MASTER IN HIGH PERFORMANCE COMPUTING

Representation of distribution networks of ships using graph-theory

Academic Supervisor :
PROF. GIANLUIGI ROZZA (SISSA)
Industrial Supervisor :
ING. MATTEO DREOSSI (CETENA S.p.A.)

Co-advisors(s):
DR. NICOLA DEMO (SISSA)
DR. ANDREA MOLA, Ph.D (SISSA)
DR. MARCO TEZZELE (SISSA)
ING. DARIO CANGELOSI (CETENA S.p.A.)

Candidate:
Dr. Aurora MAURIZIO, Ph.D

4th EDITION
2017–2018

Contents

Abstract

In the present study, we introduce a methodology based on graph theory into a High Performance Computing (HPC) environment developing a screening tool aimed at identifying the most critical units of water and electricity interconnected plants subject to fire and water casualties. The application of the methodology is exemplified via small-scale graphs representing in detail the topology of real-life plants and via large-scale random graphs reflecting only the main properties of real systems for benchmark purposes. Small graphs allow to show the practicality and efficacy of the methodology in representing the plant layout configuration before and after the casualty. Large graphs allow to estimate the improvement in terms of performance resulting from the parallelization of the code. In particular, our goal was to create a digital replica ("twin") of the ship plants for predicting the state of health and the residual functionality of degradable systems and components after a fire or a flooding casualty, thereby improving passengers safety.

This work has been carried out in collaboration with CETENA S.p.A. (Centro per gli Studi di Tecnica Navale), a Fincantieri S.p.A. group society, in the framework of an internship supported by CETENA S.p.A. as one of the sponsors of the MHPC (Master in High Performance Computing).

Chapter 1

Introduction

CETENA S.p.A., SISSA (International School for Advanced Studies) and Lloyd's Register (Class Society) have recently been involved in a challenge aimed at developing smart algorithms capable to evaluate the effect of different failure modes — caused by a fire or a flooding — on the systems of passenger ships in order to improve the design of new passenger ships [1]. Considering that a failure may cause serious accidents both to the vessel and human lives, the goal of this project is to evaluate the best reconfiguration of current ship plants after each casualty scenario so as to guarantee the minimal functioning requirements. This implies a continuous cross check activity (design against installation) that follows the whole ship construction process. The urgency of this work is motivated by the necessity to meet the International Maritime Organizations (IMO) Safety Of Life At Sea (Solas) design prescriptions defined in the Safe Return to Port (SRtP) regulations [2]. According to these criteria, a vessel should be able to safely return to port under its own propulsion after an adverse event not exceeding any of the defined casualty thresholds and criteria imposed by the regulations. Thus, the identification of all the possible failure modes and their propagation through the on-board systems has become a task of paramount importance for the proper design of the ship's systems against failure events.
Currently, in accordance with IMO MSC.1/circ.1369 [3], CETENA produces the Operating Manuals that allow the crew to reconfigure the essential systems after a SRtP casualty so as to be able to bring the ship to a port with adequate comfort and safety standards. However, the ship can be operated in a different way from what is planned in the design stage. In these scenarios, the present static Operational Manuals can be a limitation. In order to be effective during emergency operation, Operational Manuals must be dynamic so as to provide interactive information and guidance to crew members about the reconfiguration of the ship and the recovery of her functions based on the systems configuration at the moment of the

casualty.
The focus of this work is the study of domino effects triggered by fire or flooding casualties in passenger ships in order to provide crew with a tool which speeds up and facilitates the decision-making process when choices have to be made to optimize the ship residual capability after a casualty. The framework of this study may be extended to other types of domino escalation.

1.1 The digital twin

Digital twin models are computerized clones of physical objects such as devices, components, and machines that can be used for in-depth analysis of complex systems [4]. The concept is borrowed from space programs. In space missions where any changes can be fatal, all modifications of a vehicle, probe or rover on a mission, are tested on a detailed simulation model of the system to ensure that changes produce the desired outcome [5]. Now, the digital twin paradygm become also one of the guidlines for Industry 4.0. revolution, to digitalize industrial processs and products. The adoption of digital duplicates (i.e. cyber objects) of real objects helps to infer valuable insights such as points of failure, weak component connections, redundancy and segregation levels of components belonging to systems apparently difficult to analyze. In this way, digital twins can be used for monitoring, diagnostics and prognostics purposes, analyzing the current context of the system and recommending control actions for the physical environment in a dynamic interplay between real and digital objects. In this scenario, computation, information exchange and control features of the physical systems get distributed and physical devices mostly act as data sources for the computation modules [4]. Therefore, an ideal digital twin is the evolution of purely analytical models accessible only to simulation experts into handy decision support tools. These tools should provide real-time decision support to anybody with minimal technical know-how via user-friendly front-ends or apps.

It appears evident how important it is to reduce the computational burden of these operations, considering that ideally the simulations should be performed in real-time over large realistic systems such as the ship's plants object of this work (Figure **??**). Taking inspiration from this philosophy, we tried to translate inputs into analytics systems that support predictive maintenance for critical systems — especially under abnormal operating conditions — so as to predict when and where maintenance is required. The model was designed in order to mediate a fundamental tension between concrete and abstract. That is to be on one hand as concrete as possible in order to be accurate in the predictions and, on the other hand, as abstract as possible for robustness and reusability in different contexts.

Figure 1.1: The digital twin. A digital twin is a digital presentation of a vessel with associated processes and systems, based on continuous data collection (www.marinelog.com).

1.2 Graph-theory application

In order to start building the prototype digital twin, we considered different alternative and suitable strategies among which we can mention fault-trees [6], Petri-nets [7, 8] and finite state automata [9, 10], and we decided to rely on graphs for this feasibility study. Graphs are versatile data structures used in quite a variety of fields to represent different systems including physical systems such as circuits [11], biological networks [12] and social and information systems [13]. The model is so flexible that the components of a ship's plant, modeled through the graph described in this thesis, data structure capable of assuming — at any given time — only one of a finite number of possible states. States that can change from one to another in response to some external stimuli.

For example, in a gene network graph that models the cellular response to external stimuli, the genes may constitute the nodes of the graph while directed edges may be used to reflect the hierarchy of the system in a parent-child fashion [14]. In this way, it is possible to propagate the cascade effect of the extracellular perturbation from upstream receptors to downstream targets in the cellular signaling pathway. Such model makes explicit how the total effect of perturbing a single node in the pathway graph — hence interrupting the propagation of the signal — distributes across the descendant nodes logically related to the parent nodes and therefore how the performance of the whole network lays on the functioning of its individual elements. The importance of an element in a network depends on the charac-

teristics of that element and its position in the network. This assumptions are not specific to gene networks.

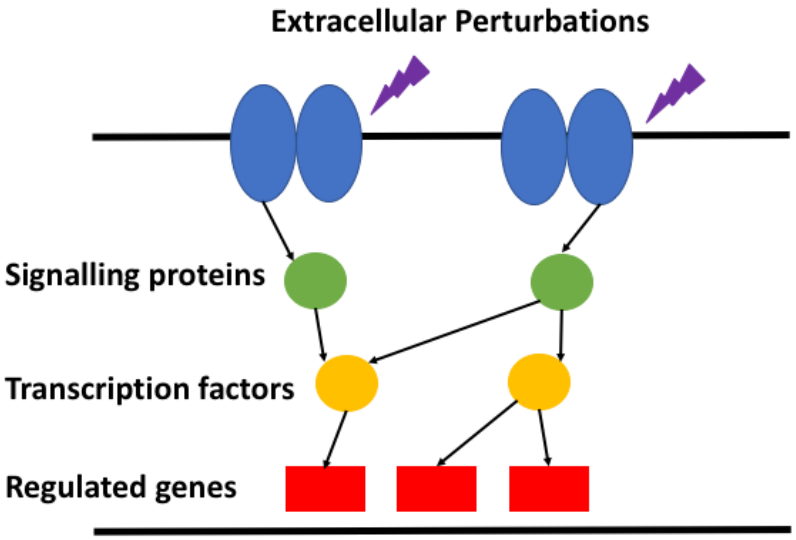


Figure 1.2: Graphs in biology. Graphs can be very helpful to predict the downstream changes in gene expression based on the simulated dynamics of transcription factors in signaling pathways.

Shifting from biology to engineering, the same principles can be applied to a power or a water infrastructure [15, 16, 17]. For example, the network graph of a power plant will be composed by nodes representing generators, distribution stations, cables, and breakers and edges linking such nodes. Similarly, the nodes of the network graph of a water infrastructure will be reservoirs, compressor stations, valves, and pipes.
As for gene networks, the different infrastructures of a ship do not operate in isolation but are mutually dependent. For example, power stations depend on communication elements for control and monitoring, while communication elements depend on power stations for electricity supply. Due to the interdependence between different infrastructures, the functions of infrastructure systems are mutually affected. A failure in one infrastructure often exceedes its boundaries and propagates to other systems and sometimes even back to the original infrastructure, making it more prone to various kinds of disturbance. Once the systems are disturbed by external or internal perturbations, disruptions of components from one system may cause components in the other systems to fail, too. Therefore, it is very important to have a good understanding of interdependences and interdependent system responses under different type of threats. For this reason, even if each particular system can be studied as a separate entity in a very exact and fitting way, a deeper abstraction level taking into account the linkage would result in a better representation of the real situation [18].

