operation iterates over all P PMUs and, for each one checks which are connected to the node, whose cost is bounded by the size of the subtree explored, i.e., $O(V)$ in the worst case. This yields a cost of $O(P \cdot V)$ per reachability query, hence $O(k \cdot P \cdot V)$ for all candidates at this level.

The subsequent coefficient computation, performs a nested loop over the P-dimensional vector, costing $O(P^2)$ per candidate, hence $O(kP^2)$ overall.

4. **Filtering and sorting.** Selecting the candidates with the best relative variation and sorting them by latency costs $O(k \log k)$.

The cost of a single invocation, excluding recursive sub-calls, is therefore

$$T_{\text{call}} = O\Big((V + E) \log V + k(D + PN + P^2 + \log k)\Big).$$

Aggregate cost of the recursion for a single rPMU. Because the for-loop over candidates recurses depth-first and backtracks upon failure without ever re-invoking `choose` on an already-visited node, and because each node of T is the `parent` argument of at most one invocation (its unique parent-child relation in the tree), the total number of invocations of `choose` during the search for a single rPMU is bounded by $|V(T)| \leq V$. Furthermore, since every node of T is counted as a "candidate" exactly once (in the invocation relative to its own parent), the sum of the candidate-set sizes k across all invocations is bounded by the number of edges of T, i.e., $\sum k \leq V - 1$.

Combining these two bounds, the dominant contributions to the total cost of a single rPMU search are:

$$\begin{aligned} T_{\text{Dijkstra}} &= O\Big((V + E) \log V\Big), \\ T_{\text{reach}} &= O\Big(P \cdot V^2\Big), \\ T_{\text{coeff}} &= O\Big(V \cdot P^2\Big), \\ T_{\text{anc}} &= O(V \cdot D) = O(V^2), \end{aligned}$$

so that, since $P \leq V$ (as PMUs are a subset of the candidate nodes) and $D \leq V$,

$$T_{\text{rPMU}}(V, E, P) = O\Big((V + E) \log V\Big) + O\Big(PV^2\Big).$$

Total complexity. Repeating the above for all M redundant PMUs, and adding the (lower-order) cost of the fallback shortest-path computation performed once per rPMU in the outer procedure, the overall time complexity is

$$T_{\text{RR}} = T_{\text{greedy}}(V, E, P) \; + \; O\Big(N \cdot V\Big) \; + \; O\Big(M \cdot \Big[(V + E) \log V + PV^2\Big]\Big).$$

We can further simplify it by recalling $T_{\text{greedy}} = O\Big(P \cdot V(E + V \log V) + P^2 \cdot V\Big)$ and $N, M \leq P$:

$$T_{\text{RR}}(V, E, P) = O\Big(P \cdot V(E + V \log V) + P^2 V^2\Big)$$

which is polynomial, but of high degree with respect to the network size V in the worst-case scenario.

4.2 Single-step approach

Having observed that the Recursively Resilient algorithm presented above is only concerned with determining the paths for the rPMUs, it is natural to ask whether an algorithm exists that instead optimizes the paths of all PMUs simultaneously, so as to maximize resilience while remaining within the latency constraint.

In what follows, algorithms are therefore presented that seek maximally disjoint paths from two (or more) sources – the PMUs belonging to the same equivalence class (Definition 1) – toward the same destination, the CC: so these algorithms are executed once per equivalence class.

4.2.1 Greedy Edge Penalty algorithm

Among the algorithms surveyed by Iqbal and Kuipers [17], the simplest is referred to as the *Iterative Disjoint Path* algorithm: it computes k totally disjoint paths by repeatedly finding the shortest path, removing its edges from the graph, and repeating the process until either the source and destination are no longer connected or k paths have been obtained.

The algorithm presented here builds on Iterative DP, but instead seeks *maximally* disjoint paths, and therefore still produces a result even in the presence of bridge topologies, i.e., topologies containing an element that must unavoidably be traversed by both of the sought paths. To this end, rather than removing already-used edges from the graph, the algorithm merely penalizes them, so that the shortest-path algorithm (e.g., Dijkstra's) prefers totally disjoint paths whenever these exist, while still allowing them to be shared when necessary. This behaviour is summarized in Algorithm 1.

It is worth noting that this algorithm, as formulated, seeks k maximally disjoint paths between two fixed nodes s and t, whereas our problem requires two paths originating from *distinct* sources (a and b) and converging to the same destination t. This mismatch is easily resolved by introducing a temporary auxiliary node x, connected to all sources (a and b) by edges of cost 0, and using x in place of s as the single source in Algorithm 1; the resulting paths are then trivially split at x to recover the two source-specific paths P_a and P_b.

4.2.2 Tiered Min-Cost algorithm

Motivation and problem formulation

A recurring strategy for resilient PDC placement is to route the measurements of an essential PMU a and of an associated redundant PMU b – i.e., two PMUs belonging to the same equivalence class $E \in S^*$ (Definition 1) – toward the CC, denoted here as the sink node t, along two paths $P_a \colon a \rightsquigarrow t$ and $P_b \colon b \rightsquigarrow t$ that are as

Algorithm 1 Greedy Edge Penalty

Require: Graph $G = (V, E)$ with edge latencies λ_e and node latencies λ_v, source s, destination t, number of paths k, penalty coefficient $\beta \gg 1$
Ensure: Set of (maximally disjoint) paths $\{P_1, \ldots, P_j\}$ from s to t, with $j \leq k$

$\lambda' \leftarrow \lambda$ ▷ working copies of edge and node latencies
for $i \leftarrow 1$ **to** k **do**
  $P_i \leftarrow \textsc{ShortestPath}(G, \lambda', s, t)$ ▷ e.g., Dijkstra's algorithm
  if P_i not found **then**
    $i \leftarrow i - 1$
    break ▷ s and t are no longer connected
  end if
  for each edge $e \in P_i$ **do**
    $\lambda'_e \leftarrow \beta \cdot \lambda'_e$ ▷ penalize the edges
  end for
  for each intermediate node $v \in P_i \setminus \{s, t\}$ **do**
    $\lambda'_v \leftarrow \beta \cdot \lambda'_v$ ▷ penalize the nodes
  end for
end for
return $\{P_1, \ldots, P_i\}$

independent of one another as possible. Unlike other algorithms (as Iterative DP, or the ones discussed in Section 4.2.4), which impose disjointness as a hard constraint, the **Tiered Min-Cost** algorithm addresses the problem more permissively: if a topology does not admit two fully disjoint paths, the algorithm still returns a pair of paths, selecting the one that minimizes overlap according to a strict, hierarchical order of priorities:

1. minimize the number of shared intermediate **nodes**;
2. subject to the previous point, minimize the number of shared **edges**;
3. subject to both previous points, minimize the combined **real latency** of the two paths.

This is therefore a multi-objective optimization problem with *preemptive* (or *lexicographic*) priorities: a higher-level objective can never be sacrificed to improve a lower-level one, no matter how large the potential gain on the latter.

Formally, let $G = (V, E)$ be the undirected graph modeling the network topology introduced in Section 3.3, in which each edge $e = (u, v) \in E$ carries a transmission latency $\lambda_e = \lambda_{(u,v)} \in \mathbb{R}_{\geq 0}$ and each node $v \in V$ carries a processing latency $\lambda_v \in \mathbb{R}_{\geq 0}$, consistently with the graph model already introduced. Given two source

nodes $a, b \in V$ and a sink node $t \in V$, the algorithm seeks the pair of simple paths (P_a, P_b) that lexicographically minimizes the triple

$$\Big(\underbrace{|V(P_a) \cap V(P_b) \setminus \{a, b\}|}_{\text{shared nodes}} , \underbrace{|E(P_a) \cap E(P_b)|}_{\text{shared edges}} , \underbrace{\ell(P_a) + \ell(P_b)}_{\text{total latency}} \Big).$$

From lexicographic priority to a single cost functional

An optimization problem with lexicographic priorities is not, in general, directly solvable by a scalar-minimizing solver. Tiered Min-Cost employs the classical *penalty method* (or *big-M method*), well known in linear programming and goal programming: a pair of coefficients $M_n \gg M_e \gg 1$ is chosen (`node_penalty` and `edge_penalty`, respectively) such that the scalar functional

$$C(P_a, P_b) \;=\; M_n \cdot |V(P_a) \cap V(P_b) \setminus \{a, b\}| \;+\; M_e \cdot |E(P_a) \cap E(P_b)| \;+\; \Big(\ell(P_a) + \ell(P_b)\Big)$$

induces, over the set of admissible path pairs, the same ordering as the lexicographic comparison defined above. For this to hold, it is necessary and sufficient that no combination of gains at lower priority levels can ever compensate for a loss at a higher one.

> **Lemma 4.2.2.1** Dominance of the edge penalty
>
> Let $T_\lambda = \sum_{e \in E} \lambda_e$ be the sum of the latencies of all edges in the graph. If
>
> $$M_e \;>\; 2\,T_\lambda,$$
>
> then no pair of paths can reduce its cost by sharing an additional edge for the sole purpose of saving latency.

Proof. A simple path uses each edge at most once, so $\ell(P) \le T_\lambda$ for every path P. Consequently, the maximum achievable difference in total latency between any two path pairs (P_a, P_b) and (P'_a, P'_b) is bounded by

$$\Big|(\ell(P_a) + \ell(P_b)) - (\ell(P'_a) + \ell(P'_b))\Big| \;\le\; 2\,T_\lambda.$$

Sharing one additional edge costs exactly M_e more in the functional C. If $M_e > 2T_\lambda$, this cost can never be offset by the maximum theoretically available latency saving, so the optimal solution of C will never share an avoidable edge merely to save latency. □

Lemma 4.2.2.2 Dominance of the node penalty

If

$$M_n \;>\; (|E|+1)\cdot M_e,$$

then no pair of paths can reduce its cost by sharing an additional node for the sole purpose of reducing shared edges and/or latency.

