Time-Dependent Bipartite Social Networks (TDBSNs) in Naval Ship Design

Enhancing Process Efficiency and Reducing Rework

Georgios Anagnostopoulos

Department of Naval Architecture, Ocean &

Marine Engineering

Supervisor: Dr Panagiotis Kaklis

Glasgow 2025

[This page is intentionally left blank]

ACKNOWLEDGEMENTS

This PhD thesis would not have been possible without the support and guidance of many individuals and organizations, and I am deeply grateful to them all.

First and foremost, I would like to express my heartfelt thanks and appreciation to my supervisor, ***Dr Panagiotis Kaklis****, for their priceless advice, continuous support and guidance throughout my PhD journey. Their counselling and patience, irrespective of this work, has helped me become not only a better researcher, but also a better person and for that I am immensely thankful.*

I would like to extend my gratitude to BAES Naval Ships and specifically the Research and Technology (R&T) team stationed in Scotstoun, Glasgow for their guidance and support during my PhD, without which this work would not have been possible. I express my deepest gratitude to ***Malcolm Robb*** *for his continuous assistance and communication during my PhD study and* ***Neil Harrison****, for his aid in narrowing down and defining the focus of my PhD, along with his help in the search for data that would be appropriate for our ideas. Furthermore, I also extend my heartfelt gratitude to* ***Gail Hughes*** *for her assistance in organizing the communication between me and R&T and her invaluable support during the preparation of my first publication and to* ***Josh Sullivan*** *who identified the dataset that was best suited to accommodate the needs of the developed methodology and problem. Finally, I would like to thank* ***Nabile Hifi*** *for his words of guidance during all meetings with BAES Naval Ships.*

Last but not least, I would like to express my cordial thanks to fellow researchers that I met and co-operated with during the period of my PhD. ***Sotiris Chouliaras*** *and* ***Shahroz Khan*** *have been valuable friends and helped me find my footing both in an office environment and virtually due to the pandemic. I also wish to extend my gratitude to* ***Adebowale Odukoya*** *for her support in relation to managing a PhD, since she started hers a few months before me and already had a similar experience.*

On a personal note, I would like to thank my family and friends for their unwavering support and understanding during this long journey.

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/T517665/1] and BAE Systems (BAES).

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF ABBREVIATIONS

AR: Augmented Reality

BC: Betweenness Centrality

BFS: Breadth – First Search

CC: Closeness Centrality

DC: Degree Centrality

DFS: Depth – First Search

D&M: Design and Manufacturing

EC: Eigenvector Centrality

EFV: Edge Feature Vector

GIRFT: Get It Right First Time

KC: Katz Centrality

LCC: Local Clustering Coefficient

MR: Mixed Reality

NCR: Non-Conformance

NFV: Node Feature Vector

PR: PageRank

SN: Social Networks

SNA: Social Network Analysis

TDG: Time – Dependent Graph

UI: User Interface

VR: Virtual Reality

ABSTRACT

The design of naval vessels is an intricate process that depends substantially on human expertise and judgment. The interplay between complexity and human involvement affects design outcomes in terms of delivery time, cost, and quality. A critical challenge in design chains is the emergence of rework tasks, whether planned or unplanned, which address errors from earlier stages or prevent potential non-conformances (NCRs) based on existing data. Rework, defined as unnecessary effort of redoing incorrectly implemented processes, manifests as either design-induced or construction rework.

This research introduces a novel representation of the naval design process using a bipartite dynamic social network. The critical role of human interaction in design necessitates a formal representation capturing both technical and social dimensions. The bipartite social network model explicitly maps relationships between design tasks and the individuals performing them, while its dual-layer structure enables analysis of how human expertise and collaboration patterns influence task execution. Given the timeframe and complexity of naval ship design, the continuous transition of participating individuals and design activities naturally leads to adopting a time-dependent graph (TDG) for process representation.

The research tracks the evolution of network metrics, including centrality, modularity, and clustering coefficients, to analyze the design process dynamics. This thesis comprises five chapters: an introduction to social networks and design rework (Chapter 1), model development and testing (Chapter 2), application to three industrial case studies proposed by the industrial supervisor (Chapter 3), network visualization options (Chapter 4), and future modeling directions focusing on rework aspects (Chapter 5).

Through this novel approach, the research provides designers and design supervisors with a comprehensive framework for understanding and managing the complex interactions between human factors and technical tasks in naval design. The temporal analysis of network metrics offers valuable insights into the nature of the design process and the criticality of specific tasks and individuals, enabling more effective management of design workflows and rework scenarios.

1 SOCIAL NETWORKS AND SHIP DESIGN

Before delving into the history and applications of social networks, it is important to establish the core research contributions of this work to naval ship design. Traditional approaches to managing complex ship design processes have relied on sequential planning tools like PERT diagrams and GANNT charts, which present significant limitations in capturing the dynamic and interconnected nature of modern naval design activities. While these tools excel at showing linear task dependencies and timelines, they struggle to represent the complex web of human interactions, iterative design processes, and emergent rework patterns that characterize contemporary shipbuilding projects.

This research introduces a novel approach by applying social network analysis and time-dependent graph theory to model naval ship design processes. Unlike traditional methods, this framework captures both the technical and social dimensions of design work by representing design tasks and participating individuals as nodes in a dynamic bipartite network. The temporal aspect of the network allows us to analyze how design relationships and task dependencies evolve throughout the project lifecycle - a critical capability not available in static planning tools.

The key research questions this work addresses are: (1) How can we effectively model and analyze the complex interactions between human actors and design tasks in naval ship projects? (2) What insights can time-dependent network analysis reveal about rework patterns and process optimization opportunities? (3) How can we leverage network metrics to improve design process efficiency? The main contributions include: a novel time-dependent bipartite network model for ship design processes, new metrics for analyzing design process robustness and efficiency, and practical insights for reducing rework through better understanding of human-task interactions. This approach differs fundamentally from existing methods by providing a dynamic, relationship-centered view of the design process rather than just a sequential task schedule.

This Chapter comprises 4 sections: Section 1.1 clearly provides the aim and the subsequent objectives of this thesis. Section 1.2 provides a brief introduction to social networks and a historical review, Section 1.3 discusses major contributions to social network analysis and important application and Section 1.4 introduces the reader to rework in design process, which is the objective kernel of this research.

1.1 RESEARCH AIM AND OBJECTIVES

The primary aim of this research is to develop a novel framework for modeling and analyzing naval ship design processes using time-dependent bipartite social networks. This approach seeks to capture both the technical and social dimensions of design work by representing design tasks and participating individuals as interconnected nodes in a dynamic network, thereby enabling a more comprehensive understanding of the complex socio-technical interactions that influence design outcomes and rework patterns. To achieve this aim, the research addresses the following objectives:

1. To develop an effective model for representing and analyzing the complex interactions between human actors and design tasks in naval ship projects through the application of bipartite social network theory and time-dependent graph analysis.
2. To identify what insights time-dependent network analysis can reveal about rework patterns and process optimization opportunities within naval design processes by applying the model to industrial case studies.
3. To determine how network metrics can be leveraged to improve design process efficiency and reduce rework through the analysis of centrality, clustering, and connectivity patterns in design networks.
4. To create visualization methods that make complex network relationships accessible to design managers, enabling more effective decision-making in naval design projects.

Through these objectives, the research aims to provide designers and design supervisors with a comprehensive framework for understanding and managing the complex interactions between human factors and technical tasks in naval design, with particular focus on identifying and mitigating rework scenarios.

1.2 BRIEF HISTORY OF SOCIAL NETWORKS

Social Networks (SNs) are a specific category of graphs covered in Graph Theory. The “father” of Graph Theory is considered Leonhard Euler, whose original paper in 1736 named “Solutio problematis ad geometriam situs pertinentis” (translated as: “Solution of a Problem Relating to the Geometry of Position”) set the foundations for graph theory and topology by solving the problem of the Seven Bridges

of Konigsberg (Euler, 1736). A graph is composed of two sets, i.e., vertices and edges. The vertices or nodes[1] can represent everything from systems in systems engineering, to cities in map and transportation applications and organs or blood receptors in medical applications. The edges represent the connections between the nodes e.g., streets in communication applications or arteries and veins in a circulatory network.