1.2.1 Definitions

In this thesis the terms *graph* and *network* will be used interchangeably. A graph G consists of a set of nodes N and a set of edges E that connect the nodes [12]. The nodes are the entities of interest and the edges represent the relationships within the entities. Each edge E connects two nodes N: u, v. Edges can be assigned weights and directions. If the graph is weighted, a *weight* — usually a positive integer value — is associated with every edge in the graph. An edge can be directed or undirected. A directed edge is an edge where an endpoint is designated the *head* and the other the *tail*. A $directedgraph$ or $di-graph$ is a graph where all edges are directed. Considering a node u in a network, the degree of u is defined as the number of edges linked directly to u plus the number of u self loops. In a directed graph, degree measure can be divided in in-degree and out-degree. In-degree is the number of incoming links to a node, or the number of predecessor nodes. Out-degree is the number of out-going links, or the number of successor nodes. When every pair of nodes is joined by an edge the graph is called *complete*. Given a directed graph $G = (N, E)$, its density is defined as $D = |E|/(|N|(|N|-1))$. A complete graph is a graph with the maximal desity (1), while the minimal density of a graph is 0; in this case the graph is called sparse. In a di- graph, a *walk* from node u to node v is an alternating sequence of nodes and edges. The length of a walk when no edge weights are defined is the number of edges traversed. If edge weights are defined, the length will be computed by summing the edge weights. A node v is said to be reachable from node u if there is a walk from u to v. A walk with no repeated nodes is called a *path*. A non-trivial closed path is called a cycle. A finite di-graph with no directed cycles is called a $directedacyclicgraph(DAG)$ (in a DAG it is not possible to start at a vertex v and eventually loop back to v again). The *distance* between two nodes u and v is the length of the shortest walk containing them.

1.2.2 Evaluation of systems vulnerability

The development of a network model is typically the starting point for the evaluation of the system's performance and its vulnerability. Vulnerability can be defined as the capability of fostering either the onset or the escalation of potential domino effects harming the system, with reference to both individual components or an entire plant [19, 20, 21]. This measure could be also regarded as the capability of the network of effectively redistributing the load over its still working part after an accident damaging one or more nodes. Topology-driven analysis of vulnerabilities provides an essential support to the screening analysis of industrial plants, aiming at identifying system connection patterns, shortest connection paths, local and global specifics, etc. For example: systems vulnerability will become higher after a

failure event if the new shortest paths which the load is to be carried along will not be as effective as those in the previous unaffected configuration. Vulnerability can also be evaluated to find out which are the most critical components through parameters such as efficiency and centrality measures proven to be effective in determining the performance of the system with respect to the escalation of failure effects.
Being the present work aimed at evaluating the vulnerability of plants subject to fire/flooding domino effects, the effect of relevant types of active and passive fire protection systems was also implemented in the model in order to obtain a more realistic vulnerability profile for ship's plants.

1.2.3 Shortest paths

The shortest path problem is a fundamental problem with numerous applications. In fact it forms the foundation of an entire class of other measures including efficiency and centrality mesures, essentail to determine the vulnerability of a system.

In graph-theory, a shortest path (Figure 1.3) represents the walk between two vertices that minimizes the sum of the weights of the traversed edges [22].

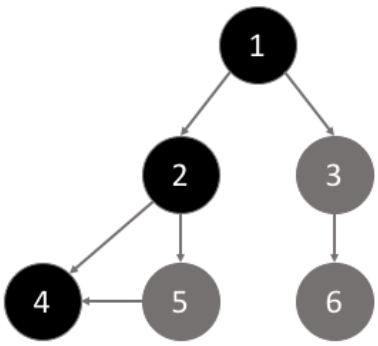


Figure 1.3: Shortest Path. In this unweighted graph $G(N, E)$ the shortest path from source node 1 to target node 4 passes by node 2 and has lenght 2, since 2 edges (of weight 1) are traversed.

Several algorithms are available to calculate a shortest path such as Breadth-First Search (BFS), Dijkstra [23], Bellman Ford [24, 25], and Floyd Warshall [26] .

A BFS from a source node N can be used to compute all the shortest paths from the node

N to all the other nodes in a graph G (Single Source Shortest Path — SSSP problem) in time $O(N+E)$, where N and E are the number of nodes and edges of the graph respectively. If this visit is repeated for all the source nodes of the graph, the cost amounts to $O(N(N+E))$. BFS algorithm works only for unweighted graphs. Unweighted graphs have edge weight equal to 1 while different numerical values can be associated to each edge's weight in weighted graphs. The length or weight of a path is the sum of the weights of its edges.

In sparse positive-weighted graphs, the standard Dijkstras algorithm [23] — a SSSP algorithm for weighted graphs — to calculate the shortest path between two nodes has the asymptotic runtime complexity of $O(E+N\log N)$ if Fibonacci heaps implementation is used [27]. A sparse graph is a graph where E is of order $O(N)$ or below. If instead of Fibonacci heaps, an adjacency list is used to represent the graph and an unordered array is used to implement the queue, the order will be $O(N^2)$; so varying over N source nodes the order will be $O(N^3)$. Dijkstra's algorithm belongs to a class of algorithms known as greedy algorithms [28]. A greedy algorithm makes the decision that seems the most promising at a given time and then never reconsiders that decision.

For dense graphs, more efficient algoritms such as the Floyd Warshall one [26] allow, instead of computing a path from a given start node to all other nodes during the simulation run (BFS, Dijkstra), to compute all shortest paths from each node to all others — All Pairs Shortest Path (APSP) problem — storing them in respective matrices. It must be noted that the APSP problem is a generalization of the SSSP problem and therefore an algorithm which solves the APSP problem will automatically give an answer to a Single Source Shortest Path, while the opposite is not true. Floyd Warshall algorithm $O(N^3)$ is optimized for graphs with positive or negative edge weights (but with no negative cycles) and distributed systems.

The computation of shortest paths however is not limited to the calculation of efficiency and centrality measures. It is strategical, for example, in optimization simulations, where the goal is to efficiently distribute resources between supply and demand points.
Since this operation on large graphs might be problematic in terms of time and memory, parallel computing could lead to a lot of improvements. Giving a better response time to the tool where this algorithm will be used, especially if the path traverses a large quantity of nodes and a lot of alternative paths are present in the network.

1.2.4 Network Efficiency

The network efficiency is a measure of how efficiently it exchanges commodities that flow from one node to the other along different paths in the system [29]. The reciprocal charac-

teristic path lengths of the network are therefore used to estimate it. Since the performance of the network depends on the functioning of its individual elements, efficiency can be measured removing nodes randomly and evaluating the network efficiency loss.

1.2.4.1 Global efficiency

On a global scale, efficiency quantifies the exchange of commodities across the whole network where commodities are concurrently exchanged. Global efficiency is the average of all the reciprocals of the non-zero distances in a network [30, 31].

1.2.4.2 Local efficiency

The local efficiency is a measure of how important a node is, obtained by quantifying how well commodities are exchanged by its neighbors when the node is removed. This measure therefore provides information on the network resistance to failures at the small-scale. It reveals how much the system is fault tolerant [30].

1.2.4.3 Centrality measures

Centrality measures quantify how important a node is within a network. Several different metrics such as degree centrality, closeness centrality, and betweenness centrality exist for measuring nodal centrality.

Closeness centrality

Closeness centrality (Figure 1.4) measures the reciprocal of the average shortest path distance from a node to all other reachable nodes [29, 32]. Thus, the more central a node is, the closer it is to all other nodes. This measure allows to identify good broadcasters, that is key elements in a graph, depicting how closely the nodes are connected with each other. This information can be used in real life to analyze the order in which the resources of a plant need to be arranged and how close each resource needs to be from each other. Closeness centrality is used, for example, to determine the central distribution point in a distribution network.

Betweenness centrality

Betweenness centrality (Figure 1.5) is an index of the relative importance of a node and it is defined by the number of shortest paths that run through it [29, 32].

Nodes with the highest betweenness centrality hold the higher level of control on the information flowing between different nodes in the network, because more information will pass through them. Interestingly, betweenness centrality differs from the other centrality

measures based on the degree of the nodes. In fact, a node can have quite low degree, be connected to others that have low degree, even be a long way from others on average, and still have high betweenness. Considering a node A that lies on a bridge between two groups of nodes within a network, since any path between nodes in different groups must go through this bridge, node A acquires high betweenness even though it is not well connected (e.g. it lies at the periphery of both groups). For example, if a component bridges two systems — lets say a hydraulic and an electric one — damaging this strategically located component/node would result in disconnected systems. Therefore nodes with the highest betweenness are also the ones whose removal from the network will most disrupt communications between other nodes. In a safety optimization context, it is advisable to allocate additional passive safety barriers to nodes with the highest betweenness scores to reduce the vulnerability of a plant to casualty-induced domino effects.