Proof. The number of edges that can be shared between two paths is bounded by $|E|$, so the maximum achievable edge-penalty contribution is $|E| \cdot M_e$. By Lemma 4.2.2.1, the residual contribution due to latency is already dominated by a single M_e. Hence the maximum overall saving achievable jointly at the "edge" and "latency" levels is strictly less than $(|E|+1)\cdot M_e$. If $M_n > (|E|+1)\cdot M_e$, sharing one additional node is never advantageous, whatever the benefit obtained at the other two levels. □

In the implementation, these two lemmas are realized directly in code as

$$M_e = \texttt{edge_penalty} = 2 \cdot \texttt{total_latency} + 1,$$
$$M_n = \texttt{node_penalty} = (|E|+1) \cdot \texttt{edge_penalty} + 1,$$

where `total_latency` is the sum (in integer-scaled units) of the latencies of all edges. Computing M_e and M_n automatically from the graph, rather than receiving them as external parameters, guarantees that the node $\gg$ edge $\gg$ latency hierarchy is *always* respected, regardless of the scale of the input weights.

Reduction to a minimum-cost flow problem

Jointly minimizing $C(P_a, P_b)$ over the *pair* of paths (rather than independently over each) is a non-trivial combinatorial problem, since the overlap term couples the two decision variables. The adopted solution reduces the problem to an instance of **minimum-cost flow** of value 2, exploiting two classical techniques from network flow theory:

- *node splitting*, traceable to Menger's theorem, in order to separately price the first and second traversal of a node;
- the creation, for each node and each edge, of two *parallel, unit-capacity copies* – one "free" and one "penalized" – instead of forbidding reuse altogether, as the strict-disjointness algorithms do.

An auxiliary directed graph H is constructed as follows. For each node $v \in V \backslash \{t\}$, two copies v_{in} and v_{out} are introduced, connected by:

- a "free" edge $v_{\text{in}} \to v_{\text{out}}$, capacity 1, cost 0;
- a "penalized" path $v_{\text{in}} \to d_v \to v_{\text{out}}$ through an auxiliary node d_v, capacity 1 on both edges, cost M_n on the first leg and 0 on the second.

The sink node t is not split, since it is by definition shared by both paths.

For each undirected edge $\{u, v\} \in E$ and each direction of traversal $(x, y) \in \{(u, v), (v, u)\}$ with $x \neq t$, the following are introduced analogously:

- a "free" edge $x_{\text{out}} \to y_{\text{in}}$, capacity 1, cost equal to the real latency $\lambda_{(x,y)}$ (or to t in place of y_{in} if $y = t$);
- a "penalized" path $x_{\text{out}} \to d_{(x,y)} \to y_{\text{in}}$, capacity 1 on both edges, cost $M_e + \lambda_{(x,y)}$ on the first leg and 0 on the second.

It is important to note that the real latency $\lambda_{(x,y)}$ is added *both* on the free copy *and* on the penalized one: physically traversing the edge incurs a propagation cost that must be paid regardless of whether the edge is shared; the penalty M_e is added to this cost, not substituted for it.

Finally, a super-source node s is introduced, with edges $s \to a_{\text{in}}$ and $s \to b_{\text{in}}$ of capacity 1 and cost 0, and flow demands $\text{dem}(s) = -2$, $\text{dem}(t) = +2$, $\text{dem}(v) = 0$ otherwise are imposed.

Theorem 4.2.1 Flow equivalence

Every admissible flow of value 2 on H satisfying the capacities defined above corresponds bijectively (up to zero-flow cycles, excluded from optimality since all costs in H are non-negative) to a pair of paths (P_a, P_b) in G, and the cost of the flow is exactly $C(P_a, P_b)$.

Proof sketch. Since every node and every edge offers exactly two unit-capacity "channels" (one free and one penalized) and the total injected flow is 2, a node or edge can be traversed by both units of flow at most once each: the first through the free channel, the second – necessarily, since the free channel is already saturated – through the penalized one. The cost paid for a second traversal is thus exactly M_n (node) or $M_e + \lambda_e$ (edge) more than for the first, faithfully replicating the definition of C. By the flow decomposition theorem (see, e.g., Ahuja, Magnanti, and Orlin [21]), an integer flow of value 2 free of active cycles decomposes uniquely into two simple $s \rightsquigarrow t$ paths; the absence of positive-flow cycles in the optimal solution is guaranteed by the fact that every edge of H has cost ≥ 0, so routing flow along a cycle would increase the total cost without contributing to satisfying the demand. □

Theorem 4.2.1, together with Lemmas 4.2.2.1 and 4.2.2.2, guarantees that solving the minimum-cost flow problem on H is exactly equivalent to solving the original lexicographic optimization problem – hence the name *Tiered* Min-Cost. Its pseudocode is available as Algorithm 2.

Algorithm 2 Tiered Min-Cost

Require: Graph $G = (V, E)$, distinct nodes $a, b, t \in V$
Ensure: Pair of paths (P_a, P_b) from a and b to t
1: $T_\lambda \leftarrow \sum_{e \in E} \lambda_e$ ▷ sum of latencies, in scaled integer units[2]
2: $M_e \leftarrow 2T_\lambda + 1$
3: $M_n \leftarrow (|E| + 1)M_e + 1$
4: Build H: split every node $v \neq t$ into $(v_{\text{in}}, v_{\text{out}})$ with a free copy (cost 0) and a penalized copy via d_v (cost M_n)
5: **for** each edge $\{u, v\} \in E$ and each direction (x, y) **do**
6: Add free copy $x_{\text{out}} \to y_{\text{in}}$ of cost $\lambda_{(x,y)}$
7: Add penalized copy $x_{\text{out}} \to d_{(x,y)} \to y_{\text{in}}$ of cost $M_e + \lambda_{(x,y)}$
8: **end for**
9: Connect super-source s to a_{in} and b_{in} (cost 0, capacity 1)
10: Set $\text{dem}(s) = -2$, $\text{dem}(t) = +2$
11: $F \leftarrow \text{MinCostFlow}(H)$ ▷ `nx.min_cost_flow`
12: $(Q_1, Q_2) \leftarrow \text{DecomposeUnitFlow}(F, s, t)$ ▷ flow decomposition theorem
13: $(P_a, P_b) \leftarrow$ simplify (Q_1, Q_2) by removing s and the auxiliary nodes, and assign based on the starting node
14: **return** (P_a, P_b)

Computational complexity analysis

The auxiliary graph H has size linear in G: each node generates 3 edges in the splitting structure, and each undirected edge generates 6 directed edges in H (two directions $\times$ three edges per direction: one free, two for the penalized copy). Overall, $|V(H)| = O(|V| + |E|)$ and $|A(H)| = O(|V| + |E|)$ (A for directed arches, instead of E for undirected edges).

The network simplex method (used inside MinCostFlow) does not, in its most general formulation, have a guaranteed strongly polynomial complexity (it can in principle require an exponential number of pivots in the worst case, although this behavior is extremely rare in practice). While strongly polynomial variants do

[2]MinCostFlow is not guaranteed to terminate if weights, capacities, or demands are floating-point numbers: this is due to the presence of exact equalities that conflicts with the rounding errors typical of floating-point arithmetic.

exist, the NetworkX implementation favors practical efficiency over a broad class of real instances rather than a worst-case theoretical guarantee, a behavior entirely adequate for the scale of the electrical network graphs typically handled in this context.

Beyond the worst-case bound discussed above, it is useful to characterize the average-case behaviour of the algorithm. Building H requires a single linear pass over the nodes and edges of G, costing $O(|V| + |E|)$, consistently with the sizes $|V(H)| = O(|V| + |E|)$ and $|A(H)| = O(|V| + |E|)$ already established. Similarly, DecomposeUnitFlow traces the flow greedily from s to t exactly twice (since $k = 2$ is fixed), with each step costing $O(1)$ and each path having length $O(|V(H)|) = O(|V| + |E|)$, for an overall cost of $O(|V| + |E|)$. All the steps surrounding `network_simplex` are therefore linear in $|V| + |E|$, and hence asymptotically negligible compared to the average-case cost of `network_simplex` itself, typically reported as $O(|V(H)| \cdot |A(H)|) = O\Big((|V|+|E|)^2\Big)$. The average-case complexity of the entire Tiered Min-Cost algorithm is therefore dominated by the flow solver, and amounts to $O\Big((|V|+|E|)^2\Big)$, where $|V|$ and $|E|$ refer to the original graph G.

Note: As formulated, Tiered Min-Cost is intrinsically *pairwise*: the $1+1$ capacities per node and per edge, and the flow demand fixed to 2, are sized exactly for two paths. A natural generalization to $k > 2$ paths (analogous to the generalization discussed for Bhandari's algorithm) would require replacing each "free/penalized" pair with a sequence of k unit-capacity copies of increasing cost $0, M_e, 2M_e, \ldots$ (or the equivalent for nodes, with M_n), so as to price the first, second, …, k-th traversal of the same node or edge with increasing cost, while keeping the dominance proof of Lemmas 4.2.2.1 and 4.2.2.2 intact, with the appropriate multiplicative constants.