A network in which the vertices represent people and the edges represent connections (e.g. marital, friendship, co-worker) between people is called a Social Network. In 1934, Jacob Moreno introduced sociograms which employ nodes as individuals and edges to depict the interactions or relationships between them, marking the first appearance of a SN and a significant development in the study of social relationships (Moreno, 1934). His work is considered pioneering in the sense that he applied graphical representation techniques to social sciences, thus allowing for better understanding of the complexities of social interactions and dynamics. Sociograms have since been used in sociology, psychology, education and organizational studies and have laid the groundwork for Social Network Analysis (SNA). The next huge impact in SNA was made by Stanley Milgram in his seminal work "The Small-World Problem" in (Milgram, 1967). Milgram set out to investigate and quantify the connections between people in a network, i.e., the average path length for a social network of the people in the USA. He sent letters to a group of random individuals in a number of different states and asked them to forward the letter to a target person in Boston, by passing it to someone they knew in a first-name basis and who would be more likely to know the target person. Then the new recipient would follow the same process until the letter reaches the target person in Boston. The results showed that on average the target person would receive the letter after six recipients (or in a network approach, the average path length from first recipient to target was 6). This is known today in social studies as "six degrees of separation", i.e., most people in the world are on average separated by six acquaintances. The impact of this work on SNA is that SNs exhibit the small-world property where most vertices can be visited by a relatively small number of steps.

Today SNs are most commonly linked with social media platforms where acquaintances are connected by friendship edges (Facebook), following edges (Instagram) or connection edges (LinkedIn), and product retailer web applications (e.g. Amazon) where connections are more likely to have similar product needs

[1] Graph Theory categorises its sets as vertices and edges. In literature one can find Graph Theory termed as Network Theory and then the sets are labelled as nodes and connections. In this thesis Graph/Network Theory, vertex/node and edge/connection are used interchangeably.

or partake in similar activities. However, networks are utilised in great extent in engineering and science domains, whereas in the shipping industry they are found everywhere from ship-hull and component design to route optimisation and systems engineering.

1.3 SOCIAL NETWORK APPLICATIONS

The first essential handbook for SNs is found in (Wasserman & Katherine, 1994), where SNs and their mathematical representations are presented, their structural and locational properties are discussed, and first applications are demonstrated. In (Watts & Strogatz, 1998) 'small-world', per Milgram, networks are found to be highly clustered and with small path lengths and the model developed was used on the power grid of the Western US. One-mode or classic networks in different fields show similar commonalities in terms of path lengths and clustering (Albert & Barabasi, 2002), (Bornholdt & Schuster, 2002), (Watts & Strogatz, 1998), (Newman, 2003).

Two-mode or bipartite networks (named herein) have been applied in management science (Kogut & Walker, 2003), (Kogut, et al., 2007), finance and marketing (Battiston, et al., 2003), (Dahui, et al., 2005), (Garlaschelli, et al., 2005), (Kim, 1998) and file exchange (Voulgaris, et al., 2004), (Guillame, et al., 2004), (Ianmitchi, et al., 2010), (Le Fessant, et al., 2004). The basic properties (e.g. clustering coefficient, projections, average path length) are described in (Latapy, et al., 2008), where the clustering coefficient is evaluated mathematically. Another definition for the clustering coefficient is given by (Robins & Alexander, 2004), where the clustering coefficient is evaluated topologically. In the context of this thesis, both definitions are employed. In (Guillaume & Latapy, 2004) and (Guillaume & Latapy, 2006) they proved that all complex networks can be represented through a bipartite structure and the structure properties can be understood (and assessed directly) as a result of the underlying bipartite structure. The use of graph projections is the most common way to move from the bipartite setting to the classic (unipartite) graph setting. The main drawback of this transformation from bipartite to unipartite setting through projection is the irreversibility of the process, since many bipartite structures can lead to the same unipartite projection. (Borgatti & Halgin, 2014) define network centrality measures for bipartite networks, by updating earlier definitions used in classic graphs. A presentation for some of the most prevalent bipartite network metrics (properties) is given in (Piccolo, et al., 2018), where the design processes for a biomass plant design are modelled as a bipartite network of employees and activities, aiming on the exploration of the robustness of complex systems and the central role people play in a design pipeline.

Networks (bipartite or unipartite) are a powerful mathematical tool for modelling design processes of a relatively low number of people and activities (e.g. in (Piccolo, et al., 2018), ~100 people and ~150 activities). However, the target design of this thesis is the design of a naval ship, which based on the data given, involves a series of tasks of 10^4 order of magnitude (initial design) or 10^3 order magnitude (sister ships), that span for more than 10 years. Therefore, a static (classic) graph representation cannot be employed in this case for visualising the design network, since the output would be overcluttered with relative information, edges and vertices. The temporal dimension of the design process though, can be utilised as a way of reducing the number of tasks after transforming the design process to a dynamic (time-dependent) graph, where depending on the time stamp, the active design network changes its structure.

Time-dependent graphs (TDGs) have been utilised in social networks ((Qiu, et al., 2016), (Moinet, et al., 2015)), transportation networks ((Yang & Zhou, 2017), (Delling & Wagner, 2009)) and bioinformatics ((Marchetti-Bowick, et al., 2016), (Przytycka, et al., 2010)). A TDG can be thought of as a labelled graph, in which labels capture some measure of time, edges are activated based on sequences of time-dependent elements (Wang, et al., 2019). As expected, the use of a TDG instead of a static graph will increase the complexity of query processing and mining problems. A simple example of this is the shortest path evaluation between two nodes. In a static graph, the process is straightforward, starting from the 1st node and finding all the step 1 neighbours, then iterating for step 2 and so forth until identifying the target node in the neighbour set. In a dynamic setting, an extra variable must be considered. The static shortest path is not always the correct answer since at some time interval the target point could be disconnected from the graph. The complexity of temporal networks has been systematically studied and quantified through distinct analytical frameworks, with research demonstrating specific computational challenges. (Orda & Rom, 1990) established the foundational understanding by examining how time-dependent edge delays affect path optimality, showing that when edge costs vary with time, even basic properties like path concatenation and subpath optimality no longer hold, requiring fundamentally different algorithmic approaches from static networks. Building on this, (Sherali, et al., 1998) proved that when temporal constraints are added, even restricted versions of the disjoint paths problem become NP-hard, revealing that this increased complexity stems from the interaction between path selection and timing constraints. Finally, (Foschini, et al., 2011) provided precise quantification of this complexity increase. Together, these works demonstrate that temporal complexity in graphs is not merely significant but also precisely characterized through rigorous theoretical analysis, with challenges arising from fundamental changes in

path properties, computational hardness of basic routing problems, and provable lower bounds on solution complexity. An overview of a TDG framework is included in (Wang, et al., 2019).

1.4 REWORK IN ENGINEERING PROJECTS

The aim of industry 4.0 for Design and Manufacturing (D&M) projects is to fulfil the Get-It-Right-First-Time (GIRFT) notion, pointing to the very large number of iterations appearing in D&M pipelines. In discussions with industrial people for this PhD project, it was suggested that the field rework tasks in D&M process may amount up to 70% of the total D&M time and the cost due to rework activities is approximately 30% of the total build cost. This section includes some definitions regarding the rework-related terminology as well as sample publications on rework that are of significance for this PhD Thesis.

(Love, 2002) defines rework as "the unnecessary effort of redoing a process or task that was incorrectly implemented the first time". The relationship between human involvement in the D&M process and GIRFT philosophy is more nuanced than simple contradiction. While human participation introduces potential for variability and error due to natural human limitations in consistency, judgment, and attention, it also brings essential capabilities like adaptability, problem-solving, and experiential learning that can enhance quality outcomes. The key lies in structuring human involvement through standardized processes, clear protocols, and systematic quality controls that leverage human strengths while implementing safeguards against common human error patterns. This balanced approach recognizes that achieving GIRFT objectives requires thoughtfully integrating human capabilities with robust systems and controls, rather than viewing human involvement as inherently opposed to quality goals. Understanding this interplay allows organizations to design processes that maximize the value of human participation while maintaining the rigorous standards essential to GIRFT implementation.

The consequence of an error may be a rework non-conformance (NCR), which requires the planning and undertaking of an extra task (rework task) to handle the raised NCR. The rework task or rework framework of tasks is a costly problem as shown in (Hwang, et al., 2009) and (Love, et al., 2018) and schedule delays are inevitable (Han, et al., 2013). In many cases, organizations experience what (Love, et al., 2019) defines as organizational zemblanity; an unpleasant un-surprise. They know that their organizational structure will cause rework in specific regions, but they consider it part of the process instead of trying to find a solution to mitigate its impact.