Degree centrality

Degree centrality (Figure 1.6) is a simple centrality measure that counts how many neighbors a node has in an undirected graph [29, 32]. The more neighbors the node has the most important it is, occupying a strategic position that serves as a source or conduit for large volumes of flux transactions with other nodes. A high in-degree centrality means that the node is strongly affected by its neighbors. High values of out-degree centrality represent the influence power of a node, i.e. the higher the value for a particular node, the more nodes are under its control.

It appears evident how calculating centrality measures for all nodes in a graph involves computing the shortest paths between all pairs of nodes in the graph. This is mainly because the selection of high-degree connection nodes may cause larger system structural destruction. Failures in these nodes, in general, result in the heaviest changes in the distribution of shortest paths and therefore in the load of residual available nodes.

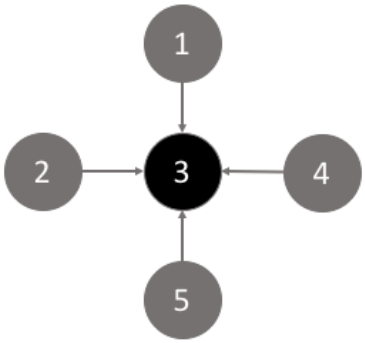


Figure 1.4: Closeness centrality. In this directed graph $G(N, E)$, node 3 is the one with the highest closeness centrality.

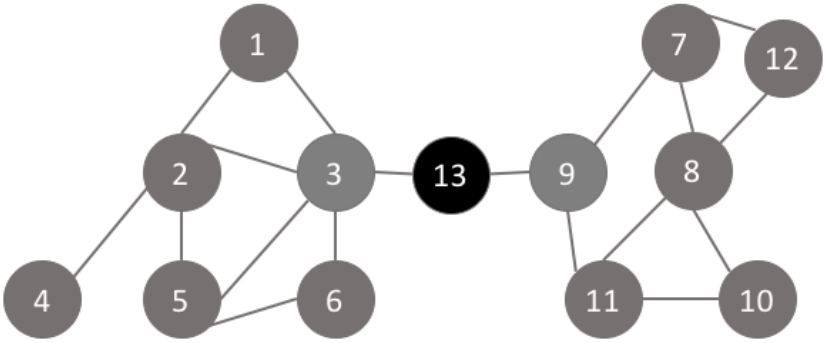


Figure 1.5: Betweenness centrality. In this undirected graph $G(N, E)$, node 13 is the one with the highest betweenness centrality.

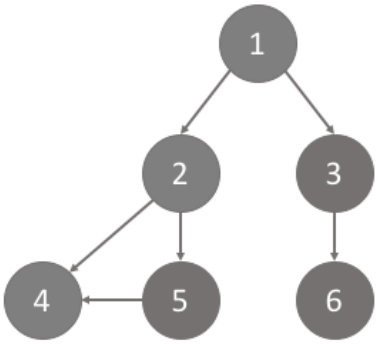


Figure 1.6: Degree centrality. In this directed graph $G(N, E)$, node 4 is the one with the highest in-degree centrality (2) while node 2 is the one with the highest out-degree centrality (2).

Chapter 2

Case study methods: using graphs for representing the distribution networks of ships.

In this section it will be described how graph concepts and HPC can be used to develop a screening tool aimed at identifying the most critical units of water and electricity interconnected plants subject to fire and water casualties.

This work was organized in six distinct stages:

1. **Data collection**
2. **Graph generation**
3. **Integer graph characterization**
4. **Failure simulation and propagation**
5. **Damaged graph characterization**
6. **Parallelization**

2.1 Data Collection

The availability of data is one of the greatest bottlenecks of this activity. Different plants data is retrieved from different sources and databases and shall be hand processed to produce a standardized input. Manual acquisition and elaboration are slow processes that still limit the

potential of the computer simulation. Standardization of data acquisition would certainly result in more agile and adaptable systems' implementations.

Considering this constraint, two approaches have been adopted to perform the study. At first, toy plants representing the topology of real-life plants in detail were used to build the model. With these toy plants the practicality and efficacy of the methodology were proven in representing within a graph different systems layout configurations before and after a casualty. Then, larger plants were simulated with random graphs in order to estimate the improvement in terms of performance resulting from the parallelization of the code.

2.1.1 Toy plants

Small inputs condensing the trickiest features of complex real hydraulic and electric systems for which no general solving strategy was yet available were kindly provided by CETENA. These files were used to build the model.
The input for the graph construction currently consists of text files reflecting the hierarchy of the plant components and their features (component, parent of the component, parent-child relationship, type of component, state of the component, room in which the component is located, component fire resistance, component water resistance, etc.). The hierarchy of the components explains how commodities flow from one component to another component and from one system to another system. In fact, if the input is properly formatted, with this simple digraph model it is possible to represent and integrate different interconnected plants in a unique graph without losing information about their peculiarities. For this reason, correct input formatting is one of the most important steps of the analysis.

2.1.2 Large plants

Large-scale simulated directed unweighted graphs reflecting the properties of real systems (in-degree, out-degree indexes) were generated with *NetworkX* [33] Python package (version 2.0) to benchmark the efficacy of the parallel implementation of the shortest paths computation. Random directed graph (digraphs) can be employed for this task because the computation of the shortest paths between components is not dependent on the properties of the system (e.g. room in which the component is located, type of component etc.)

2.2 Graph generation

A detailed enough set of rules was imposed for the construction of the graph, hence it was necessary to constantly interact with plant technicians during the tool development process [34]. The properly formatted files described above were imported in *NetworkX* [33] to generate the directed graph. In the graph, the nodes represent the plant components (such as generators, cables, breakers, valves, and pipes) while the edges connecting the nodes harbor the logic relations existing between the components (ORPHAN, SINGLE, AND, and OR).

- An **ORPHAN** edge is the edge of a node without predecessors.
- A **SINGLE** edge connects a node to its only one predecessor.
- An **AND** edge indicates that the node/component has more than one predecessor. All the predecessors are necessary for the functioning of that component
- An **OR** edge indicates that the node/component has more than one predecessor. Just one of the node's predecessors should be active to guarantee the functioning of the component

In the text input files each line corresponds to a node/component. The same line reports the name of the predecessor of a particular node/component, the relationship between them, and the list of node's attributes (room in which the component is present, fire resistance, etc.).

In this first scenario, the plants are analyzed under normal operational conditions, without any failures. Source-target (e.g. suction bilge - overboard discharge of a rule bilge) component paths and shortest paths as well as indices such as source-target efficiency, graph global efficiency, local efficiency, betweenness centrality and closeness centrality are computed.

2.3 Failure Propagation

Different scenarios were simulated to investigate the effects of a failure on the topology of the graphs. In particular:

1. Single component damage
2. Fire/water damage in a single room (multiple component damage)
3. Fire/water damage in multiple rooms (multiple component damage)

Cascading failure effects resulting from fire or flooding casualties were simulated in one or multiple rooms at one time, breaking all the fire/water vulnerable nodes present in the room and propagating the damage to the descendents of the nodes in a depth first way until a barrier capable of stopping the propagation was encountered. It must be highlighted that a failure, depending on the considered system, may spread in all directions starting from the damaged component regardless the hierarchy of the components in the commodity supply chain. The way a failure is going to spread from a node to its neighbors must be specified in the input text files. Among the possible barriers to the domino failure effect we can list: breakers to be opened, valves to be closed, active and functional parent nodes in OR relationship with a node that is going to be involved in the failure cascade. In real plants are present passive protection systems, active protection systems, and procedural emergency measures to prevent fire or flooding from spreading. A generic passive protection device is a system or a barrier which does not require either power or external activation to trigger the protective action. A fire resistant cable covered by fireproofing sheath represents a typical example of fire protection device. Although a fire resistant component could, in principle, be affected by a failure propagation cascade, it is assumed that, for the implementation of the fire protection logics in the graph model, a fire resistant component cannot be damaged by a fire outbreak in the room where it is installed. The potential inefficiency of the heat resistant insulator protections and possible degradation phenomena were not cosidered. This is out of the scope of the present study and may be object of further implementations in an optimization framework.

Active protection equipment such as valves or breakers are components that need an external (mechanical, instrumented, or manual) activation to perform their protective effect. For example, a valve can be closed to limit the propagation of a flooding providing effective control of the casualty and preventing its spread to nearby compartments.

Actions on valves and breakers have been modelled in the graph to stop the propagation of the failure as well as to allow the flow of water and electricity through alternative paths when the default ones are not available anymore as a consequence of a specific casualty. This information, reported in the output table are aimed at providing guidelines to the emergency response teams. After the propagation of the failure, the status of the system is reported in two output tables. In the first table is listed the new status of the components (active, not-active) and the rooms in which the components are located (affected, not affected) as well as the new status of valves and breakers that have been operated to stop the propagation of the failure or to open new paths (by-passes etc.) in case of unavailabilty of the default ones. The efficiency and closeness indices are then recalculated. In a second table are reported the new paths, if any, that connect source (supply points) and target components (demand points, or

services to be guaranteed). This second table can be further analyzed to check if the criteria set by the regulations for the proper functioning of that system in degraded conditions are met.