4.2.3 k-Shortest Candidates algorithm

Rationale

Rather than solving a single combined optimization problem over the pair of paths, as Tiered Min-Cost does, the **k-Shortest Candidates** algorithm follows a simpler, decoupled heuristic: for each of the two PMUs a and b, a pool of k candidate paths toward the CC is generated independently, in increasing order of cost and within the latency constraint; the two pools are then compared exhaustively, and the pair of candidates exhibiting the least overlap is selected. As in Tiered Min-Cost, overlap is penalized more heavily on shared nodes than on shared edges, reflecting the fact that a shared PDC constitutes a more critical single point of failure than a shared communication link alone; unlike Tiered Min-Cost, however, this preference is enforced only as a *weighted* penalty rather than as a strict lexicographic priority.

Candidate generation

For a given PMU p and target node t (the CC), let $\ell(P) = \sum_{e \in P} \lambda_e + \sum_{v \in P} \lambda_v$ denote the cost of a path P, combining edge and node latencies as in the graph model of Section 3.3. The candidate paths for p are obtained by enumerating simple paths from p to t in non-decreasing order of cost, via Yen's algorithm for the k shortest loopless paths [22] (implemented by `nx.shortest_simple_paths`), retaining at most k of them and discarding any path whose cost exceeds the maximum admissible latency $L_{\max}$. Since the paths are enumerated in non-decreasing order of cost, as soon as one candidate exceeds $L_{\max}$, all subsequent ones necessarily do as well, allowing the enumeration to be stopped early rather than exhaustively filtered. The pseudocode for this is available as Algorithm 3.

Algorithm 3 k-Shortest Candidate Paths

Require: Graph $G = (V, E)$, source PMU p, target t, number of candidates k, maximum latency $L_{\max}$
Ensure: List of up to k candidate paths from p to t

```
candidates ← [ ]
for each path P returned by YenKShortestPaths(G, p, t), in non-decreasing order of cost ℓ(P) do
    if |candidates| ≥ k then
        break
    end if
    if ℓ(P) > L_max then
        break                          ▷ all subsequent paths also exceed L_max
    end if
    append P to candidates
end for
return candidates
```

Overlap score

Given two candidate paths P_a and P_b, respectively from a and b to the CC, let $\widehat{P}$ denote the sequence of intermediate nodes of P obtained by removing both the CC and the originating PMU, i.e., the PDCs actually traversed by the path. Let $N_p = 10$ and $E_p = 1$ be, respectively, the node and edge overlap penalties. The *overlap score* of the pair (P_a, P_b) is defined as

$$\text{overlap}(P_a, P_b) \;=\; -N_p \cdot \left|V(\widehat{P}_a) \cap V(\widehat{P}_b)\right| \;- E_p \cdot \left|E(\widehat{P}_a) \cap E(\widehat{P}_b)\right|,$$

where $E(\widehat{P})$ denotes the set of edges traversed by $\widehat{P}$, treated as undirected: an edge of $\widehat{P}_a$ is counted as shared with $\widehat{P}_b$ if it is traversed by $\widehat{P}_b$ in either direction.

A higher (less negative) overlap score therefore corresponds to less overlap, and hence to a more resilient pair of paths; the heavier penalty $N_p \gg E_p$ reflects the higher priority given to avoiding shared nodes over shared edges.

Selecting the best pair

Given the two candidate lists for a and b, the algorithm evaluates the overlap score of every possible pair of candidates and returns the pair achieving the highest (least negative) score, which is formalized in Algorithm 4.

Algorithm 4 Best Candidate Pair

Require: Candidate lists $\text{paths}_a = [P_a^1, \ldots, P_a^{k_a}]$, $\text{paths}_b = [P_b^1, \ldots, P_b^{k_b}]$
Ensure: Pair of paths (P_a^*, P_b^*) minimizing overlap

```
best ← −∞
for i ← 1 to k_a do
    for j ← 1 to k_b do
        s ← overlap(P_a^i, P_b^j)
        if s > best then
            best ← s;   (P_a^*, P_b^*) ← (P_a^i, P_b^j)
        end if
    end for
end for
return (P_a^*, P_b^*)
```

Computational complexity analysis

Generating k candidate paths for a single PMU via Yen's algorithm costs $O(k \cdot |V| \cdot (|E| + |V| \log |V|))$, using Dijkstra's algorithm with a binary heap as the underlying shortest-path routine; this bound is incurred twice, once for a and once for b. Evaluating the overlap score of a single pair of candidates costs $O(|V|^2)$ in the worst case, and since all $k_a \cdot k_b \leq k^2$ pairs are evaluated exhaustively, selecting the best pair costs $O(k^2 \cdot |V|^2)$. The overall complexity of the k-Shortest Candidates algorithm is therefore:

$$O\Big(k \cdot |V| \cdot (|E| + |V| \log |V|) \; + \; k^2 \cdot |V|^2\Big)$$

Considering that in the evaluation

$$k = \frac{|V| - 1}{4} + 1$$

the complexity can be rewritten as:

$$O\Big(|V|^2 \cdot (|E| + |V| \log |V|) \ + \ |V|^4\Big) = O\Big(|V|^4\Big)$$

Obviously, this is the worst case, in which path length grows linearly with $|V|$.

4.2.4 Unsuitable algorithms after evaluation

Before converging on the Greedy Edge Penalty (Section 4.2.1), Tiered Min-Cost (Section 4.2.2), and k-Shortest Candidates (Section 4.2.3) algorithms, several alternative approaches to computing a pair of maximally independent paths P_a, P_b from two PMUs a, b belonging to the same equivalence class $E \in S^*$ to t (the CC) were evaluated and subsequently discarded. They are briefly reviewed here, together with the reason for their exclusion.

Node-disjoint and edge-disjoint paths via max-flow

The first, natural approach was to rely on NetworkX routines to search for disjoint paths, namely the `node_disjoint_paths` and `edge_disjoint_paths`, both based on Menger's theorem. Both PMUs of an equivalence class can be attached to a synthetic super-source connected to a and b with zero-cost edges, reducing the two-source problem to a standard single-source one. While conceptually simple, they return only totally node- (or edge-) disjoint paths whenever they exist, and both routines are entirely unaware of latency: they return *some* set of disjoint paths, with no guarantee of proximity to the shortest ones, nor any mechanism to respect the maximum latency $L_{\max}$. Since latency is one of the core constraint of the problem addressed in this thesis, and they didn't converge with bridge topologies, both approaches were discarded.

Greedy edge-disjoint paths by hard edge removal

A second approach directly implements the naive *Iterative Disjoint Path* algorithm surveyed by Iqbal and Kuipers [17] and already introduced in Section 4.2.1: the shortest path is computed, its edges are removed from the graph, and the process is repeated. Unlike its penalty-based counterpart, this variant enforces strict disjointness by construction. For this reason, also this approach fails outright on bridge topologies: if the only path between a, b and t is forced through a single shared node or edge, removing it after the first path disconnects the second PMU from the CC entirely, rather than returning a best-effort, minimally overlapping alternative. This limitation directly motivated the substitution of hard removal with the edge-penalty mechanism of the Greedy Edge Penalty algorithm.

Suurballe's algorithm

Suurballe's algorithm [23, 24] computes k totally edge-disjoint paths minimizing the *sum* of their costs, using node potentials derived from an initial shortest-path tree to reduce edge costs to non-negative values, reversing the edges of the first path, and iterating the shortest-path computation while cancelling any edges traversed in opposite directions by successive paths (an *interlacing* step). This yields a provably optimal solution to the edge-disjoint, cost-minimizing path pair problem. However, the algorithm shares two limitations with the max-flow-based approaches above: it guarantees only edge-disjointness, with no notion of a stricter, higher-priority node-disjointness objective, and, being an exact-disjointness method, it degrades ungracefully on bridge topologies, returning fewer than k paths rather than the best available overlapping alternative. Both limitations are precisely what motivated the lexicographic, always-solvable formulation of Tiered Min-Cost (Section 4.2.2).

Bhandari's algorithm

Bhandari's algorithm [18], already discussed in Section 2.2.1 as one of the foundational works on disjoint routing, was also implemented and generalized from its original pairwise formulation to return up to k maximally edge-disjoint paths. The generalization follows a successive shortest augmenting path scheme – the same family of techniques underlying minimum-cost flow with unit capacities – in which already-used edges are heavily penalized in their original direction and made available in reverse at negative cost, allowing later iterations to partially undo suboptimal earlier choices via interlacing cancellation, before decomposing the resulting edge set into simple paths. This method can be regarded as a direct precursor of the flow-based construction underlying Tiered Min-Cost (Section 4.2.2). It was nonetheless discarded for the same core reason as Suurballe's algorithm: it only targets edge-disjointness, with no mechanism to additionally prioritize node-disjointness.

Chapter 5

Evaluation

5.1 Evaluation Setup

The evaluation was carried out on a CrownLabs Ubuntu VM with 4 vCPUs and 8 GB of RAM. Fifty seeds were generated, from which an equal number of random graphs were produced for each of the graph sizes considered. Graphs were generated using the functions defined by Bagnara [20], with the following parameters:

- Edge Latency Min: 1 ms
- Edge Latency Max: 10 ms
- Edge Bandwidth: 1000 kbps
- Node Datarate: 100 kbps
- Node Latency: 10 ms
- Max Latency: 500 ms
- CC Max Links: $\lfloor \text{size}/2 \rfloor$
- (Random, Greedy) Splitting: True (for size ≥ 20)
- (RR) Consider Neighbors: True
- (RR) Parent-Child Constraint: False

The sizes reported throughout this chapter refer to the number of PMUs and the number of candidate nodes present in the graph; the total number of nodes in the graph is therefore $2 \cdot \text{size} + 1$ (including the CC). For each graph, the four algorithms selected in the previous chapter were executed, together with two baseline algorithms (Random and Greedy) defined and presented by Bagnara [20].