Errors arising in D&M processes are typically the result of human errors. However, most errors come as a consequence of organizational and the environmental factors. The majority of errors are not random, but are closely linked to the tools people use, the tasks they perform and their work environment. An individual's cognitive ability can be compromised by a multitude of factors, mainly separated in three categories (Love, et al., 2019):

a) **pathogens** which are individual-related conditions the person is experiencing (personal circumstances), such as boredom or illness.

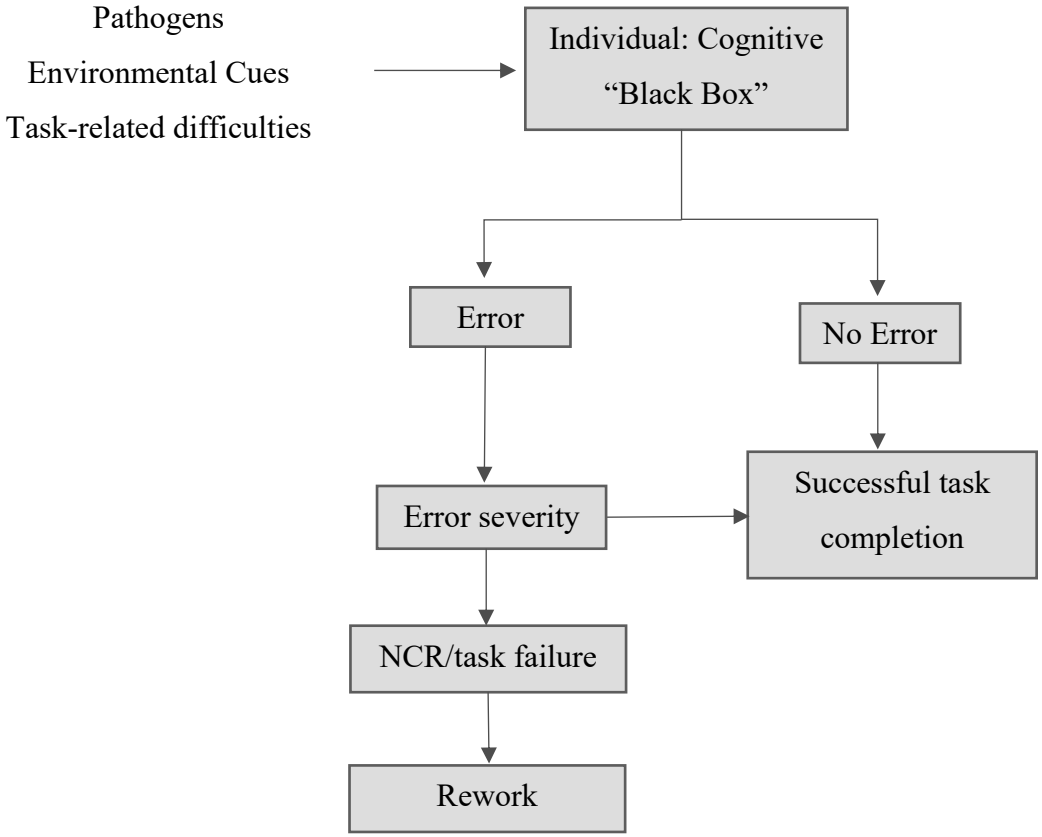

Figure 1-1 Path to rework

b) **environmental cues** or latent conditions, like failure to undertake design reviews or incomplete documentation related to the task at hand

c) **Task-related difficulties**, which have to do with the mode of operation and level of adversity of the task, like welding in uncomfortable angles, working in a noisy environment etc.

Depending on the magnitude of these factors, an individual may or may not commit an error (Figure 1-1). In case no error is committed, the task is completed correctly. Not all errors cause NCRs though, and

according to (Festinger, 1983) errors are necessary for the development of the organisation's culture, leading in a project's maturity and the path to innovation is paved with errors. Depending on the error severity, NCRs may be raised and then additional tasks, personnel, resources and time must be invested to remedy NCRs, i.e., generation of rework takes place.

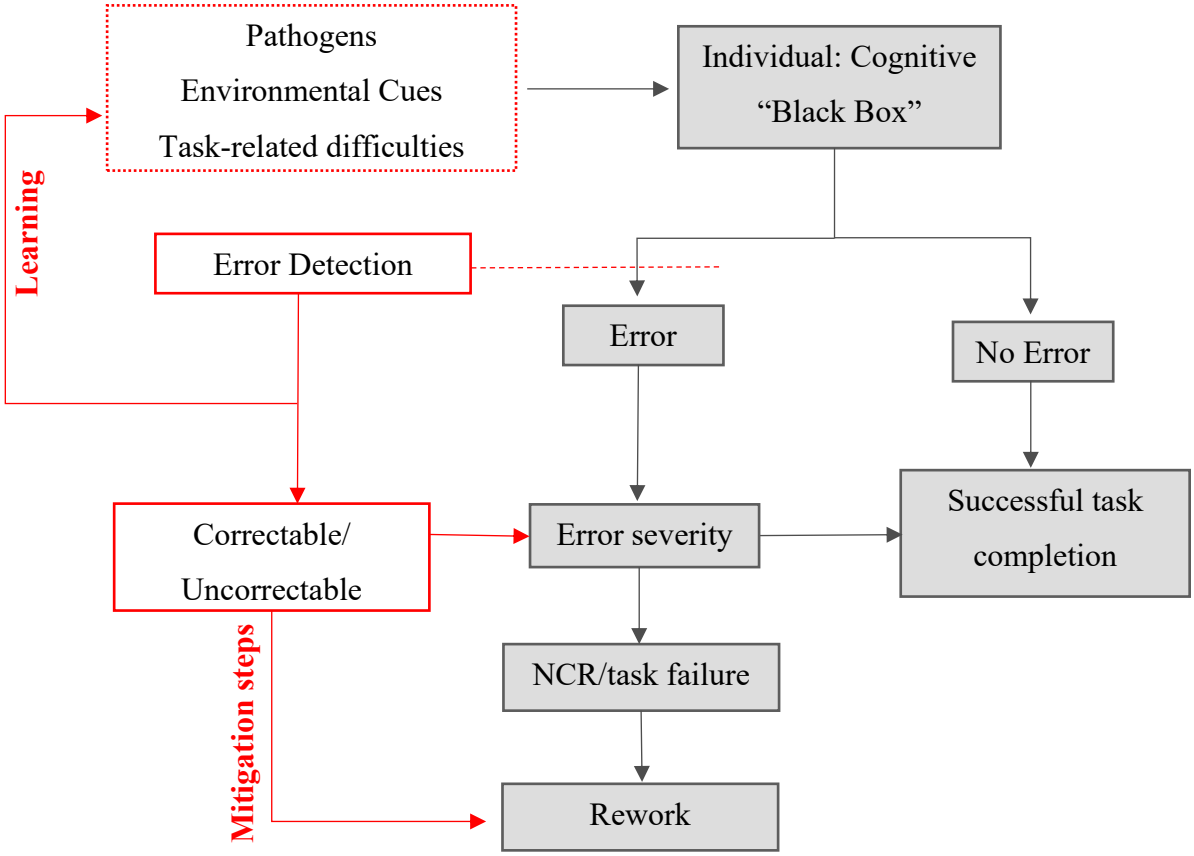

Figure 1-2 Path to rework including a rework analysis and mitigation module

The development and inclusion of a rework module is demonstrated in concept level in Figure 1-2. The error detection module identifies whether the committed error is immediately correctable or not. In the latter case, the necessary mitigation steps are taken to reduce the impact of the rework tasks in terms of cost and time delays, while the nature of the error is used to improve the organisation's working strategies and environment, aiming for fewer similar errors in the future.