2.4 Parallelization

The code was profiled with the *cProfile* module.
Considering that the most computationally expensive parts of the program involved the calculation of shortest paths, the parallelization efforts were aimed toward this goal. Two different sequential algorithms for the solution of the shortest path problem have been tested including a Single Source Shortest Path (SSSP) algorythm and an All Pairs Shortest Path one (APSP). However, all these algorithms mostly suffer from the following two drawbacks: long running times or huge memory requirements. As the ultimate goal of this project is to provide cruise ship crew with a tool to monitor in real-time the state of the ship plants, these problems can be a limit as fast response times are a prerogative of this kind of application. In order to speed-up the running time of the sequential algorithms without requiring much more memory, the computation of shortest paths was parallelized taking advantage of the Python modules *multiprocessing* and *threading*.

The motivation underlying the choice to use these Python modules relies on the necessity to make a user friendly program which can be, in the testing phase, easily employed by the technical team performing the plants' safety assessments on a daily basis on their workstation. In order to reach the maximum speed possible considering the operator's computer architecture, the program is able to automatically switch from single to parallel mode depending on the size of the graph. Benchmarks to evaluate the scaling of the program before and after the parallelization were performed on Ulysses cluster.

2.4.1 Multithreading

Multithreading allows to submit jobs in shared memory mode, that is creating sub-tasks of a single process which access to the same memory areas. An advantage of this technique is that the communication overhead is minimal. A disadvantage is that this approach can easily lead to conflicts if the synchronization fails when multiple threads try to write to the same memory location simultaneously. Moreover, since threads share the same memory area, it is impossible to use multithreading in a distributed system with multiple nodes. During a parallel task, this becomes a limit to the maximun number of available threads.

2.4.2 Multiprocessing

Relying on distributed memory (multiple processes, independent from each other, are submitted to completely separate memory locations), multiprocessing is a safer approach which nevertheless comes at a price. Overhead, resulting from process communication, will limit the speedup of the application.

Chapter 3

Case study results: using graphs for representing the distribution networks of ships.

With the aim of achieving the highest level of generality and applicability, the software was developed to effectively evaluate plant systems reliability both at single system or subsystems level (e.g standalone power-plant of the electric system) and at combined system level (e.g. simulating the interctions between the hydraulic and electrical systems) without imposing any plant specific changes to the code.

3.1 Main failure propagation scenarios

The plants were analyzed under normal operational conditions and after the propagation of a failure. Two main scenarios, one calling the other, were simulated to investigate the effects of the failure on the topology of the graph and on the residual efficiency of the system. Damage to a single component (and resulting domino effect involving its descendants), and damage to a room (and to all the components present in the room but the shielded ones). The room damage scenario was further divided in two sub-scenarios: fire and flooding.

In the four simplified complex systems described below are exemplified the main rules of the program for component's failure propagation. In expert technicians' opinion, the most common and the most interesting cases of failure propagation can arise in these systems.

3.1.1 The power-plant and distribution system

The power-plant and distribution system (electrical system) was the first system to be considered (Figure 3.1). In this system a failure of one component propagates to its descendents in a downward manner. It means that this system is strongly hyerarchical. This is because no external actions are usually expected to be performed to stop the upward propagation of the failure, since damaged component's upstream breakers switch automatically after the event. This behaviour was modeled via a DAG, a directed graph that has a topological ordering: a sequence of nodes such that every edge is directed from predecessor node to successor node in the sequence of graph nodes. In a power-plant, cascading failure propagation can be stopped by closing a broken component's downstream breaker or by reaching a component that has at least one active predecessor in OR relatioship (a component that is not damaged during the casualty, not directly, nor indirectly during the domino effect). A node that is linked by an AND relationship to its predecessors will not survive a casualty's domino effect unless all its predecessors are alive.

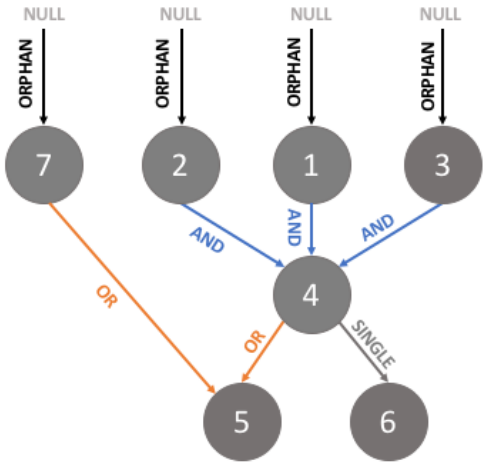


Figure 3.1: The power-plant principles. In this directed graph, representing an electric system, ORPHAN edges are depicted in black, OR edges are depicted in orange, AND edges are depicted in blue, and SINGLE edges in green. Damaging node 1 would result in the damage of nodes 4 and 6. To guarantee the energy supply to node 4 all its three predecessors are needed (1, 2, 3). Node's 6 supply depends in turn on node's 4 availability. On the other hand, damaging node 1 would not affect node 5. This because node 5 has another intact predecessor in OR, node 7.

3.1.2 The rule-bilge system

In the rule-bilge system (hydraulic system) failure may propagate in all possible directions, starting from the damaged component until it reaches a closed valve upstream or downstream. In Figure 3.2 (d), damaging node 3 would result in the breakage of nodes 1, 2, and 5 and in the closure of the node VALVE if it is open. As a consequence, node 4 will survive. This is because actions are always expected to be performed on valves to stop the flooding and to allow new source - target paths after the flood is stemmed. For this reason this system can not be modelled as a topological graph. Parallel, bidirectional edges (forward edge - backward edge) were introduced to allow the correct failure propagation in the rule-bilge system. Parallel bidirectional edges in a directed graph are the equivalent of simple edges in an undirected graph. The choice to model this system as a directed graph (like the power-plant and distribution system) lies in the fact that it allows to link together directed (e.g the electrical systems) and potentially undirected systems (e.g. each hydraulic system). In this way, different systems can be represented within the same graph and the failure can propagate from one system to the other.

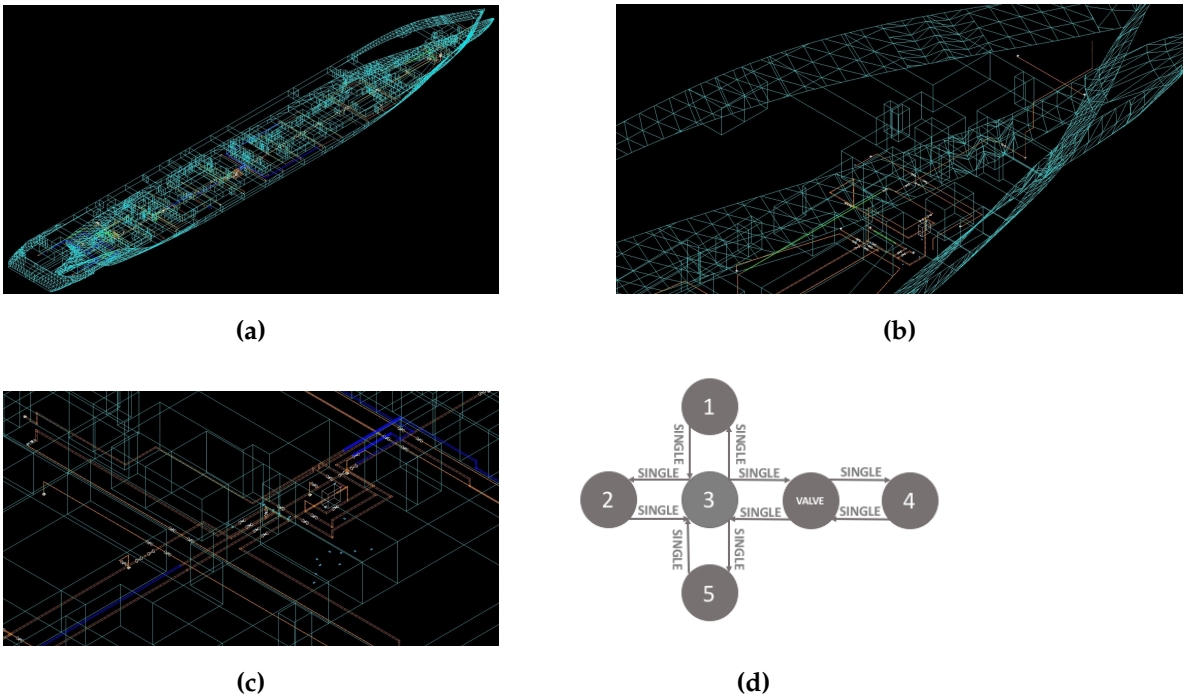


Figure 3.2: The rule-bilge system. In (a), (b) and (c) is represented a CAD model of a passengers' ship rule-bilge system; in (d) a simple bi-directed graph depicts the hydraulic system's failure propagation principles.

3.1.3 Bidirection bus-tie feature for electrical systems

An electrical system equipped with a bus-tie collects in itself all the features of the two previously described systems (Figure 3.3). It is an hybrid system where it is possible to find both directed and parallel, bidirectional edges. Therefore it can be compared to a power-plant system interconnected with a rule-bilge. Modelling in a single graph the interdependence between different infrastructures is a fundamental feature of this program. It allows to represent in a realistic way the different infrastructures of the ship which do not operate in isolation but are mutually dependent.