For each run (i.e., each tuple of seed, size, and algorithm), two types of metrics were collected: *static* metrics, obtained by analyzing only the paths selected by the algorithm, and *dynamic* metrics, obtained by simulating a sequence of faults and measuring how much data continues to reach the CC.

5.2 Static evaluation

The static metrics evaluated are the following:

- **Execution time**: the time taken by the algorithm to produce a solution (complete or incomplete), with a timeout of 120 s.
- **PDC number**: the number of PDCs required by the selected paths.
- **Latency**: the latency of the selected paths, broken down by PMU category (ePMU or rPMU).
- **Jain index**: a measure of load distribution across the nodes of the network (0: fully unbalanced, 1: fully balanced).

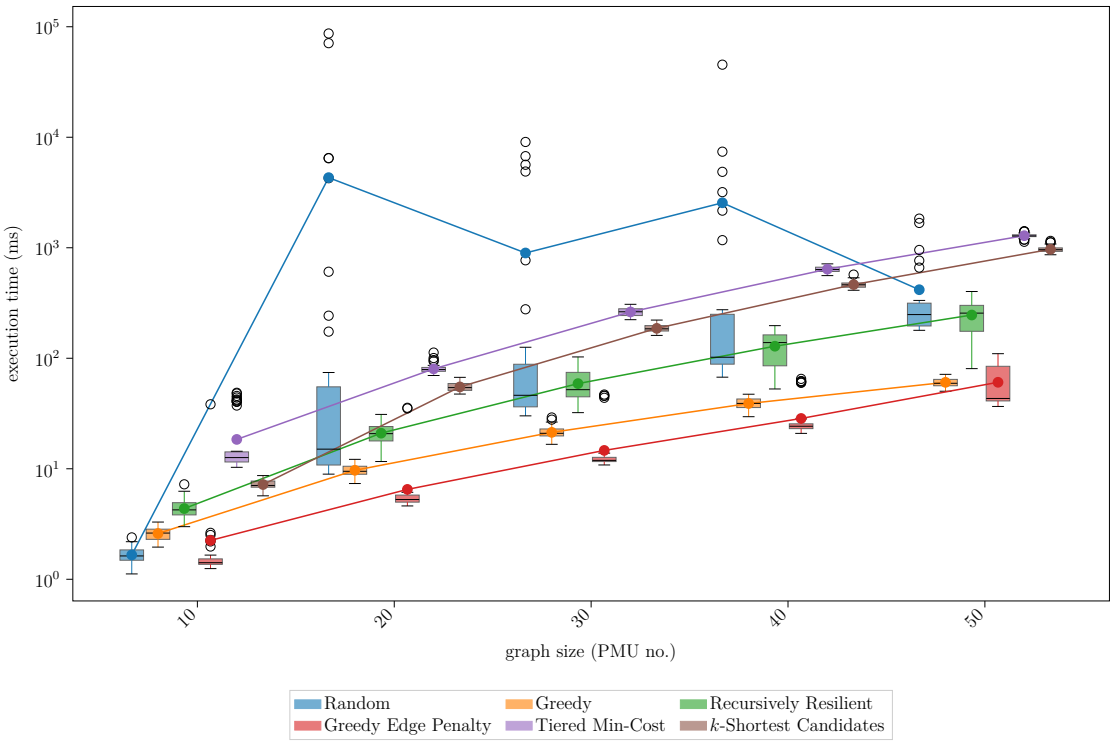


Figure 5.1: Execution time distribution as a function of graph size

Figure 5.1 shows the execution time of the various algorithms as a function of graph size. While Random varies more unpredictably, the four selected algorithms tend to follow a similar curve, scaling linearly with graph size (logirithmically on a logarithmic time axis).

More specifically, the heuristic-based algorithms (Greedy Edge Penalty and Recursively Resilient) tend to exhibit reasonable convergence times (on the order of 100 ms) even at fairly large sizes; the same cannot be said for Tiered Min-Cost and k-Shortest Candidates, which, by their very nature – seeking an optimum over the solution space – scale worse, settling around 1 s for the largest graph size analyzed.

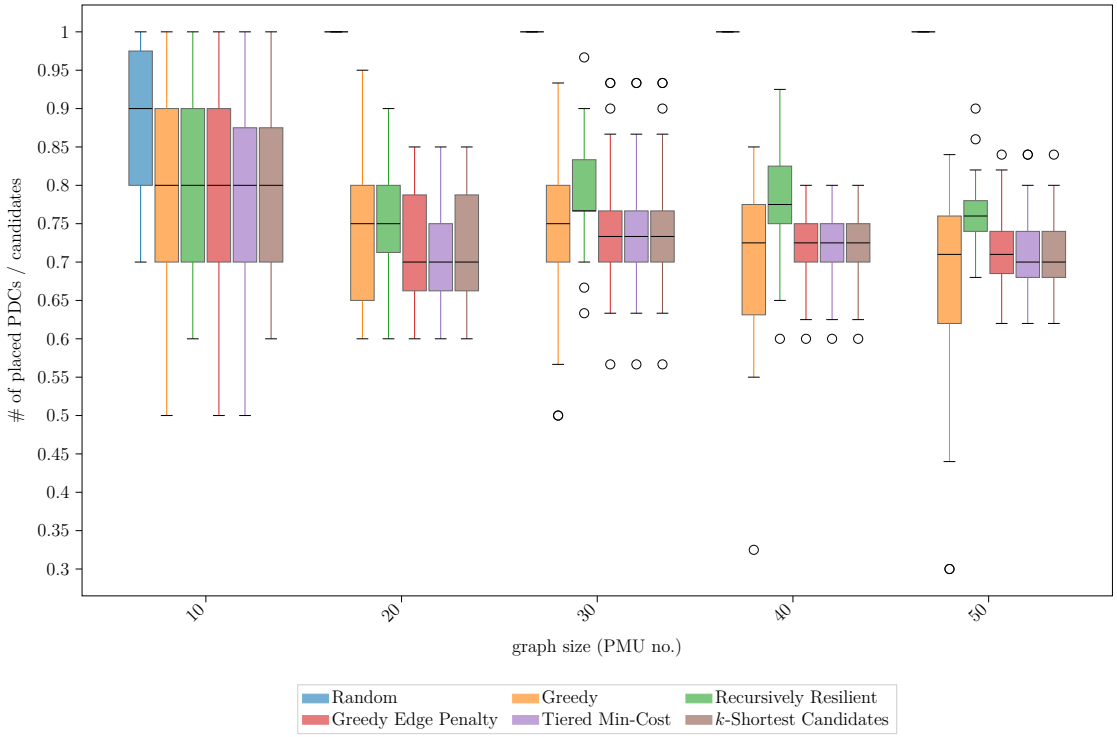


Figure 5.2: Ratio between placed PDCs and candidate locations as a function of graph size

Figure 5.2 shows how many candidate nodes are selected to host a PDC, and hence how much of the network's computational capacity is being used. Here we observe a fairly similar pattern across all algorithms, with the number of placed PDCs settling at around 70% as graph size increases. The only algorithm that deviates upward from this pattern is Recursively Resilient, which – unlike all the others – also attempts to balance traffic across the available nodes, thereby preferring to add a new PDC rather than overloading one that is already in place.

Figures 5.3 and 5.4 show how latency distributions vary as a function of algorithm

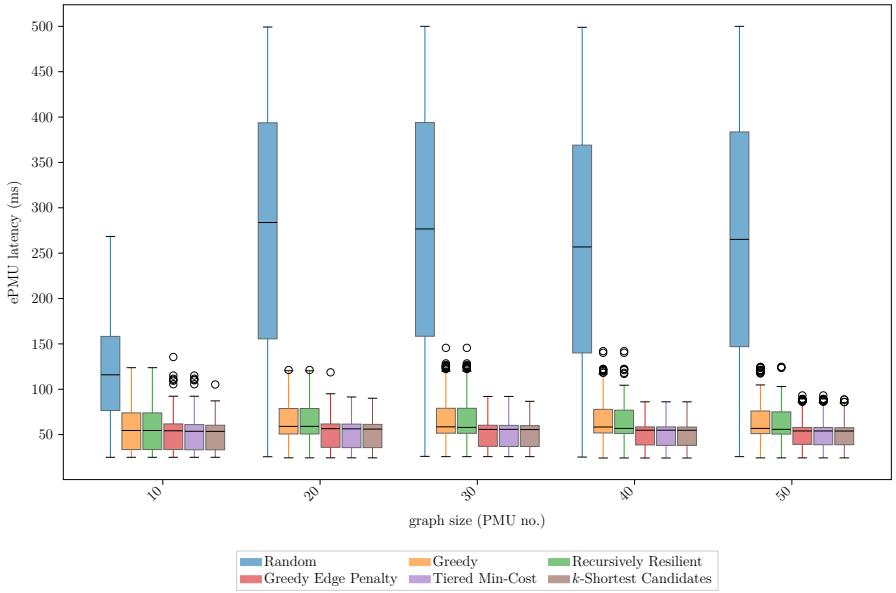


Figure 5.3: ePMU latency distribution as a function of graph size

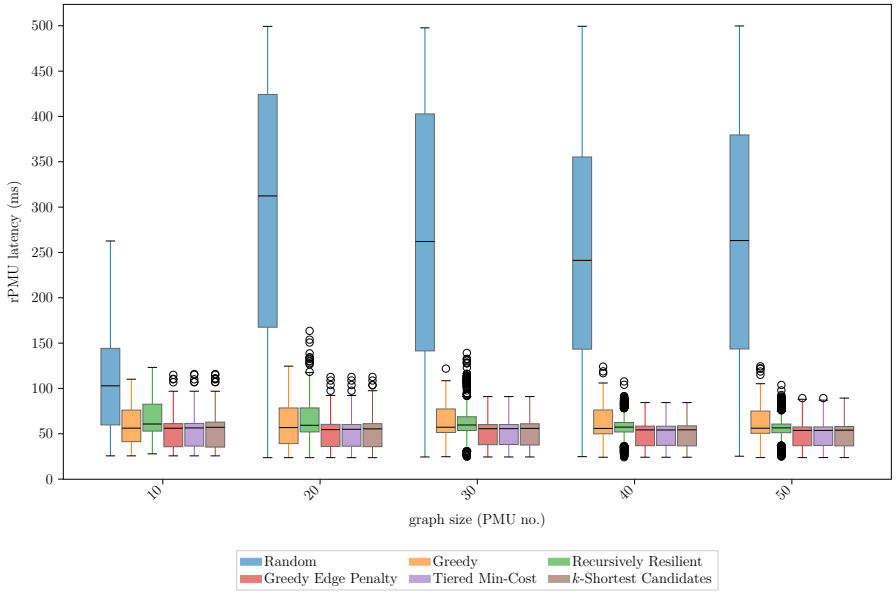


Figure 5.4: rPMU latency distribution as a function of graph size

and graph size. The noteworthy aspect of these plots is the substantial difference between the latency distribution of Recursively Resilient and that of the other algorithms; considering that rPMUs are placed only in a second phase, it is nonetheless remarkable that Recursively Resilient still manages to keep the median