Rework is a multidisciplinary problem, and its consequences vary based on the area of application. Figure 1-3 shows the impact of rework on total cost across different industrial fields. The highest value is 12.4% for heavy industrial engineering projects but this could reach up to 28-30% of total cost for defence projects. The majority of the added cost components relate to the delay due to rework. Figure 1-4 depicts

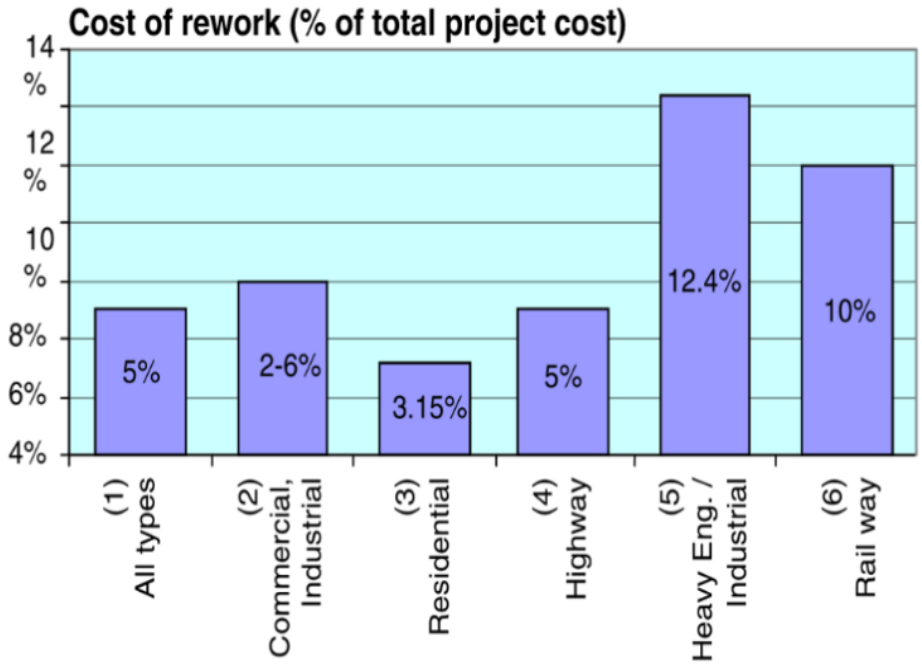

Figure 1-3 Cost impacts due to rework in different fields (Hegazy, et al., 2011)

a temporal analysis of a school building project progression, juxtaposing initial planning expectations against empirical measurements and simulated forecasts. The visualization employs a standard S-curve methodology to delineate four distinct trajectories: the baseline initial plan (represented as a gray dashed line), the documented actual progress (solid blue), the simulation-derived projection (red), and the anticipated progression curve after a rework tracking tool is employed (dotted blue). The most significant

chronological milestones are annotated, including the delay initiation (November 15, 2006), the 25% completion threshold (June 1, 2007), the contemporaneous assessment point (November 1, 2007), and two divergent completion projections—August 1, 2008 for when rework is tracked and November 15, 2008 according to simulation model that doesn't consider rework as an problematic issue. The quantitative divergence between these completion forecasts illustrates the inherent uncertainties in project

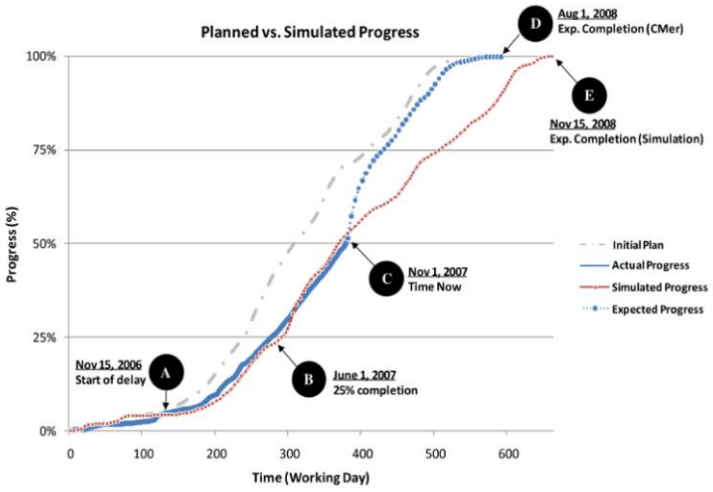

Figure 1-4 Rework impact and rework tracking effect (Love, et al., 2019)

management predictive methodologies, particularly following the emergence of schedule variances that stem from rework.

Figure 1-5 presents empirical evidence illustrating the adverse safety implications of rework activities across various incident categories. Most notably, high potential incidents—those with the greatest capacity for serious harm—exhibit the highest association with rework at 42%, substantially higher than the

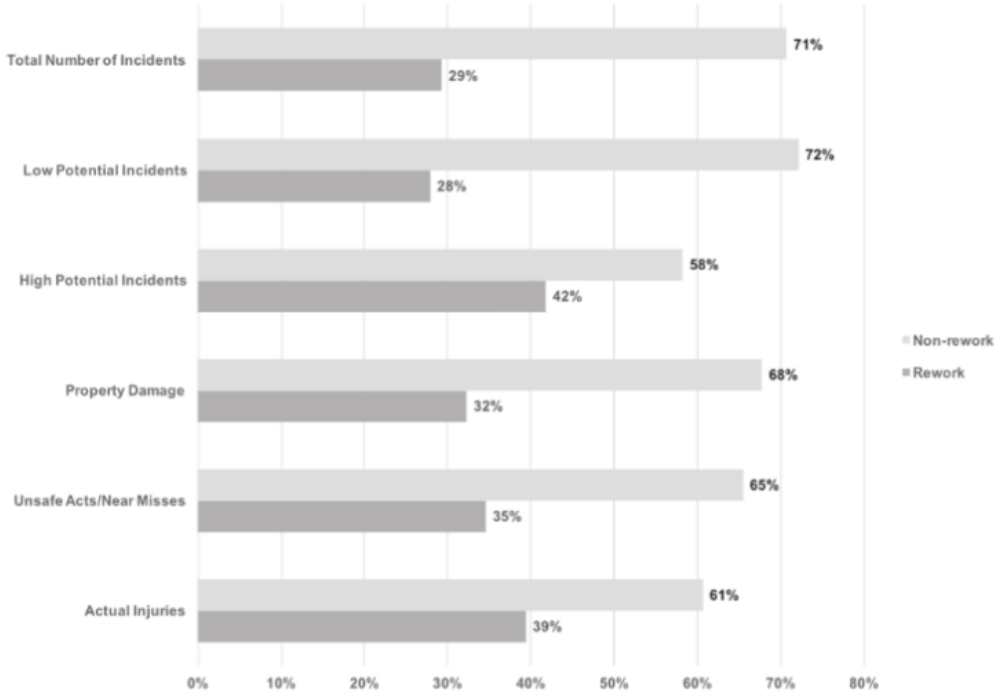

Figure 1-5 Rework and safety (Han, et al., 2013)

baseline rework rate of 29% for total incidents. This disproportionate representation suggests rework activities substantially elevate safety risks in potentially catastrophic scenarios. Similarly, actual injuries show a concerning 39% association with rework, indicating that repeated work processes significantly contribute to physical harm outcomes. Unsafe acts and near misses (35%) and property damage events (32%) also demonstrate elevated rework correlation compared to the overall average. Only low potential incidents maintain a rework association (28%) below the baseline. These percentages reveal a clear pattern wherein rework disproportionately contributes to more serious safety incidents, suggesting a complex relationship where the procedural disruptions, time pressures, or resource constraints inherent in rework scenarios create elevated risk conditions. The data strongly indicates that rework represents not merely an operational inefficiency but a substantive safety hazard, with potentially severe consequences for

workplace injury rates and incident severity profiles. These findings underscore the imperative to address root causes of rework as a fundamental safety management strategy, however safety data were not available, thus rework tracking impact on safety is part of the next steps for this line of work.

The complexity inherent in naval vessel design necessitates innovative approaches that transcend traditional modeling paradigms. In contrast to similar works in ship design, this research introduces a modelling framework that reconceptualizes the naval design process through the lens of network science, with particular attention to the critical issue of rework that significantly impacts project outcomes. At the core of this research's novel contribution is the development of a bipartite dynamic social network representation that explicitly acknowledges the socio-technical nature of naval design. Unlike conventional approaches that often separate technical processes from human factors (e.g. GANNT chart, PERT diagrams), this model integrates both dimensions by formally mapping the relationships between design tasks and the individuals performing them. This dual-layer structure enables analysis of how human expertise, collaborative patterns, and communication networks influence task execution and, by extension, design outcomes.