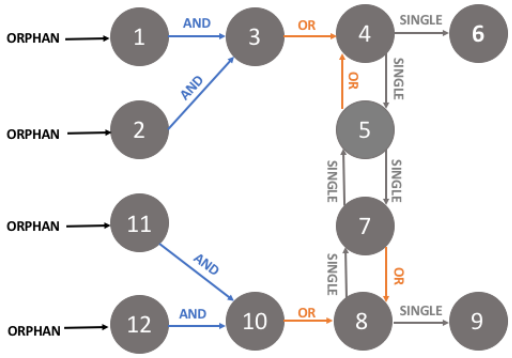
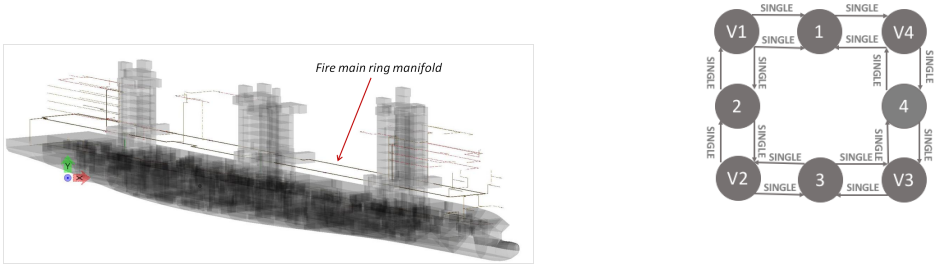


Figure 3.3: The bus-tie feature principles. The bus-tie electrical model extremely simplified in this diagram, contains all the features of the rule-bilge and power-plant systems. Therefore solving this model imposed to generalize the code to different graph models making possible to integrate in a unique graph different interconnected complex systems without losing their inherent characteristics. In the example, a failure in node 1 is going to affect only node 3 which is in AND relationship with nodes 1 and 2. Sice node 4 is in OR relationship also with node 5, it will not be affected by the failure cascade. Nor will be its successor node 6. This is going to happen only if breaker nodes 5 and 7 are both closed, allowing the current produced by generator nodes 11 and 12 to flow through the bus-tie. This plant is an example of a redundant sistem. Breakers installed on the bus tie should be operated to stop the propagation of nodes failure (in the open configuration) and to allow the current to flow through alternate paths (in the closed configuration). Cables, modelled in the real graph, are not represented in this toy diagram.

3.1.4 Fire Main system and Integrated Alarm and Monitoring system

Some on-board systems have the peculiar characteristic of being designed to form a loop path – called ring —- within the entire volume of the vessel. Typical examples of these par-

ticular systems are the Fire Main system (Figure 3.4) (hydraulic system) and the Integrated Alarm and Monitoring system (electrical/automation system). In the former, the ring — fed by at least two main pumps connected to it through large raiser pipes — supplies the numerous fire stations distributed throughout the ship with the water necessary to extinguish possible fires on board. The SRtP regulations require that this hydraulic system continues to operate according to well-defined criteria even in the event of a casualty affecting one of the rooms crossed by the ring manifold and eventually the integrity of the ring itself. In the latter, several automation cabinets — designed to monitor and control various on-board safety systems — are mutually wired up to form a ring network monitored by two different control stations included in the loop. In the event of damage to the ring and consequent loss of continuity, the system must be designed in such a way that all the unaffected automation cabinets continue to be monitored by at least one of the available control stations. The ring, which can be traversed in both directions by water or low-voltage current and is the key feature of both these systems, can be modeled as a cyclic graph. A cyclic graph is a directed graph with at least one cycle. A cycle is a path from a node to itself along directed edges. Cyclic graphs have two issues. The first is that they can not be topologically sorted, the second is that in these graphs a failure may potentially propagate in all possible directions, starting from the damaged component and ending up back at it. For these reasons, the same solving strategies implemented for the rule-bilge have been adopted to model these systems: parallel bidirectional edges (a bi-connected component already represents a subgraph describing a cycle) and depth-first failure propagation. At the same time, valves and breakers shall be capable of efficiently breaking the ring in the event of a failure so as to isolate the damaged parts of the system and operating as barriers to stop the domino failure effect.



(a) **(b)**

Figure 3.4: The Fire Main system. In (a) a CAD model of a passengers' ship Fire Main system is represented; in (b) a simple bi-directed graph depicts the hydraulic system's failure propagation principles. If component/node 1 is damaged valves V1 and V4 should be closed to stop the failure propagation so that components V1, V4, 2, V2, 3, V3, and 4 remain available and efficient.

3.2 Parallelization

3.2.1 Architecture

Two different architectures have been used for the tests performed on the Ulysses cluster.

- A single node with 20 cores, equipped with an Intel®Xeon®CPU E5-2680 v2 @ 2.80GHz processor.
- Two nodes, with 20 cores each, equipped with Intel®Xeon®CPU E5-2680 v2 @ 2.80GHz processor and connected by Intel®infiniband.

Two parallel approaches have been tested to parallelize the computation of shortest paths: multiprocessing and multithreading.

3.2.2 The Parallel Single Source Shortest Path algorithm

Computing shortest paths is one of the fundamental problems in algorithmic graph-theory. The efficiency of the computation is strongly limited by both the size of the graph and the unpredictability of memory accesses. In particular, long memory access time is the dominating factor in the execution time of Single Source Shortest Path (SSSP) algorithms. Current shortest paths algorithms are memory-efficient only on dense graphs whereas most real-world graphs, like the one presented in this section, are sparse. For this reason it is useful to solve the SSSP problem in parallel, taking advantage of HPC.
The first algorithm to be considered for the computation of the shortest path is Breadth First Search (BFS). The complexity of the SSSP based BFS algorithm is of order $O(N + E)$. Given a source node N, the goal of a SSSP algorithm is to find the minimum cost paths from node N to every "target" node in G, where the length of a path is the sum of the weights of its edges. In BFS algorithm all the edges are expected to have the same weight, equal to 1.

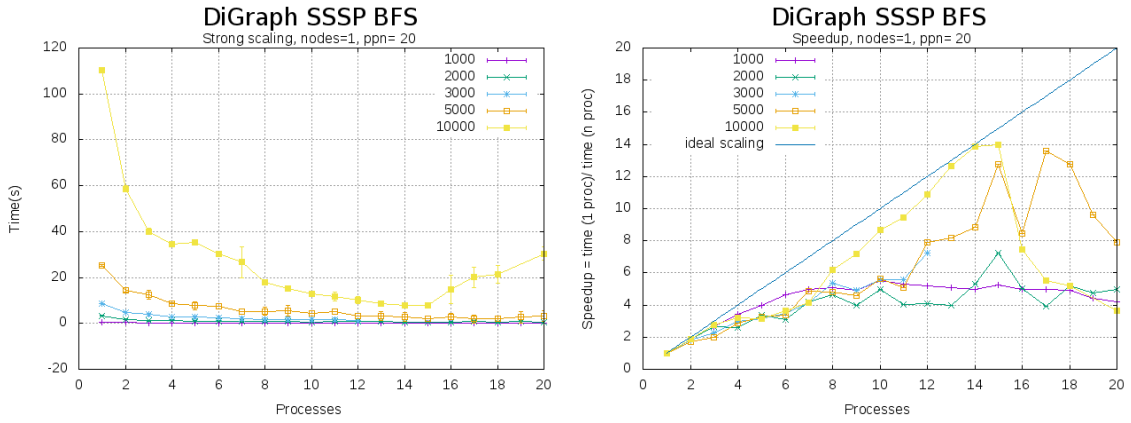
Algorithm 3.1: Parallel APSP BFS.

```
input: graph G, int n_processes
output: dict shortest_path
begin
    processes ← spawnProcesses(n_processes)
    parallel for node ∈ G:
        shortest_path ← BFSFromSource(G, node)
end
```

Shortest paths computed in this way can be stored and used multiple times to compute efficiencies, centrality measures and to check if a commodity (water or electricity) can flow

without barriers from source to target components. In other words to assess the availability of a service. BFS APSP computation was parallelized taking advantage of threading and multiprocessing Python modules simply partitioning the source nodes of the graph in chunks equal to the number of available threads/processes. Each node chunk was assigned to a process together with the full set of target nodes represented by all the nodes in the graph.