rPMU latency at the same level as the single-step algorithms.

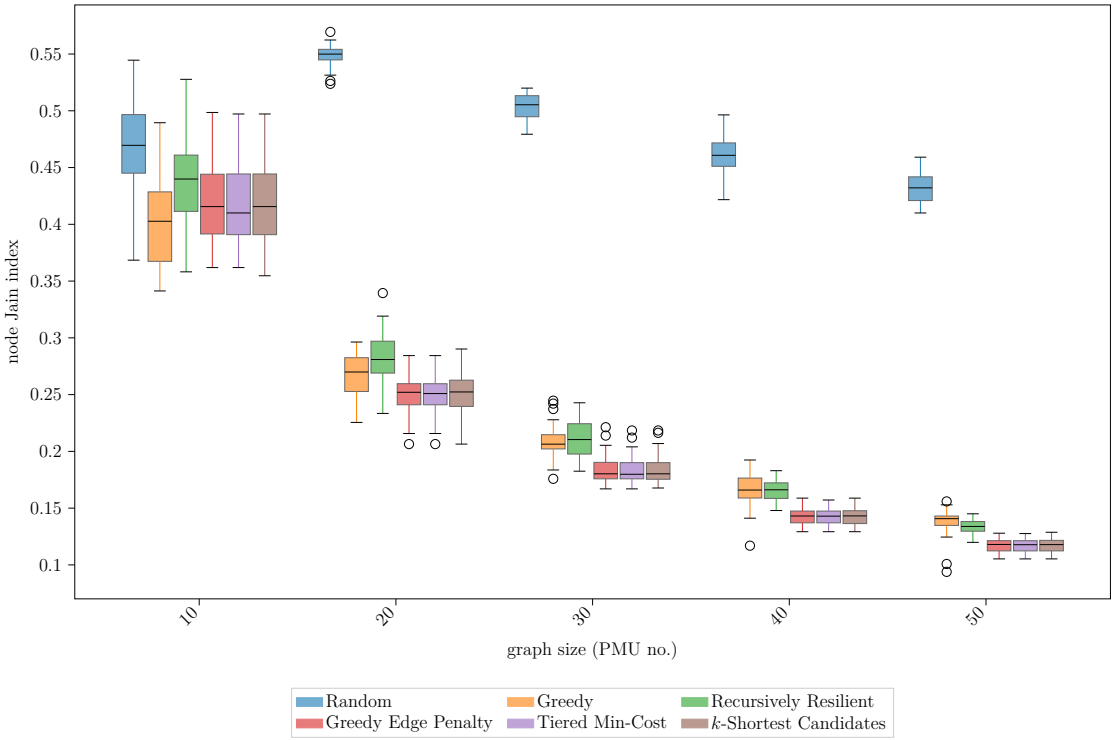


Figure 5.5: Node Jain index as a function of graph size

Lastly, Figure 5.5 shows how node usage (i.e., the number of flows passing through each candidate node) is distributed: a value closer to 0 indicates that a small number of nodes handle most of the traffic, which is the case for all algorithms – except for Random, which, by performing random walks, naturally balances the load – since not every node leads to the CC within a useful latency.

5.3 Dynamic evaluation

The dynamic metrics, instead, are the following:

- **Number of Faults To Lose Observability (NFTLO)**: the number of random faults required before the data of an entire equivalence class fails to reach the CC.
- **Edge Faults To Lose Observability**: the number of edge faults required before losing an equivalence class.

- **Node Faults To Lose Observability**: the number of node faults required before losing an equivalence class.

It should be noted that, in order to obtain these metrics, a sequence of faults was generated for each (size, seed) pair – that is, for each graph – and the same sequence was replayed identically for every algorithm, until an entire equivalence class was lost.

A value of zero for these metrics indicates that the algorithm's result was incomplete and failed to connect at least one equivalence class, so that the network already lacked observability upon termination of the algorithm, before any fault occurred. The lower bound established in the Problem Formulation chapter is therefore 1, represented in the plots below by a dashed red line. Furthermore, since the fault sequence is not infinite, an algorithm that survives the entire sequence resiliently is assigned a preset value equal to the length of the sequence plus one; for this reason, in one of the following plots the values appear capped at a value depending on the graph size.

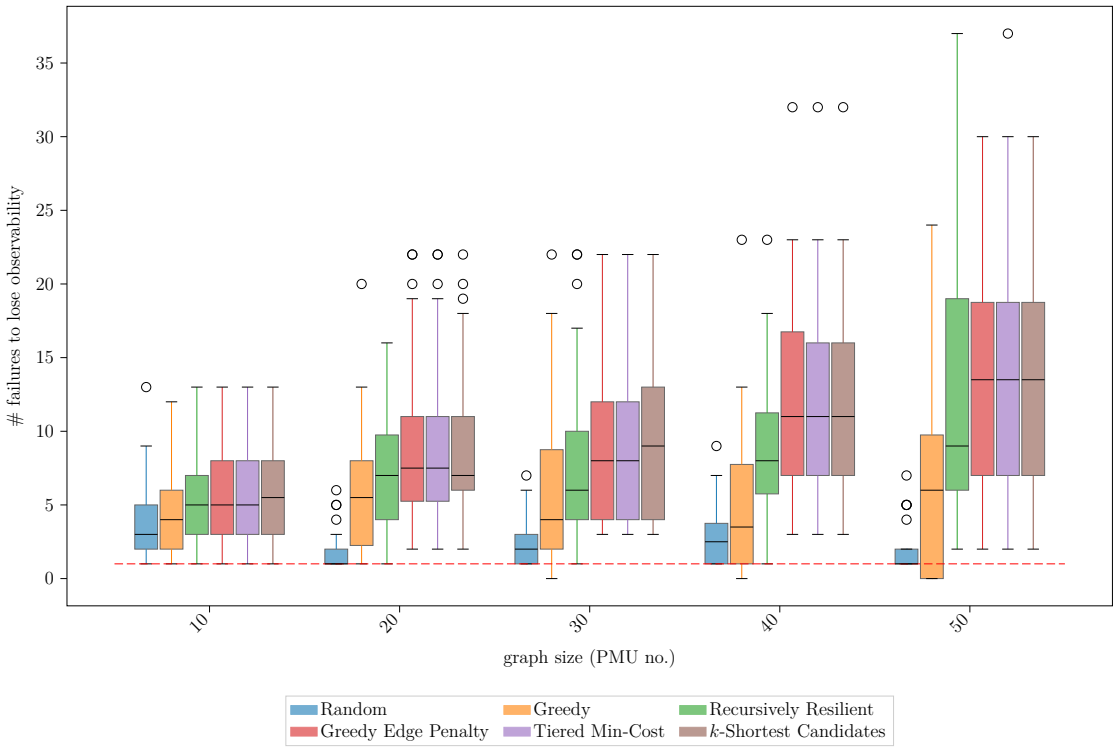


Figure 5.6: Number of failures required to lose observability

Figure 5.6 shows the number of faults required for the CC to lose observability over part of the network. What is immediately apparent is that the baseline

algorithms, which do not account for resilience at all, consistently fall below the four selected algorithms, while the single-step algorithms outperform Recursively Resilient in most cases, likely owing to their more comprehensive, global view of the problem.

A similar pattern can be observed in Figure 5.7, where Random and Greedy again tend to fall below the minimum resilience line, while the presented algorithms manage to be resilient well beyond the required minimum in most cases.