The incorporation of temporal dynamics through time-dependent graphs (TDGs) represents another significant advancement in the field. Naval design projects typically span years, involving personnel transitions, evolving requirements, and emergent knowledge. Where traditional models offer static representations, the proposed approach captures the evolutionary nature of design networks, revealing how disruptions propagate, how expertise transfers during personnel changes, and how collaborative structures adapt throughout the project lifecycle. This temporal dimension is particularly crucial for understanding rework dynamics, as it allows tracing the origin and propagation of design changes across both time and organizational boundaries. The temporal dimension granularity can be controlled by the system user as days, weeks, months and years.

The research further distinguishes itself through its specialized analytical framework for characterizing rework in naval design. By differentiating between design-induced and construction rework, and between planned and unplanned iterations, this approach provides nuanced insights into rework patterns that remain invisible to conventional analysis. Network-based metrics such as centrality, modularity, and clustering coefficients offer new perspectives on process vulnerability, communication bottlenecks, and

critical knowledge nodes that potentially contribute to rework scenarios, which by itself is an advantage compared to traditional ship design process representation methods.

Empirical validation through three industrial case studies demonstrates the practical applicability of this theoretical framework. These real-world three cases studies are used for validating the utility of the approach for complex naval design projects. The case studies reveal patterns in how network structures evolve during periods of intensive rework, identifying signature topological changes that could serve as early warning indicators for emerging design issues. The research also explores innovative visualization techniques specifically designed to make complex network relationships accessible to design managers. These visual analytics tools transform abstract network data into intuitive representations that illuminate process dynamics, making structural vulnerabilities and critical dependencies immediately apparent to decision-makers who may lack specialized knowledge in network science.

2 DESIGN PROCESS MODEL

The modelling of complex engineering design processes, particularly within naval architecture, has traditionally relied on linear or hierarchical representations that often fail to capture the intricate socio-technical dynamics at play. This chapter presents a novel framework that reconceptualizes naval design as an evolving, interconnected network of human actors and technical activities, challenging the conventional reductionist approaches to design process management.

The philosophical underpinning of this research adopts a critical realist stance, acknowledging that naval design exists as both a concrete technical reality and a socially constructed activity mediated through human expertise, judgment, and collaboration. This perspective recognizes that design complexity emerges not merely from technical challenges but from the interplay between human knowledge networks and technical task dependencies—a perspective that has been inadequately addressed in traditional naval design literature. By developing a bipartite dynamic social network model, this research bridges engineering analytics with insights found in social network theory. This integration enables a more holistic examination of how design knowledge is created, transformed, and sometimes compromised through complex human interactions across organizational boundaries and temporal constraints. The choice of network science as a methodological framework reflects a fundamental shift away from linear cause-effect paradigms toward understanding design as an emergent phenomenon within a complex adaptive system.

The innovative application of time-dependent graphs to naval design represents a significant advancement, recognizing that knowledge flows and decision-making processes evolve dynamically throughout project lifecycles. This temporal dimension is particularly crucial for understanding the genesis and propagation of rework—a phenomenon that traditional static models have failed to adequately characterize or predict. Within this theoretical framing, the research questions examine not only how network properties correlate with design outcomes but more fundamentally how the structural characteristics of these networks reflect underlying organizational and knowledge management challenges in naval design. The methodology employs a mixed-methods approach, combining quantitative network analytics with qualitative analysis of design decision pathways to develop a more nuanced understanding of design process dynamics.

This chapter details the mathematical foundations and computational framework for this novel modeling approach, beginning with fundamental network properties (Section 2.1) before progressing to more

sophisticated temporal analytics (Section 2.2) and extra material on network robustness (Section 2.3). The model development (Section 2.4) process was iterative and challenging, constantly engaging with both theoretical insights from complex systems literature and practical constraints from industrial contexts. This approach ensures that the resulting framework not only advances the theoretical understanding of naval design processes but also offers practical utility for design management in real-world settings. By establishing this innovative perspective on naval design processes, this research creates a new analytical lens through which to examine the critical challenges of design rework, knowledge transfer, and process resilience—aspects that have profound implications for project performance but have remained elusive to traditional analysis methods. The subsequent sections of this chapter detail the precise mathematical formulation of this modeling approach and its computational implementation, setting the foundation for the empirical case studies presented in Chapter 3. Figure 2-1 presents an overall methodology diagram from the beginning of the study until the end of the results section.

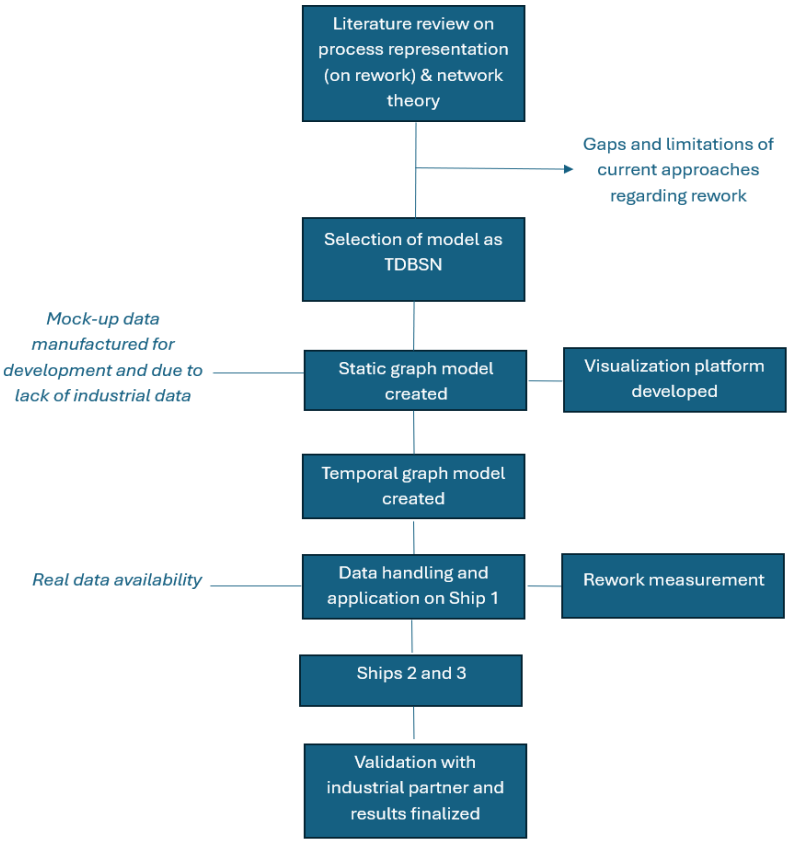

Figure 2-1 Methodology diagram

2.1 BIPARTITE NETWORKS

2.1.1 Clustering properties

A **network** or a **graph** $\boldsymbol{G}$ is formally defined as an ordered pair $G = (N, E)$ consisting of a non-empty set N of **nodes** or **vertices** and a set E of **edges** or **connections**, which are 2-element subsets of N. That is, an edge $e \in E$ is of the form $e = \{u, v\}$ where $u, v \in N$ and $u \neq v$.

A **bipartite graph** $\boldsymbol{B} = (U, V, E)$ is a special type of graph whose vertices can be partitioned into two disjoint and independent sets U and V such that:

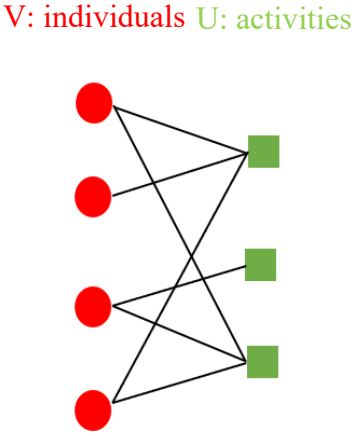

Figure 2-2 Example bipartite graph

1. The vertex set N is the union of two disjoint non-empty sets: $N = U \cup V$ where $U \cap V = \emptyset$.
2. Every edge $e \in E$ connects a vertex in U to a vertex in V. Formally, $E \subseteq (u, v) : u \in U, v \in V$.
3. There are no edges between vertices within the same set. Formally, $\forall\, u_1, u_2 \in U, (u_1, u_2) \notin E$ and $\forall\, v_1, v_2 \in U, (v_1, v_2) \notin E$.

Equivalently, a graph $\boldsymbol{G} = (N, E)$ is bipartite if and only if its vertex set N can be partitioned into two subsets such that every edge in E has one endpoint in each subset, and no edge has both endpoints in the same subset.