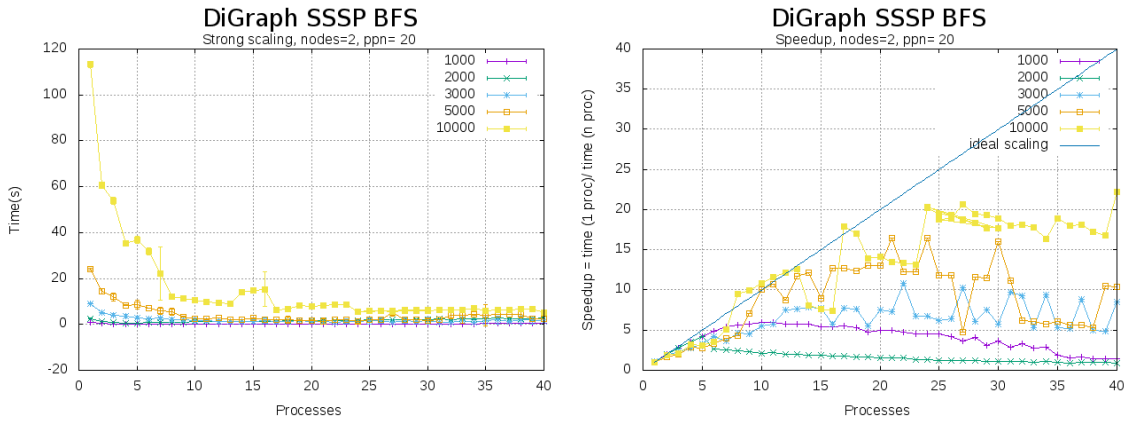
The performance of the parallel version was tested on Ulysses cluster with fixed problem size (1000, 2000, 3000, 5000, 10000 nodes) increasing the number of processes on one node (20 cores) (Figures 3.5, 3.7) and two nodes (40 total cores, 20+20) (Figure 3.6). The tests were performed with graphs of up to 10000 nodes because this is the maximum number of components expected in a passenger ship. Together with strong scaling and speedup, week scaling was computed increasing the problem size together with the number of processes (Figure 3.8).



(a) **(b)**

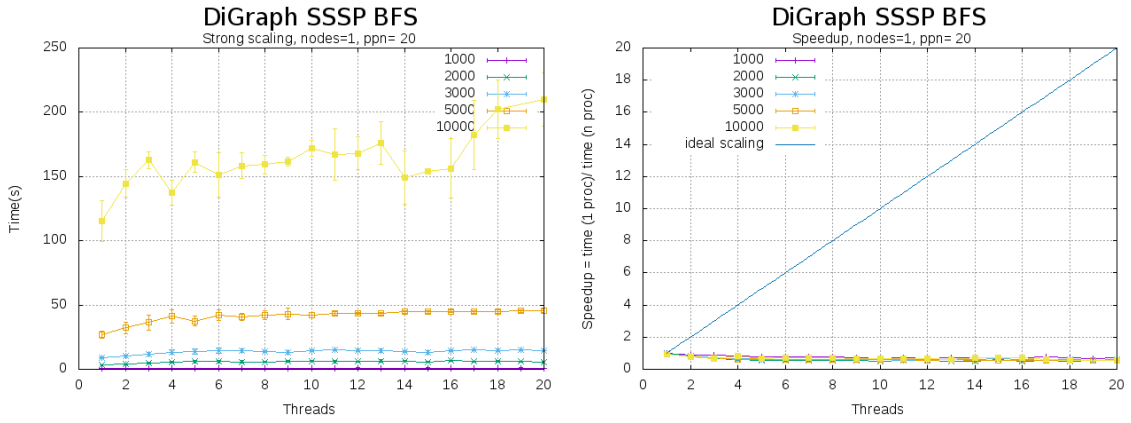
Figure 3.5: BFS SSSP 1node:20ppn. (a) Strong scaling. The different curves correspond to increasing graph sizes (1000, 2000, 3000, 5000, 10000 nodes). In order to measure the strong scaling the problem size (number of nodes in the graph) is fixed while the number of processes is increased. (b) Speedup. Interestingly, whith 15 processes, when the size of the graph reaches 10000 nodes, the speedup (work units completed per unit time) is equal to the number of processing elements used (yellow curve). Then, a pronounced speedup decrese can be observed.

No speedup is appreciable for the parallel threading version (Figures 3.7, 3.8). This is due to the Global Interpreter Lock problem (GIL). The GIL is a lock on the interpreter itself which allows only one thread to be executed at a time in a Python program to obtain safe memory management and prevent deadlocks (e.g. to avoid that data structures shared across threads are inconsistently modified). The reason underlying the necessity of adding locks to all data structures that are shared across threads is that Python uses reference counting for memory



(a) **(b)**

Figure 3.6: Strong scaling BFS SSSP 2nodes:20ppn - multiprocessing. (a) Strong scaling. The different curves correspond to increasing graph sizes (1000, 2000, 3000, 5000, 10000 nodes). The program scales up to 15 nodes. (b) Speedup.



(a) **(b)**

Figure 3.7: Strong scaling and speedup BFS SSSP 1node:20ppn - threading. The different curves in (a) and (b) correspond to increasing graph sizes (1000, 2000, 3000, 5000, 10000 nodes). In order to measure the weak scaling, the problem size (number of nodes in the graph) is fixed while the number of threads is increased. No scaling is observable in this plot. This is to the GIL problem described in this section.

management. It means that objects created in Python have a reference count variable that keeps track of the number of references that point to the object. When this count reaches zero, the memory occupied by the object is released. This reference count variable needs protection from race conditions where two threads increase or decrease its value simultaneously. Race conditions can cause either leaked memory that is never released or, even worse, incorrectly released memory while a reference to that object still exists.

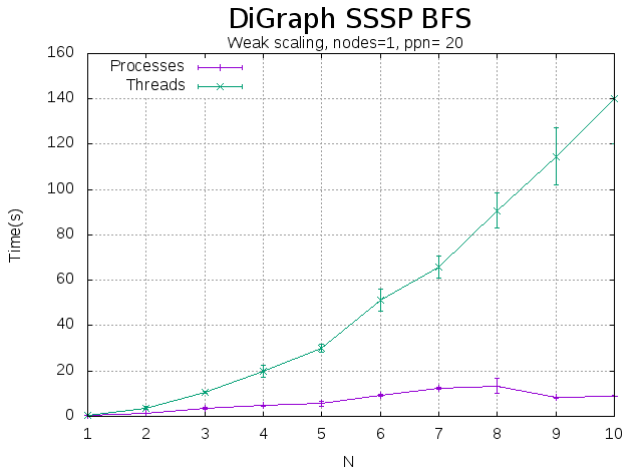


Figure 3.8: Weak scaling BFS SSSP. The initial problem size (number of graph nodes) is 1000 nodes. To measure the weak scaling the problem size assigned to each processing element stays constant and additional elements (processes, threads) are used to solve a larger total problem. No weak scaling is observable when threading module is used, while an almost flat profile is appreciable when the multiprocessing module is chosen instead.

BFS was parallelized simply partitioning the source nodes of the graph in chunks equal to the number of available threads/processes and assigning each chunk to a process together with the full set of target nodes represented by the whole graph. This strategy does not reduce the time needed in computing the SSSP of one node but ideally the time needed in computing the APSP (all shortest paths from each node to all others). One of the possible drawbacks of this approach is the load imbalance [35]. Even the same number of different source nodes was provided to each process, differences of time in computing the shortest paths among processes were expected. The reason is that different source nodes can have a different number of descendendants. For example in a tree like DAG the "root" node would have the maximum number of descendants while the leaves would have none (out-degree 0 and SSSP lenght 0). Therefore, leaf nodes can be safely removed from the job assignment without affecting the shortest path calculation.
In order to avoid a great variation in the computation of the SSSP between different processes, the nodes can not be distributed randomly among them but according to their degree. In general the biggest is the out-degree of a node, the longest is the computation time but exceptions do exist and depend on the node's neighborhood. Nodes with low out-degree whose neighbors have a high out-degree will have a high probability of long computing time. Conversely, nodes with a big out-degree, surrounded by neighbor nodes with a small

out-degree will result in a short computation of the SSSP. Experiments were done on graphs of various size, but no differences in the time spent by the different node chunks assigned to each process to compute the shortest paths was appreciated in our data (Figure 3.9). The graphs were homogeneous and therefore the load was balanced. However, these considerations must be taken into account.

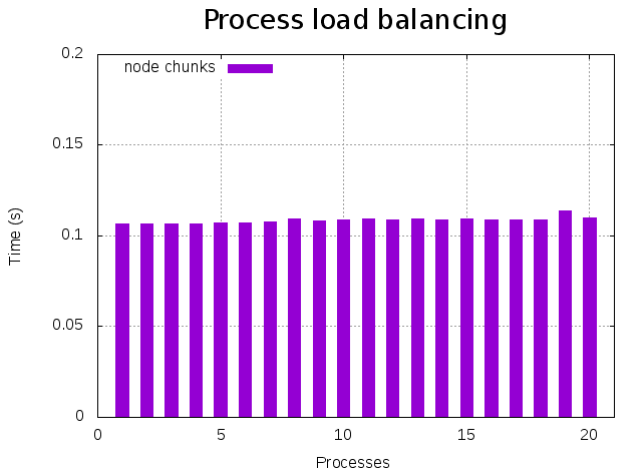


Figure 3.9: Process load balancing. In the plot is represented the time taken by 20 processes to compute the SSSP on a sparse graph of 1000 nodes. Looking at the barplot it is possible to appreciate that the computational load was distributed equally among processes.