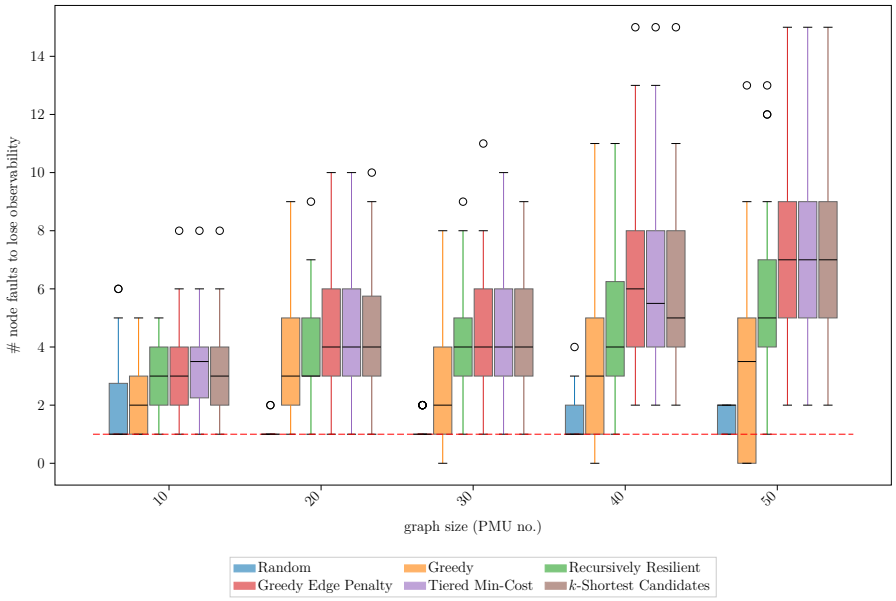


Figure 5.7: Number of node failures required to lose observability

Finally, Figure 5.8 shows that all algorithms, in runs in the range between the first and third quartile, survive most of the edge-failure sequence, which is certainly not the case for the baseline algorithms, further confirming what was observed above for the other two dynamic metrics.

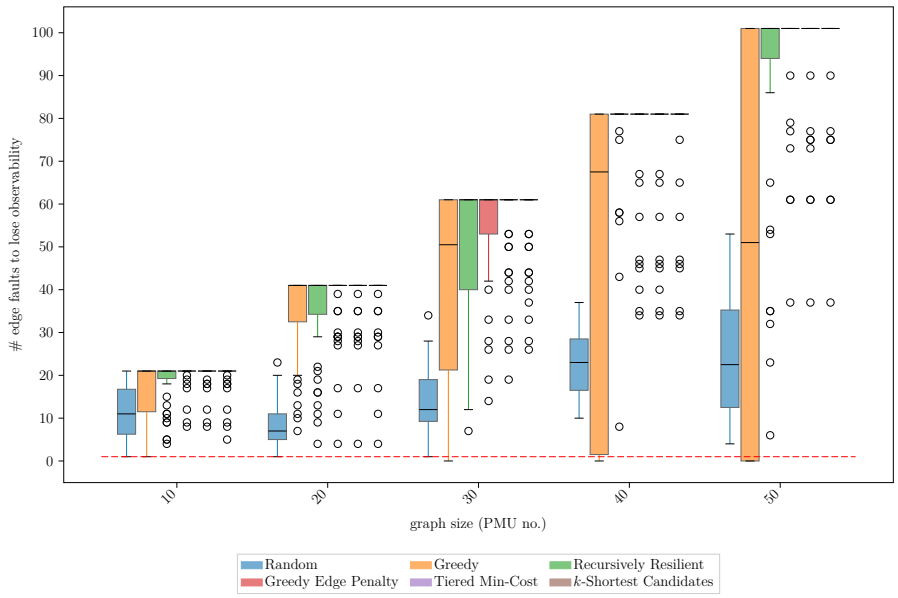


Figure 5.8: Number of edge failures required to lose observability

Chapter 6

Future Enhancements

This chapter outlines several directions along which the work presented in this thesis could be extended.

6.1 Generalizing the ePMU/rPMU Classification via Partial Observability

Throughout this thesis, PMUs have been treated as belonging, in an absolute sense, to one of two categories: essential (ePMU) or redundant (rPMU). This section argues that this classification, while a reasonable simplification for the purposes of the algorithms developed here, does not reflect how observability actually arises in a power system, and outlines how the underlying model could be generalized to capture this more precisely.

6.1.1 Observability from PMU Measurements

Power system state estimation (SE) can be performed starting from different classes of measurements. When only conventional SCADA measurements are available (power flows and injections), the relationship between measurements and the state vector is nonlinear, requiring the Weighted Least Squares (WLS) estimation problem to be solved iteratively, by repeatedly linearizing the system around the current estimate. When PMU measurements are available, on the other hand, the state can be expressed directly in terms of voltage phasors, which are related *linearly* to the PMU measurements themselves; this allows the corresponding WLS problem to be solved in a single step, without iterative linearization [25, 26].

A PMU installed at a given bus measures the voltage phasor at that bus, together with the current phasor on every line incident to it. Since the voltage at a neighboring bus can be recovered from the voltage at the measured bus

and the current flowing on the connecting line via Ohm's law, a single PMU does not only provide direct observability of the bus it is installed on, but also *indirect* observability of every bus directly connected to it. As a consequence, full observability of a power system – meaning that the voltage phasor at every bus is either directly measured or can be uniquely computed from the available measurements – typically requires PMUs to be installed on only 20–30% of the buses, rather than on every bus [27].

Redundant measurements, beyond what is strictly required for observability, further allow the measurement error to be reduced, or bad data to be identified and discarded, when a bus is observed by more than one independent source.

6.1.2 An Illustrative Example: IEEE 14-Bus System

Figure 6.1 shows the IEEE 14-bus standard test network, a widely used benchmark topology representing a small power system. Several placements of only 4 PMUs are sufficient to achieve full observability of this network; achieving *redundant* observability, in the sense that every bus is measured at least twice (directly or indirectly), instead requires 7 PMUs. The placement of the 4 PMUs shown in the figure – at buses 2, 6, 7, and 9 – is one such solution, adopted in this example. It is worth stressing that redundancy, in this sense, is a property of the *buses* of the electrical network, not of the PMUs themselves.

6.1.3 Why the ePMU/rPMU Classification is Only Locally Well-Defined

Everything developed in the previous chapters of this thesis remains valid under this more detailed picture; however, it reveals that the essential/redundant classification of a PMU is, in general, only well-defined *relative to a specific bus*, rather than as an absolute property of the PMU. Consider, for instance, the PMU installed at bus 2 in Figure 6.1: it directly or indirectly observes buses 1, 2, 3, 4, and 5. Similarly, the PMU installed at bus 7 observes buses 4, 7, 8, and 9. From the point of view of bus 4, which is observed by both, one of the two PMUs could naturally be designated as the essential one and the other as its redundant counterpart. The same cannot be said, however, from the point of view of bus 1, which is observed *only* by the PMU at bus 2: here, no redundant source exists. It is therefore not possible to globally classify the PMU at bus 2 as an ePMU and the PMU at bus 7 as an rPMU, since this relationship holds only with respect to the buses they happen to observe in common, and not with respect to every bus each of them observes.

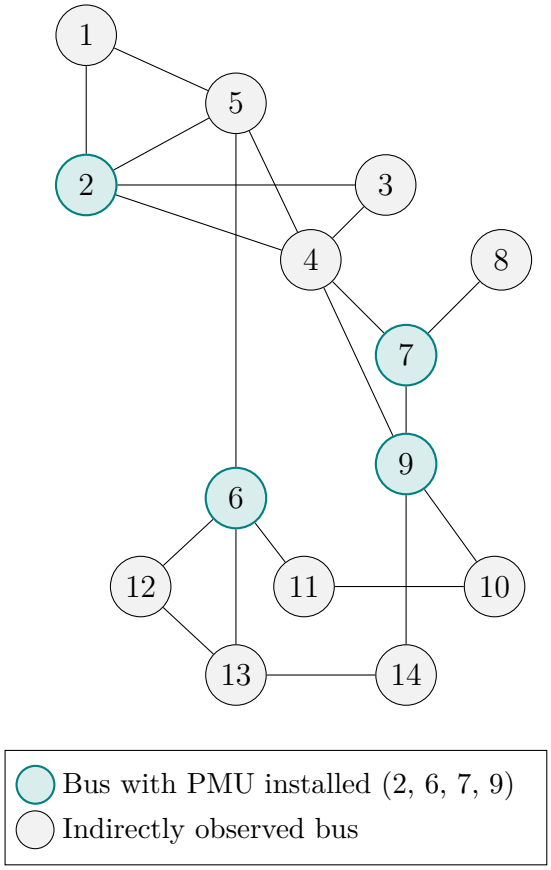


Figure 6.1: The IEEE 14-bus test network, with PMUs installed at buses 2, 6, 7, and 9 (highlighted). Every other bus is observed indirectly, via a directly-measured neighboring bus.

6.1.4 Toward a Per-Bus PMU Decomposition

A natural way to resolve this ambiguity is to stop treating each PMU as a single, monolithic source of information, and instead decompose it into as many logical sub-flows as there are buses it observes. Denoting by $\mathrm{PMU}_{i.j}$ the sub-flow of information that PMU i provides about bus j, the PMU installed at bus 2 in the example above would be decomposed into five sub-flows, $\mathrm{PMU}_{2.1}, \cdots, \mathrm{PMU}_{2.5}$, and similarly for every other PMU. Under this decomposition, $\mathrm{PMU}_{2.4}$ and $\mathrm{PMU}_{7.4}$ – both providing information about bus 4 – can be unambiguously classified as, respectively, an ePMU and an rPMU with respect to bus 4, without this classification interfering with the treatment of the remaining sub-flows of either PMU, which observe different buses and are therefore, in general, not redundant with one another. In terms of the model introduced in this thesis, each sub-flow $\mathrm{PMU}_{i.j}$ would occupy its own equivalence class in the partition S^* of Definition 1, rather than the PMU as a whole.

It is worth noting that, while current PMU technology reports a single, undivided measurement stream per device and does not allow this per-bus decomposition to be extracted at the source, this is a purely technological limitation of existing hardware and firmware, rather than a theoretical obstruction: nothing in principle prevents a future PMU design, or a software layer built on top of an existing one, from tagging and separating its outgoing measurement stream by the bus each component of information pertains to, thereby making this generalized, per-bus notion of essential and redundant PMU directly applicable to the algorithms developed in this thesis.