An alternative characterization of bipartite graphs is that a graph is bipartite if and only if it does not contain any cycles of odd length. This is a fundamental result in graph theory known as the "Bipartite Graph Theorem". In the context of a directed bipartite graph, it is defined as $\boldsymbol{B} = (U, V, E)$ where $E \subseteq (u, v) : u \in U, v \in V \cup (v, u) : v \in V, u \in U$, meaning that edges have a direction and can go from U to V or from V to U. For weighted bipartite graphs, we further define a weight function $w: E \to R$ that assigns a real value to each edge.

The graph **density** δ_G measures the fraction of existing links to all possible ones and for bipartite graphs it is:

$$\delta_G = \frac{|E|}{|U||V|} \quad (2.1)$$

where $|(\cdot)|$ is the number of elements in the $(\cdot)$ set. The **clustering coefficient** in classic graphs measures the prevalence of triadic closure in the network and there are two main measures: clustering coefficient (local/node LCC or global GCC) and **transitivity**. The graph global clustering coefficient can be derived by averaging LCC for all nodes. Transitivity measures the percentage of "open triads" that are triangles in the network. In bipartite graphs the concept of triadic closure is not applicable since triangles are inherently illegal. In (Latapy, et al., 2008), an expansion of LCC is presented where, instead of node local clustering, a clustering coefficient for pairs of nodes is used:

$$c_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}, \quad (u, v) \in U \text{ or } V \quad (2.2)$$

where $N(i)$ is the **neighbourhood** set of node i. This definition depicts clustering as an overlap between neighbourhoods. In Figure 2-3, three different cases for two individuals who work on at least one common activity are presented. In the first case (left), individual u is connected to 5 activities ($|N(u)| = 5$) and individual v is connected to 3 activities ($|N(v)| = 3$). They collaborate in two activities

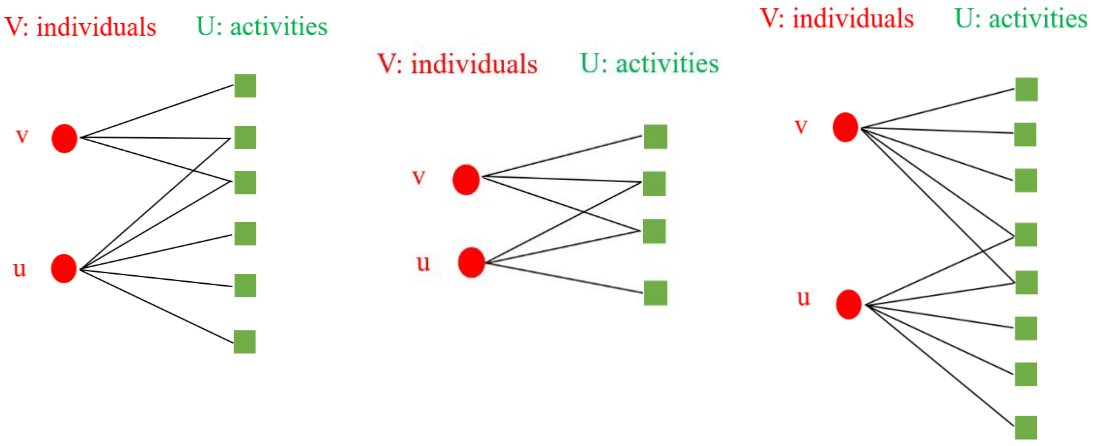

Figure 2-3 Pair clustering coefficient for different neighbourhood sizes

($|N(u) \cap N(v)| = 2$) and the total activities are 6 ($|N(u) \cup N(v)| = 6$). Therefore, the pair clustering coefficient is $c_{uv} = 2/6 = 0.33$. For the second (centre) and third (right) cases the number of activities changes and this results in pair clustering coefficients of 0.5 and 0.25, respectively.

(Latapy, et al., 2008) defines the clustering coefficient for a node as the average of the clustering coefficient with all the other nodes in the network:

$$c_u = \frac{\sum_{v \in N(N(u))} c_{uv}}{|N(N(u))|} \quad (2.3)$$

where $N(N(u))$ is the neighbourhood of node u at distance 2 (2nd order neighbours). An averaging process with respect to the two bipartite sets returns the global average clustering coefficient:

$$C(G) = \frac{\sum_{u \in G} c_u}{|U| + |V|} \quad (2.4)$$

which is termed as **Latapy clustering** $C(G)$. Similarly to Latapy clustering, an expansion of average LCC, the **Robins-Alexander clustering** $CC_4(G)$ (Robins & Alexander, 2004) is an expansion of the notion of transitivity. Instead of triangles, the simplest cycle (Figure 2-4**Error! Reference source not found.**) in a bipartite graph is a square (C_4) and instead of open triads, paths of length three (L_3) are used:

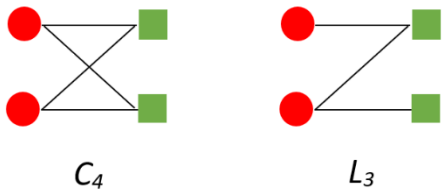

Figure 2-4 Simple cycle and path of length three

$$CC_4(G) = 4\frac{C_4}{L_3} \quad (2.5)$$

Considering the example cases of Figure 2-3, for the first network there is only one square formation and 12 paths of length three. From equation (2.5), the resulted transitivity value is 0.33. For the second and third network, the paths of length three are 8 and 16 respectively, leading in Robins-Alexander clustering values of 0.5 and 0.25. In these examples, the Latapy clustering coefficient and Robins-Alexander

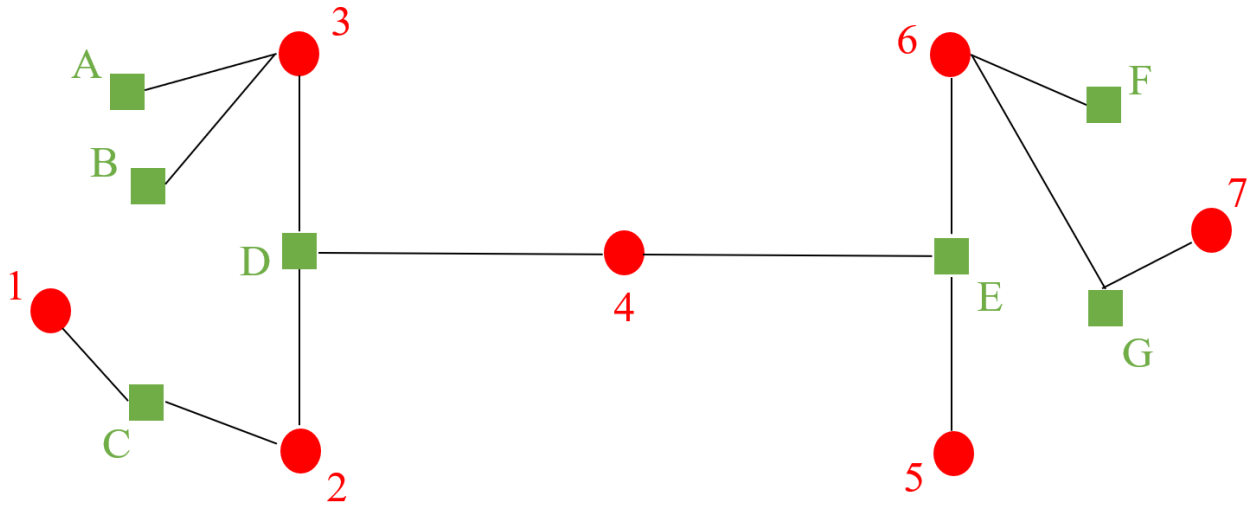

Figure 2-5 Example network for centrality metrics

clustering are the same. This is not the case for networks that include few or no square topologies, such as the one depicted in Figure 2-5. In this case, the Latapy clustering is 0.387 and the Robins-Alexander clustering is 0 since there are no squares.

2.1.2 Centrality properties

Another important measure of the network is centrality. There are various centrality metrics that can be employed, each with its own use and insight into the network connectivity. In this paper, degree centrality, betweenness centrality, and closeness centrality are calculated in node-wise form. (Borgatti & Halgin, 2014) provides an analysis of centrality measures for bipartite social networks.