3.2.3 The Parallel All Pairs Shortest Path algorithm

In the previous section were described the peculiarities of an algorithm used to compute the SSSP on unweighted, sparse graphs. In this section, is presented an algorithm able to compute the APSP in weighted, dense graphs where most nodes are connected by edges: the Floyd-Warshall algorithm. It does so with $O(N^3)$. For sparse graphs, where N is significantly lower than E^2 better solutions such as Dijkstra's algorithm for weighted graphs or BFS for unweighted graphs can be adopted.

In this analysis Floyd-Warshall algorithm was implemented on unweighted graphs, like BFS algorithm. The reasons of this choice are the following: the first reason is that this is a qualitative analysis on the graph; the second reason is that in this way the performances of this algorithm can be compared with the ones of the previously described one for both dense and sparse graphs (Figures 3.10, 3.11). It must be noted that in its weighted form, Floyd-Warshall algorithm would allow to perform a qualitative analysis of the system, resulting very useful in a system optimization framework. For example, with this algorithm it is possible to detect the optimal routing between a source and a target node. This could be the one with the minimal total weight but also the one with the maximum total weight, or in other words the one that allows the maximum flow between a supply point and a service point.

Floyd-Warshall algorithm was parallelized taking advantage of the multiprocessing Python module.

The parallelization involved two steps. In the first step it was parallelized the update of the adjancency matrices D and P, respectively the adjacency matrix and the distance matrix. If $w(i, j)$ is the weight of the edge between nodes i and j, D is the adjacency matrix in which the weights w of adjacent pairs of vertices are stored, the shortest path between i and j passing through an intermediate node k can be defined with the following recursive formula which is the core of Floyd-Warshall algorythm:

$$D_{ij} = min(D_{ij}; D_{ik} + D_{kj})$$

In the second step it was parallelized the path reconstruction routine, operating on matrix P where intermediate node identifiers are stored.

In the code below, $D.rows$ and $D.columns$ represent the number of nodes, *iter* represents the iterations, while the entries in *row*'s rows and *col*'s columns represent the weight of the shortest paths between nodes. In the classic Floyd-Warshall algorithm, one can easily notice that the nested *row* and *col* for-loops are totally independent and therefore parallelizable.

A $parallel for$ construct applied on the *row* loop yields the first, straightforward paral-

lelization so that at each iteration of the outermost loop *iter* the algorythm is executed on the current block.

Algorithm 3.2: Parallel APSP Floyd-Warshall.

```
input: matrix D, matrix P, int n_processes
output: matrix D, matrix P
begin
    processes ← spawnProcesses(n_processes)
    barrier ← spawnBarrier()
    for iter = 1 to D.rows():
        parallel for row = 1 to D.rows():
            for col = 1 to D.columns():
                if D[row, col] > D[row, iter] + D[iter, col]:
                    D[row, col] ← D[row, iter] + D[iter, col]
                    P[row, col] ← P[iter, col]
        waitBarrier(processes, barrier)
end
```

So, in order to parallelize the computation of the APSP it was performed an horizontal partitioning of the matrix in row chuncks (since in Python matrices are stored in row-major order), in order to distribute the computation among processes. According to this approach, it was possible to update the elements of serveral rows indipendently. In fact, to each process were provided: a shared array representing the distances' matrix (multiprocessing module allows to share arrays allocated from shared memory between processes), a predecessors' matrix and the row chuncks. A barrier was employed for the synchronization.

The reconstruction of the paths for each source-target combination was simply parallelized dividing the source nodes in chunks according to the number of available processes.

These computations involve only a few CPU-bound operations. Therefore most of the time spent for the calculation of the APSPs is determined by pauses during memory access as some tasks need to wait for their turn in order to take place. As expected Floyd-Warshall algorithm scales better than BFS when the goal is to compute APSP in dense graphs (Figures 3.10, 3.11).

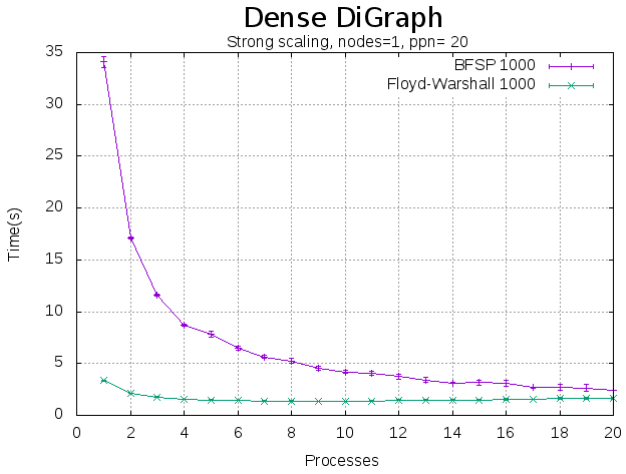


Figure 3.10: Strong scaling APSP in dense graphs, 1000 nodes, 1node:20ppn. As expected the parallel version of Floyd-Warshall algorithm outperforms the parallel version of BFS algorithm when the goal is to compute the shortest path in a dense graph of 1000 nodes (in-degree = 999, out-degree = 999, density = 1).

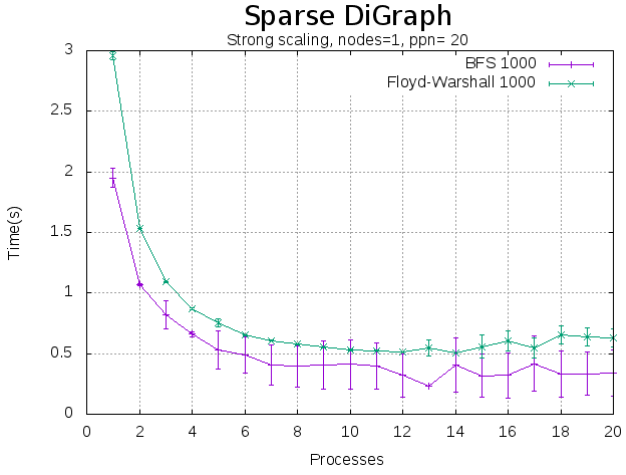


Figure 3.11: Strong scaling APSP in a sparse graph, 1000 nodes, 1node:20ppn. Also in this case, the parallel version of BFS algorithm outperforms the parallel version of Floyd-Warshall algorithm when the goal is to compute the shortest path in a sparse graph of 1000 nodes (in-degree = 2, out-degree = 2, density = 0.002.

Chapter 4

Discussion and Conclusions

The goal of this study was to build from scratch a first virtual but high-fidelity representation of real ship plants in order to test the effect of different casualty scenarios and accelerate the prediction of the best system's reconfiguration strategies to mitigate the damage. In fact, early problem detection and rapid task-planning are fundamental aspects of the risks management process.
In previous sections, the efficacy of the graph metrics was demonstrated in supporting a detailed vulnerability analysis of hydraulic and electric interconnected systems with regard to fire and water-induced domino effects.

The results obtained can be considered from two perspectives.

On one hand, the developed methodology can be employed as a practical tool for a preliminary screening of both the plant residual efficiency after an adverse event and the evaluation of the severities of consequences resulting from the dysfunctionality of a subset of the system.
The results obtained from the graph analysis can be used to predict the best mitigation strategies and improve topology, robustness, and resilience profile of industrial facilities against domino effect propagation. In particular, the components contribution to the cascade effects resulting from fire and water casualties can be evaluated, highlighting the plants major criticalities, vulnerabilities and potential weak points. It should be noted that in the present study, neither the failure probability of the components nor the efficiency of fire protection measures were considered.
However, the applicability of these indications must always take into account economic considerations. On the other hand, it was widely demonstrated how high performance comput-

ing can be beneficial for this kind of analysis. Parallelization was integrated in a program that is user friendly, and relatively fast to run, requiring only a computer terminal and the installation of a few python packages. An intuitive graphical user interface however is still lacking.

To sum up, the results of the case study presented, based on a graph model representing hydraulic and electric plants, have demonstrated that:

- it is possible to represent and integrate in a unique graph different interconnected complex systems without losing their inherent characteristics;
- network analysis is suitable for identifying structural criticalities, e.g. the most connected nodes, shortest path lengths of connection, most vulnerable nodes, etc. ;
- network analysis is suitable for estimating residual functionality of systems after a casualty such as the one related to a fire or flooding event;
- the code scales but broad margins of improvement are still possible.

Therefore, although model refinements are necessary for a more detailed description of the systems, the applicability of the technique was demonstrated as effective.

The promising preliminary results of this feasibility study form the basis for future work aimed at evaluating how and why a component or system can fail, and how it can be designed, repaired and tested to prevent failures from occurring or reoccurring.

In future work, more emphasis must be placed in the fact that real-world systems present unexpected modes of behaviour. This must be taken into account during the extension of the model, when more features will be added and it will become more complex. Considering too many parameters in the design phase can result in model overfitting and consequently in poor model behaviour when new data is presented to the network. On the other hand, chasing the highest model generalization can increase the risk to underestimate uncertainties due to unmodelled disturbances, simplifications, idealisations, linearisations, and so on. If these uncertainties are added to the ones that naturally stem from limitations in measurements, predictions and manufacturing, unforeseen effects can be expected, with the consequence of compromising the particular outcome of an engineering analysis. Therefore a systematic quantification of the uncertainties affecting dynamical systems and the characterization of the uncertainty of their outcomes must be a preliminary step for any engineering design and analysis, where risks must be reduced, safety must be improved, unforseen outcomes must

be predicted, and the revenue must be maximized. Uncertainty Quantification (UQ) analysis may help to provide objective confidence measures of the numerical predictions, especially for non-linear systems which show the most complex behaviors when perturbed [36].