6.2 Algorithmic and Implementation Optimizations

The Recursively Resilient algorithm, as well as the single-step algorithms presented in Chapter 4, were implemented with a focus on correctness and clarity rather than on runtime efficiency. As a result, several aspects of the current implementation leave room for improvement: data structures used to represent paths and partial solutions could be replaced with more efficient alternatives, redundant recomputation of intermediate quantities (such as the observability coefficient $\rho(n)$) could be avoided through caching or incremental updates, and parts of the codebase could be simplified and consolidated across the different algorithms, which currently duplicate a fair amount of graph-handling logic. None of these changes would alter the resulting placements, but they could improve the practical execution times reported in Chapter 5, particularly for the single-step algorithms, whose execution time was shown to scale less favorably than that of the heuristic ones.

6.3 Waiting-Time Coordination and Stability-Oriented Placement

As discussed in Section 3.2, the resilience-oriented placement problem addressed in this thesis is one of three intents originally identified for intent-based PDC orchestration. The remaining two were left outside the scope of this work and constitute natural directions for future research.

The first, *intent of scalability and availability*, concerns the joint optimization of PDC placement and PDC waiting-time configuration, so as to maximize the availability of phasor data at the final application under a given latency budget, and to support automatic reconfiguration as the number of PMUs grows or as channel latency degrades. Incorporating waiting-time optimization into the resilient placement algorithms developed in this thesis – for instance, by extending the cost function used by Tiered Min-Cost or the observability coefficient used by Recursively Resilient to also account for the waiting time of each candidate PDC – would allow resilience and availability to be pursued jointly, rather than treating waiting-time configuration as an entirely separate concern.

The second, *intent of latency and stability*, extends the resilience objective addressed in this thesis with an additional concern for *stability*: rather than only minimizing end-to-end latency while tolerating a single failure, this intent also seeks to minimize the extent to which the PDC placement must be reconfigured after a failure occurs, so as to reduce the disruption caused by the recovery process itself. Extending the algorithms presented in this thesis to also account for this reconfiguration cost – for example, by penalizing placements that would require moving a large number of existing PDCs or rerouting a large fraction of already-established paths in the event of a failure – represents a natural continuation of the present work, bringing it closer to the original, broader intent it was derived from. It is worth noting that this intent pursues an inherently *reactive* goal – minimizing reconfiguration after a failure has already occurred – which is complementary, rather than alternative, to the *proactive* resilience objective pursued throughout this thesis; combining the two could yield a network that is not only resilient by design, but also capable of recovering from failures exceeding its proactive guarantees with minimal disruption.

6.4 Coupling the Electrical and ICT Grid Layers in Simulation

Throughout this thesis, resilience has been evaluated purely at the level of the ICT communication network: a fault is modeled as the failure of a node or link of the graph G introduced in Section 3.3, without reference to the underlying electrical

grid that the PMUs and PDCs are ultimately monitoring. In reality, however, the ICT infrastructure and the electrical grid it monitors are two coupled layers, and a fault in one can be correlated with, or even caused by, a fault in the other – for instance, the loss of power at a substation may simultaneously disable both a portion of the electrical grid and the ICT equipment co-located with it.

A natural direction for future work is therefore to extend Bagnara's orchestrator and simulator [20], discussed in Section 2.3, to explicitly represent both layers: the electrical grid, on which PMUs observe bus voltages as discussed in Section 6.1, and the ICT network, on which PDCs aggregate and forward PMU data toward the CC. This would require modeling two distinct kinds of dependency between the layers. The first is a *power dependency*: which ICT nodes rely on which electrical buses to remain powered, so that an electrical fault can be propagated to the corresponding ICT equipment. The second, equally important, is a *data dependency*: which bus, or how many buses, each PMU actually observes – directly or indirectly, as discussed in Section 6.1 – so that the loss of a PMU or of a communication path can be evaluated not merely as the loss of one generic data flow, but as the loss of observability over a specific, identifiable set of buses.

Making both couplings explicit would allow fault scenarios to be simulated more realistically, and would make the resilience guarantees provided by the algorithms developed in this thesis verifiable against a simulated environment that reflects actual dependencies between the electrical and ICT infrastructures, rather than against faults injected independently and uniformly across the ICT graph alone.

6.5 Shared Risk Link Groups

Every algorithm presented in this thesis implicitly assumes that if the data of two associated PMUs travels over two distinct edges of the graph G, then the two flows are redundant, in the sense that they cannot be lost simultaneously by a single failure. This assumption, however, does not hold in general: a logical edge between two nodes in G only represents the possibility that those two nodes can communicate directly, and carries no information about its relationship to any other edge in the graph. It is therefore entirely possible for two distinct edges, connecting two different pairs of nodes, to physically share – in whole or in part – the same underlying cable or conduit. This issue is well known in the disjoint-routing literature surveyed by Iqbal and Kuipers [17] under the name of *Shared Risk Link Groups* (SRLGs): groups of links that share a common risk factor, such that a single physical failure can cause every link in the group to fail simultaneously.

This is far from a purely theoretical concern. A real-world instance of exactly this failure mode occurred in Piedmont, Italy, where two supposedly redundant telephone links connecting Turin and Milan were routed, at some point along their

path, over the same bridge (in Chivasso); when that bridge was swept away by a flood in 1994, both links were severed simultaneously, defeating the redundancy they were meant to provide [28].

Among the algorithms presented in this thesis, the Greedy Edge Penalty algorithm (Section 4.2.1) is the only one that can be readily adapted to account for SRLGs: it would suffice to also penalize every other edge belonging to the same shared-risk group as an edge already selected for the primary path, rather than penalizing only the edge actually traversed. The remaining algorithms presented in this thesis do not offer an equally straightforward extension to this setting; addressing SRLG-awareness for these would likely require revisiting the problem from the ground up, for instance by adapting the underlying shortest-path routine (e.g., Dijkstra's algorithm) to penalize or forbid entire shared-risk groups rather than individual edges, in the same spirit as the star-property-based algorithms discussed by Bermond et al. [19]. It should be noted that these algorithms were not further pursued in this work, as they rely on structural assumptions – most notably the star property, whereby all links subject to a given SRLG are required to be incident to a common node – that cannot be assumed to hold in a generic ICT network topology such as the one considered in this thesis.

Chapter 7

Conclusions

This thesis addressed the problem of placing Phasor Data Concentrators (PDCs) within the ICT infrastructure of a smart grid so that the resulting PMU-to-CC communication paths remain resilient to at least a single failure, whether affecting a link, a low-level PDC, or a PMU, anywhere in the network, while respecting a maximum end-to-end latency requirement. Building on and extending the placement algorithms of Bagnara [20] (Section 2.3), this work introduced a classification of PMUs into essential (ePMU) and redundant (rPMU) categories, formalized through the notion of an equivalence-class partition of the sensor set (Definition 1) and the associated observability coefficient $\rho(n)$ (Definition 2), which quantifies how much non-redundant information about the grid state reaches a given node of the network.

7.1 Summary of the Proposed Approach

Two complementary families of algorithms were developed around this model. The *two-step* approach, embodied by the Recursively Resilient algorithm (Section 4.2.2), first places PDCs for the ePMUs by minimizing latency using Bagnara's existing algorithms, and then recursively extends the resulting tree to accommodate the rPMUs, greedily selecting, at each step, the candidate node that maximizes the relative variation $\Delta_r\rho(n)$ of the observability coefficient, while respecting the latency constraint $L_{\max}$. The *single-step* approach instead seeks, for each pair of associated PMUs, two maximally independent paths to the CC directly, via three distinct algorithms of increasing sophistication: Greedy Edge Penalty, which penalizes rather than forbids the reuse of already-selected edges, remaining robust even on bridge topologies; K-Shortest Candidates, which generates a pool of low-cost candidate paths for each PMU and selects the least-overlapping pair among them; and Tiered Min-Cost, which reduces the lexicographic minimization of shared nodes,

shared edges, and total latency to a single minimum-cost flow instance, provably equivalent to the original multi-level optimization problem (Theorem 4.2.1). Several further algorithms – node- and edge-disjoint paths via max-flow, hard-removal greedy disjoint paths, Suurballe's and Bhandari's algorithms – were implemented, evaluated, and ultimately discarded, primarily for their inability to gracefully degrade on bridge topologies or to prioritize node-disjointness over edge-disjointness (Section 4.2.4).

7.2 Summary of the Evaluation

The four selected algorithms, together with Bagnara's Random and Greedy baselines, were evaluated on 50 randomly generated ICT topologies for each of several graph sizes, considering both static metrics (execution time, number of PDCs placed, latency, load balancing via the Jain index) and dynamic metrics obtained by simulating sequences of random node and link failures (Chapter 5). The results confirmed that all four proposed algorithms consistently satisfy the target resilience guarantee – surviving at least one failure without losing observability of any equivalence class – unlike the Random and Greedy baselines, which frequently fall below this minimum. Among the proposed algorithms, a clear trade-off emerged between solution quality and computational cost: the heuristic-driven Recursively Resilient and Greedy Edge Penalty algorithms scale favorably with graph size, remaining close to the Greedy baseline in execution time, while the more globally optimizing Tiered Min-Cost and K-Shortest Candidates algorithms achieve comparable, and at times superior, resilience – particularly against sequences of random faults, as shown by the Number of Faults To Lose Observability metric – at the cost of noticeably higher execution times as the network grows. Recursively Resilient further stood out for its explicit load-balancing behavior, favoring the placement of additional PDCs over overloading existing ones, at the cost of a slightly higher total number of PDCs placed compared to the other algorithms.