Degree centrality DC of a node is the fraction of nodes connected to it; the normalisation being performed with the opposing set of the node under examination. The benefit of this normalisation is the immediate depiction of whether a given individual is more central than a given activity, i.e., it allows for comparative analysis across sets.

$$DC(u) = \frac{d_u}{|V|}, \qquad for\ u \in U \qquad (2.6a)$$

$$DC(v) = \frac{d_v}{|U|}, \qquad for\ v \in V \qquad (2.6b)$$

where d_i is the degree of node i.

Closeness centrality CC of a node is the sum of the distances to all other nodes in the graph. It measures how close a node is to all the other nodes. Closeness centrality is normalised with the minimum distance possible. For a bipartite graph, the minimum distance is 1 for connections across sets and 2 for connections within sets.

$$CC(u) = \frac{|V| + 2(|U| - 1)}{\overline{d_u}}, \quad for\ u \in U \qquad (2.7a)$$

$$CC(v) = \frac{|U| + 2(|V| - 1)}{\overline{d_v}}, \quad for\ v \in V \qquad (2.7b)$$

where $\overline{d_i}$ is the sum of distances from i to all other nodes. This normalisation results in higher CC values for nodes with high centrality.

Betweenness centrality BC of a node is the sum of the fraction of shortest paths that pass through the node. Its main advantage is that it accounts for nodes that have few neighbours (low DC) but connect different regions of the graph; nodes that act like connectors of distant clusters of nodes. Values of betweenness are normalised by the maximum possible value which, for bipartite graphs, is limited by the relative size of the two node sets.

$$b(i) = \sum_{i \neq k \neq j \in (U \cup V)} \frac{g_{jk}(i)}{g_{jk}} \qquad (2.8)$$

where $b(i)$ is the betweenness value of node i, g_{jk} is the number of shortest paths from node j to node k, $g_{jk}(i)$ the number of shortest paths from node j to node k passing through i. The normalising quantities are:

$$b_{Umax} = \frac{1}{2}[|V|^2(s+1)^2 + |V|(s+1)(2t - s - 1) - t(2s - t + 3)] \qquad (2.9)$$

$$s = \frac{|U| - 1}{|V|}, \qquad t = (|U| - 1)\, mod |V|$$

The normalising quantity for set V is obtained by alternating U and V in relations above. After normalisation the betweenness centrality is:

$$BC(u) = \frac{b(u)}{b_{Umax}}, \qquad for\ u \in U \quad (2.10a)$$

$$BC(v) = \frac{b(v)}{b_{Vmax}}, \qquad for\ v \in V \quad (2.10b)$$

Figure 2-5 depicts a network that illustrates the importance of investigating network centrality with a variety of centrality metrics, each returning a different "more central" node (Anagnostopoulos & Kaklis, 2023). Degree centrality ranks nodes with more connections as more central in the network. This would mean that nodes 3, 6, D and E are more central since they have the maximum number of connections which is 3. The resulting centrality value is $DC(3) = DC(6) = DC(D) = DC(E) = 0.4286$. Closeness centrality measures how close a node is to all other nodes. By examining the example network, one would assume that nodes near the network centre, i.e., nodes D, E, 4 are going to have the highest centrality values. The resulting centrality values are $CC(D) = CC(4) = 0.5758$ and $CC(E) = 0.5423$. One would assume that nodes E and D would have the same value for closeness centrality. They have the same number of paths with length 5: E→4→D→2→C→1, D→4→E→6→G→7. However, node D has only one path of length 4, D→4→E→6→F, while node E has two such paths, E→4→D→3→A and E→4→D→3→B, which means that node E is less central than node D. Examining the betweenness centrality values, it is expected that nodes D, E, and 4 would be more central, since removal of node D results in 3 connected networks, removal of node E results in 2 connected networks and an unconnected node 5, and removal of node 4 results in 2 connected networks. The betweenness centrality values are $BC(D) = 0.7083, BC(E) = 0.6111$ and $BC(4) = 0.5833$. Betweenness centrality ranks nodes as the more important connectors of distant clusters of nodes. Therefore, evaluating the betweenness centrality of node 1 returns zero since it is not a connector, but a node on the periphery. For node 1, the centrality values are: $DC(1) = 0.1489, CC(1) = 0.3016$ and $BC(1) = 0$.

Eigenvector centrality EC^2 is defined as the principal eigenvector of the adjacency matrix of a graph (Bonacich, 1972). Using this type of centrality, a node's centrality value is proportional to the sum of the centrality values of its neighbours. Practically EC measures the influence of a node in the network, by assigning scores to all vertices in the network (connected) based on the concept that connections to high scoring nodes contribute more to the score of the node than equal connections to low scoring nodes.

$$EC(i) = \frac{1}{\lambda} \sum_{j \in N(i)} EC(j), \quad for\ i \in U,V \quad (2.11)$$

Assuming that the centrality values should be always non-negative (Perron – Frobenius theorem) the λ should be the largest eigenvalue of the bi-adjacency matrix (Newman, 2006).

PageRank is a variant of eigenvector centrality developed by Google (Page, 1997) to rank web pages in their search engine results. The evaluation operation counts the number and the quality (weighted edges) of the links to a page to determine the importance of the page. The basic assumption behind it is that more important pages are more likely to receive more links from other pages. Assume a network with 4 nodes (Figure 2-6). The starting PageRank for each node is assumed to be 0.25 $PR(A) = PR(B) = PR(C) = PR(D) = 0.25$, so that the sum of the PageRank of the nodes is 1 (treated as a probability

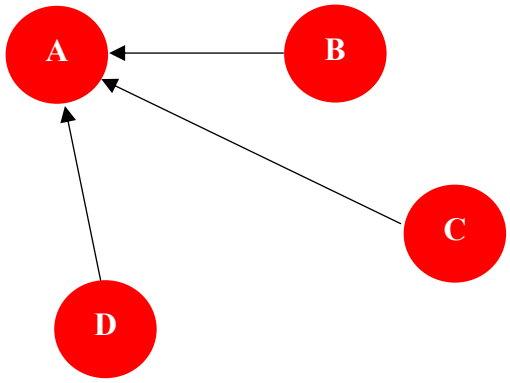

Figure 2-6 PageRank example (1)

[2] Eigenvector centrality, Page Rank and Katz centrality calculators were developed (not readily available for bipartite networks) and tested on the given dataset during the late stages of the study when data availability became easier, thus not included in (Anagnostopoulos & Kaklis, 2023), but incorporated in the thesis.

distribution between 0 and 1). Applying the algorithm, each link would transfer the PageRank of the originator of the edge to the destination node.

$$PR(A) = PR(B) + PR(C) + PR(D) = 0.75$$

A more interesting example is displayed in Figure 2-7. Again, the starting PageRank of each node is 0.25. Applying the algorithm node B would split its value to nodes A and C, node C would give its value to A and node D would split its value to nodes A, B and C. Then:

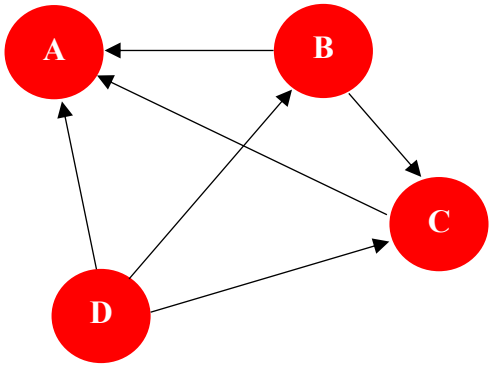

Figure 2-7 PageRank example (2)

$$PR(A) = \frac{PR(B)}{2} + PR(C) + \frac{PR(D)}{3} = 0.458$$

In general, the PageRank value transferred by an edge is the PageRank value of the origin of the edge divided by the number of outbound edges L from that origin. A generalised equation for the PageRank is:

$$PR(i) = \sum_{j \in N(i)} \frac{PR(j)}{L(j)} \quad (2.12)$$

Katz centrality *KC* (Katz, 1953) is a measure of the centrality of a node in a social network which considers the total number of paths between a pair of nodes, instead of the shortest path (geodesic distance). Katz centrality initially identifies all distance 1 nodes of the node under evaluation i , i.e., $N(i)$. Then it considers all other nodes that have a path to i through nodes belonging in $N(i)$.

$$C_{Katz}(i) = \sum_{k=1}^{\infty} \sum_{j=1}^{n} a^k (A^k)_{ji} \quad (2.13)$$

Connections made with distant neighbours (long paths) are penalized with an attenuation factor α. Each path then is assigned with a weight determined by a. Assume that A denotes the adjacency matrix of a network and a_{ij} are elements of A that take value 1 or 0 depending on whether i, j are connected or not with an edge. k represents whether i, j are connected through intermediary nodes, e.g., A^2 means that i, j are connected at distance 2 (2nd order neighbours).