In particular, further research efforts and guidelines may be worthly directed to:

- prepare a standardized input for the graph representing all systems of the ship in a programmatic way;
- relate to each node a frequency of failure to understand how often and why a component or system fails. This can be achieved in two ways. Extracting the information from historical data (property of CETENA S.p.A.) or inferring it from repeated simulations of failure of different nodes;
- predict containment actions through optimization in order to make the system more robust and resilient. This can be obtained through simulations aimed at improving the performances of the system, rethinking the distribution of the components, the arrangement of the systems and the parameters regulating the distribution of the flows. In this context, linear and non-linear differential equations can be employed for the state space representation of the plant [37]. In hydraulic system, for example, coefficients that can be used in this framework are number of pipes, pipe lenght and diameter, water pressure, load balance etc. This kind of analysis would also provide quantitative information about the system;
- apply artificial neural networks to the modelling and fault diagnosis of complex systems [38, 39]. When no sufficiently accurate traditional models are available to solve very complex system's criticalities, neural networks can be a solution. Artificial neural networks exhibit two main benefits: they provide an excellent mathematical tool for dealing with non-linear problems and have self learning ability. A neural network can extract the system features from historical training data using a learning algorithm requiring little or no a priori knowledge about the process. Despite the set of data can never be complete, this limitation can be overcome thanks to the generalization properties of the neural classifier. This provides the modelling of non-linear systems with a great flexibility. In fact, training data must be sufficiently large to satisfy the maximum number of contraints, but the number of used parameters should be small enough to let the classifier achieve the better generalization performances [40]. Once this classifier has been properly trained and validated it can be adopted for realtime fault detection in a digital twin.

In particular, deep learning can be employed to estimate and validate the state of degradation of a machinery learning from the topological characteristics extracted from complex, constantly changing time-series data collected by sensors equipped on the machiney's components. This technology, attempting to generalize structured deep neural models to non-Euclidean domains such as graphs, is called geometric deep learning [41].

Bibliography

[1] D. Cangelosi, A. Bonvicini, M. Nardo, A. Mola, A. Marchese, M. Tezzele, and G. Rozza, "Srtp 2.0 the evolution of the safe return to port concept," in *Technology and Science for the Ships of the Future: Proceedings of NAV 2018: 19th International Conference on Ship & Maritime Research*, pp. 665 – 672, IOS Press, 2018. 3

[2] SOLAS and IMO, "International convention for the safety of life at sea. london," *International Maritime Organization*, 2006. 3

[3] IMO and MSC.1, "Circ. 1369 interim explanatory notes for the assessment of passenger ship systems capabilities after a fire or flooding casualty," 2010. 3

[4] A. Banerjee, R. Dalal, S. Mittal, K. P. Joshi, *et al.*, "Generating digital twin models using knowledge graphs for industrial production lines," in *Workshop on Industrial Knowledge Graphs, co-located with the 9th International ACM Web Science Conference 2017*, 2017. 4

[5] P. Goossens, "Industry 4.0 and the power of the digital twin, adopt a systems approach to machine design and survive the next industrial revolution," *Retrieved*, vol. 5, no. 3, 2017. 4

[6] R. K. Wood, K. G. Stephens, and B. O. Barker, "Fault tree analysis: An emerging methodology for instructional science," *Instructional Science*, vol. 8, no. 1, pp. 1–22, 1979. 5

[7] E. Simon, J. Oyekan, W. Hutabarat, A. Tiwari, and C. Turner, "Adapting Petri nets to des: stochastic modelling of manufacturing systems," 2018. 5

[8] A. Camurri and A. Coglio, "A Petri net based architecture for plant simulation," in *Emerging Technologies and Factory Automation Proceedings, 1997. ETFA'97., 1997 6th International Conference on*, pp. 397–402, IEEE, 1997. 5

[9] X. Chen and J. Jiao, "A fault propagation modeling method based on a finite state machine," in *Reliability and Maintainability Symposium (RAMS), 2017 Annual*, pp. 1–7, IEEE, 2017. 5

[10] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of theoretical biology*, vol. 22, no. 3, pp. 437–467, 1969. 5

[11] F. Dörfler, J. W. Simpson-Porco, and F. Bullo, "Electrical networks and algebraic graph theory: Models, properties, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 977–1005, 2018. 5

[12] W. Huber, V. J. Carey, L. Long, S. Falcon, and R. Gentleman, "Graphs in molecular biology," *BMC bioinformatics*, vol. 8, no. 6, p. S8, 2007. 5, 7

[13] E. Otte and R. Rousseau, "Social network analysis: a powerful strategy, also for the information sciences," *Journal of information Science*, vol. 28, no. 6, pp. 441–453, 2002. 5

[14] S. Klamt, J. Saez-Rodriguez, J. A. Lindquist, L. Simeoni, and E. D. Gilles, "A methodology for the structural and functional analysis of signaling and regulatory networks," *BMC bioinformatics*, vol. 7, no. 1, p. 56, 2006. 5

[15] N. Khakzad, G. Landucci, and G. Reniers, "Application of graph theory to costeffective fire protection of chemical plants during domino effects," *Risk analysis*, vol. 37, no. 9, pp. 1652–1667, 2017. 6

[16] A. Herrera Fernandez, E. Abraham, and I. Stoianov, "A graph theoretic framework for assessing the resilience of sectorised water distribution networks," 6

[17] D. Meijer, M. van Bijnen, J. Langeveld, H. Korving, J. Post, and F. Clemens, "Identifying critical elements in sewer networks using graph theory," *Water*, vol. 10, no. 2, p. 136, 2018. 6

[18] S. M. Rinaldi, J. P. Peerenboom, and T. K. Kelly, "Identifying, understanding, and analyzing critical infrastructure interdependencies," *IEEE Control Systems*, vol. 21, no. 6, pp. 11–25, 2001. 6

[19] Y. Y. Haimes, "On the definition of vulnerabilities in measuring risks to infrastructures," *Risk Analysis: An International Journal*, vol. 26, no. 2, pp. 293–296, 2006. 7

[20] H. Tong, B. A. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. H. Chau, "On the vulnerability of large graphs," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 1091–1096, IEEE, 2010. 7

[21] S. Wang, L. Hong, and X. Chen, "Vulnerability analysis of interdependent infrastructure systems: A methodological framework," *Physica A: Statistical Mechanics and its applications*, vol. 391, no. 11, pp. 3323–3335, 2012. 7

[22] H. Selim and J. Zhan, "Towards shortest path identification on large networks," *Journal of Big Data*, vol. 3, no. 1, p. 10, 2016. 8

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. 8, 9

[24] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958. 8

[25] L. R. Ford Jr, "Network flow theory," tech. rep., RAND CORP SANTA MONICA CA, 1956. 8

[26] Floyd and R. Warshall, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962. 8, 9

[27] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987. 9

[28] V. N. Temlyakov, "Greedy approximation," *Acta Numerica*, vol. 17, pp. 235–409, 2008. 9

[29] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977. 9, 10, 11

[30] V. Latora and M. Marchiori, "Efficient behavior of small-world networks," *Physical review letters*, vol. 87, no. 19, p. 198701, 2001. 10

[31] B. Ek, C. Ver-Schneider, and D. A. Narayan, "Global efficiency of graphs," *AKCE International Journal of Graphs and Combinatorics*, vol. 12, no. 1, pp. 1–13, 2015. 10

[32] I. Kivimäki, B. Lebichot, J. Saramäki, and M. Saerens, "Two betweenness centrality measures based on randomized shortest paths," *Scientific reports*, vol. 6, p. 19668, 2016. 10, 11

[33] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. 16, 17

[34] E. Chow and A. Willsky, "Analytical redundancy and the design of robust failure detection systems," *IEEE Transactions on automatic control*, vol. 29, no. 7, pp. 603–614, 1984. 17

[35] G. Mao and N. Zhang, "Analysis of average shortest-path length of scale-free network," *Journal of Applied Mathematics*, vol. 2013, 2013. 29

[36] H. Agarwal, J. E. Renaud, E. L. Preston, and D. Padmanabhan, "Uncertainty quantification using evidence theory in multidisciplinary design optimization," *Reliability Engineering & System Safety*, vol. 85, no. 1-3, pp. 281–294, 2004. 37

[37] M. Vaezi, A. Izadian, and M. Deldar, "A model linearization technique for hydraulic wind power systems," in *Power and Energy Conference at Illinois (PECI), 2014*, pp. 1–5, IEEE, 2014. 37

[38] K. Patan, *Artificial neural networks for the modelling and fault diagnosis of technical processes*. Springer, 2008. 37

[39] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013. 37

[40] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," in *Measures of complexity*, pp. 11–30, Springer, 2015. 37

[41] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017. 38