7.3 Final Remarks

Taken together, these results show that resilience to a single failure can be incorporated into the PDC placement problem without sacrificing the practicality of the resulting infrastructure, and that different algorithmic strategies – from lightweight, incrementally-applied heuristics to globally optimizing flow-based formulations – offer different, complementary trade-offs between solution quality, computational cost, and ease of extension, allowing the most suitable algorithm to be selected according to the scale and requirements of the specific deployment at hand. At the same time, this work highlighted several open directions along which it could be

extended, discussed in Chapter 6: a more granular, per-bus notion of essential and redundant PMU capturing the fact that observability is not an absolute property of a PMU but one relative to each electrical bus it observes; the integration of waiting-time coordination and reconfiguration-stability objectives, addressing the two other intent-based orchestration use cases identified alongside resilience; purely engineering improvements to the current implementations; a tighter coupling, in simulation, between the electrical and ICT layers of the grid; and explicit support for Shared Risk Link Groups, ensuring that the disjointness guaranteed at the level of the ICT graph translates into genuine physical independence of the underlying communication infrastructure. Addressing these directions would bring the work presented in this thesis closer to a complete, deployable realization of the original intent-based, resilience-aware orchestration vision it stems from.

Bibliography

[1] K. Maheshwari, M. Lim, L. Wang, K. Birman, and R. Van Renesse. «Toward a reliable, secure and fault tolerant smart grid state estimation in the cloud». In: *2013 IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*. 2013 (cit. on p. 4).

[2] F. Luo, J. Zhao, Z. Dong, Y. Chen, Y. Xu, X. Zhang, and K. Wong. «Cloud-based information infrastructure for next-generation power grid: Conception, architecture, and applications». In: *IEEE Transactions on Smart Grid* 7 (2015), pp. 1896–1912 (cit. on p. 4).

[3] D. Anderson, T. Gkountouvas, M. Meng, K. Birman, A. Bose, C. Hauser, E. Litvinov, X. Luo, and Q. Zhang. «GridCloud: Infrastructure for cloud-based wide area monitoring of bulk electric power grids». In: *IEEE Transactions on Smart Grid* 10 (2019), pp. 2170–2179 (cit. on p. 4).

[4] A. Meloni, P. Pegoraro, L. Atzori, A. Benigni, and S. Sulis. «Cloud-based IoT solution for state estimation in smart grids». In: *Computer Networks* 130 (2018), pp. 156–165 (cit. on p. 4).

[5] M. Pau, M. Mirz, J. Dinkelbach, P. Mckeever, F. Ponci, and A. Monti. «A service oriented architecture for the digitalization and automation of distribution grids». In: *IEEE Access*. Vol. 10. 2022, pp. 37050–37063 (cit. on p. 4).

[6] S. Galantino, F. Risso, A. Cazzaniga, F. Garrone, R. Terruggia, and R. Lazzari. «An edge-based architecture for phasor measurements in smart grids». In: *2022 AEIT International Annual Conference*. IEEE, 2022, pp. 1–6 (cit. on p. 4).

[7] H. Lin, C. Chen, J. Wang, J. Qi, D. Jin, Z. T. Kalbarczyk, and R. K. Iyer. «Self-healing attack-resilient PMU network for power system operation». In: *IEEE Transactions on Smart Grid* 9 (2018), pp. 1551–1565 (cit. on p. 5).

[8] Y. Qu, G. Chen, X. Liu, J. Yan, B. Chen, and D. Jin. «Cyber-resilience enhancement of PMU networks using software-defined networking». In: *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2020, pp. 1–7 (cit. on p. 5).

[9] T. Duan and V. Dinavahi. «Fast path recovery for single link failure in SDN-enabled wide area measurement system». In: *IEEE Transactions on Smart Grid* 13 (2022), pp. 1645–1653 (cit. on p. 5).

[10] N. Tzanis, D. Brodimas, K. Plakas, M. K. Birbas, and A. N. Birbas. «Optimal relocation of virtualized PDC in edge-cloud architectures under dynamic latency conditions». In: *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. 2022 (cit. on p. 5).

[11] S. Galantino, J. Marino, F. Risso, S. Braghin, L. Nedoshivina, A. F. Skármeta Gómez, and D. Siracusa. «Edge-to-cloud continuum made real». In: *Computer* 59 (2026), pp. 51–59 (cit. on p. 5).

[12] J. Marino and F. Risso. *Is the computing continuum already here?* 2023. arXiv: 2309.09822 (cit. on p. 5).

[13] Aron Laszka, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. «Resilient observation selection in adversarial settings». In: *2015 54th IEEE Conference on Decision and Control (CDC)*. Dec. 2015, pp. 7416–7421. DOI: 10.1109/CDC.2015.7403391. URL: https://ieeexplore.ieee.org/abstract/document/7403391 (visited on 2026-03-27) (cit. on p. 6).

[14] Md. Zahidul Islam, Shamsun Nahar Edib, Vinod M. Vokkarane, Yuzhang Lin, and Xiaoyuan Fan. «A Scalable PDC Placement Technique for Fast and Resilient Monitoring of Large Power Grids». In: *IEEE Transactions on Control of Network Systems* 10.4 (Dec. 2023), pp. 1770–1782. ISSN: 2325-5870. DOI: 10.1109/TCNS.2023.3240200. URL: https://ieeexplore.ieee.org/document/10027195 (visited on 2026-04-26) (cit. on pp. 6, 12).

[15] Reza Pourramezan, Houshang Karimi, Mario Paolone, and Jean Mahseredjian. «Optimal Coordination of Phasor Data Concentrators in Hierarchical Synchrophasor Networks». In: *2022 IEEE Power & Energy Society General Meeting (PESGM)*. ISSN: 1944-9933. July 2022, pp. 1–1. DOI: 10.1109/PESGM48719.2022.9917141. URL: https://ieeexplore.ieee.org/document/9917141 (visited on 2026-05-10) (cit. on p. 6).

[16] «IEEE Standard for Phasor Data Concentrators for Power Systems». In: *IEEE Std C37.247-2019* (Sept. 2019), pp. 1–44. DOI: 10.1109/IEEESTD.2019.8830511. URL: https://ieeexplore.ieee.org/document/8830511 (visited on 2026-03-27) (cit. on p. 6).

[17] Farabi Iqbal and Fernando A. Kuipers. «Disjoint Paths in Networks». en. In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. Ed. by John G. Webster. 1st ed. Wiley, June 2015, pp. 1–11. ISBN: 978-0-471-34608-1. DOI: 10.1002/047134608X.W8254. URL: https://onlinelibrary.wiley.com/doi/10.1002/047134608X.W8254 (visited on 2026-07-25) (cit. on pp. 7, 23, 32, 47).

[18] R. Bhandari. «Optimal diverse routing in telecommunication fiber networks». In: *Proceedings of INFOCOM '94 Conference on Computer Communications*. June 1994, 1498–1508 vol.3. DOI: 10.1109/INFCOM.1994.337532. URL: https://ieeexplore.ieee.org/document/337532 (visited on 2026-07-24) (cit. on pp. 7, 33).

[19] Jean-Claude Bermond, David Coudert, Gianlorenzo D'Angelo, and Fatima Zahra Moataz. «SRLG-diverse routing with the star property». In: *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*. Mar. 2013, pp. 163–170. URL: https://ieeexplore.ieee.org/document/6529856 (visited on 2026-07-24) (cit. on pp. 7, 48).

[20] Dario Bagnara. *Orchestration of Monitoring Services for the Electrical Grid Sector Through Multi-Cluster and Cloud Technologies*. 2026-03-27 (cit. on pp. 7, 17, 18, 34, 47, 49).

[21] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993 (cit. on p. 27).

[22] Jin Y. Yen. «Finding the K Shortest Loopless Paths in a Network». In: *Management Science* 17.11 (1971), pp. 712–716 (cit. on p. 30).

[23] J. W. Suurballe. «Disjoint paths in a network». In: *Networks* 4.2 (1974), pp. 125–145. DOI: 10.1002/net.3230040204 (cit. on p. 33).

[24] J. W. Suurballe and R. E. Tarjan. «A quick method for finding shortest pairs of disjoint paths». In: *Networks* 14.2 (1984), pp. 325–336. DOI: 10.1002/net.3230140209 (cit. on p. 33).

[25] A. G. Phadke, J. S. Thorp, and K. J. Karimi. «State Estimation with Phasor Measurements». In: *IEEE Power Engineering Review* PER-6.2 (1986), pp. 48–48. DOI: 10.1109/MPER.1986.5528179 (cit. on p. 42).

[26] Arun G. Phadke and James S. Thorp. *Synchronized Phasor Measurements and Their Applications*. Power Electronics and Power Systems. New York, NY: Springer US, 2008. ISBN: 978-0387567976 (cit. on p. 42).

[27] R.F. Nuqui and A.G. Phadke. «Phasor measurement unit placement techniques for complete and incomplete observability». In: *IEEE Transactions on Power Delivery* 20.4 (2005), pp. 2381–2388. DOI: 10.1109/TPWRD.2005.855457 (cit. on p. 43).

[28] Regione Piemonte – Direzione Servizi Tecnici di Prevenzione. *Rapporti sull'evento alluvionale del 4-6 novembre 1994. Prime note e aggiornamenti fascicolati sull'interruzione delle infrastrutture di trasporto e dei sottoservizi di rete.* Torino, Italia: Regione Piemonte, Assessorato Ambiente, Protezione Civile e Tutela del Suolo, Nov. 1994. URL: https://www.regione.piemonte.it/web/temi/protezione-civile-difesa-suolo-opere-pubbliche/calamita-naturali/rapporti-sullevento-alluvionale-4-6-novembre-1994 (cit. on p. 48).