2.1.3 Distance properties

In most network applications, the distance from one node to another is an important property that entails vital information. The first attempt to measure distance was by (Milgram, 1967), which was also the first attempt to quantify networks of people. The most common example is a road infrastructure network, where distance can be easily quantified by length units. The more prevalent technique is to add weights on the edges, which represent the length between the nodes that each edge connects. However, there are applications where an edge represents a connection type (relation, friendship, marriage etc.), as in social media. In these types of networks, the distance between two nodes can be defined as the number of edges one must go through in order to get from node 'A' to node 'X'. The use of such measures helps answering distance questions such as:

- How "far" are two nodes from each other?
- Are nodes far away or close to each other?
- Which nodes are "closest" and "farthest" to other nodes?

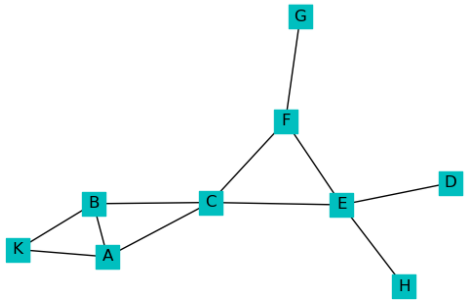

Figure 2-8 Example network

Distance between two nodes is measured simply by the number of edges in paths connecting these two nodes. For bipartite graphs, the essence of these measures does not change. The only thing that changes is the minimum distance of two nodes belonging in the same set. For classic graphs, this is 1, while for bipartite graphs it is 2. The following pages discusses briefly some of these measures, their definition and examples applied to Figure 2-8.

Path is a sequence of edges connecting distinct nodes. Usually there are more than one paths to reach from a node to another.

- Path I: D→E→C→B
- Path II: D→E→F→C→B
- Path III: D→E→C→A→K→B

In this example, some of the paths from D to B are examined. How far is node D from B? Path I shows that D is 3 "hops" (edges) from B, Path II shows that D is 4 "hops" from B and Path III shows that D is 5 "hops" from B.

Path length is the number of edges ("hops") a path contains from the 1st node to the last node.

- Path I length: 3
- Path II length: 4
- Path III length: 5

Distance between two nodes is the length of the shortest path between the two nodes. In the given example (Figure 2-8), the distance from D to B is:

$$d_{DB} = distance(D, B) = 3$$

In most real-life applications, the functionality of examining a single node is necessary, so finding the distance of a particular node to all other nodes is advantageous. This is a relatively simple process to do manually in small networks, however it is very tedious in large (most engineering and data-related) networks, thus a systematic way is needed.

Breadth – First Search (BFS) provides a systematic and efficient procedure for computing distances from a node to all other nodes in a network by "discovering" nodes in layers (Korf, 1985). The application of BFS on node A from Figure 2-8 is displayed below in Table 2-1 which is constructed step by step. At step 1, node A's connections in the network are examined, resulting to B, C and K, i.e., they are distance 1 from A. At step 2, node B's, C's and node K's connections are examined, and the result is that K is connected to A and to B, but both have already been written down in the table, so no new connections are added. B is connected to A, C and K, so no new connections are added. C connected to F and E so thus it is added and one more distance unit is added. At step 3, E is connected with D and H and F is connected with G. At step 4, only H has new additions, since it is connected with J. Now all nodes have been recorded in the table so BFS algorithm has been completed. Concluding the application of the BFS algorithm for A shows that A has distance 1 from B, C and K, distance 2 from E and F, distance 3 from D, H and G, and distance 4 from J.

Table 2-1 Example of BFS application

Step	Nodes	Distance
0	A	-
1	B C K	1
2	E F	2
3	D H G	3
4	J	4

From this straightforward application, the BFS algorithm can be seen as a tree-graph starting from the node under investigation and evolving until all nodes appear once in the tree-structure. So far, this discussion has been limited to measures that relate to specific nodes in the graph. The next set of properties are global metrics of distance in a graph.

Average distance of a graph is simply the average distance between every pair of nodes.

$$\bar{d} = \frac{1}{|U \cup V|^2} \sum_{i \in (U \cup V)} \sum_{j \in (U \cup V)} d_{ij} \qquad (2.14)$$

Eccentricity is the largest distance between a node and all other nodes.

$$e_i = \max\left(d_{ij}\right), \ j \in (U \cup V) \quad (2.15)$$

This is a node-specific measure, but it is shown here because it relates to the next global measures.

Diameter is maximum distance between any pair of nodes.

$$D(G) = \max(e_i), \qquad i \in (U \cup V) \qquad (2.16)$$

Radius of the graph is the minimum eccentricity.

$$R(G) = \min(e_i), \qquad i \in (U \cup V) \qquad (2.17)$$

Periphery is the set of nodes that have eccentricity equal to the diameter.

$$P(G) = \{i \,|e_i = D(G)\}\,, \qquad i \in (U \cup V) \quad (2.18)$$

Center is the set of nodes that have eccentricity equal to the radius.

$$K(G) = \{i \,|e_i = R(G)\}, \qquad i \in (U \cup V) \quad (2.19)$$

There is an intuitive relationship (reminds the case of the circle) between diameter, periphery, radius, center and eccentricity. First the eccentricity of all the nodes is evaluated, i.e., the largest distance (greatest shortest path) of each node and all other nodes, so all nodes relate to a single value of eccentricity. The highest eccentricity is called the diameter and the nodes that possess this highest eccentricity form the periphery of the graph. The lowest eccentricity is called the radius and the nodes that have this lowest eccentricity form the center of the graph.

2.2 TIME-DEPENDENT GRAPHS (TDGS)

Activities and individuals involved in activities are not always active during the design process. For example, activities pertaining to detailed design and analysis are inactive during early design stages. Considering the time-evolution of the network from concept design to detailed design completion, an individual may have worked on several tasks sequentially. Using a "static graph" approach (Wang, et al., 2019), this individual is connected to those tasks concurrently, so this limitation of the static model can lead to erroneous conclusions and poor decision-making. The conversion of the static graph to a **time-dependent graph** (TDG)[3] solves these issues and simplifies the network, since the static graph is the superposition of all time instances of the TDG (discrete TDGs). For creating the TDG, two methods are considered: variable weights and presence function. In the variable weights method, depending on the design process stage, the edges of the graph are assigned a weight value which quantitatively represents the information flow between activity and individual. The advantage of

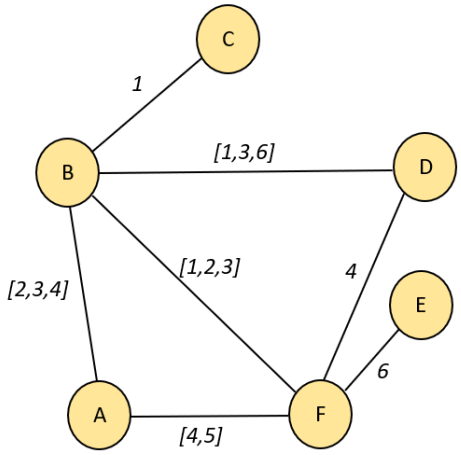

Figure 2-9 Time – Dependent Graph

[3] TDG and dynamic graph are used interchangeably in this thesis.

this approach is that edges can be activated, deactivated or semi-activated, when an activity is put on hold. The main drawback of the variable weights approach is the heavy dependence on the data, documented by the design company, related to quantifying the put-on-hold level and subsequently on the formulation of the weight function of each edge.

Due to the above weakness, the presence function approach is used as described in (Wang, et al., 2019). Using the presence function, edges are either active or inactive and this can be deduced from the available data (example in Figure 2-9). The presence function takes value 1 when the edge is active in the time interval under examination, and 0 when the edge is inactive. The use of TDGs allows the examination of the behaviour of the graph properties at different stages of the design process as all properties can be plotted as functions of time.

There are two main categories for distinguishing dynamic graphs based on the time discretization approach: discrete TDGs and continuous TDGs. An edge on a discrete dynamic graph is activated by disjoint time intervals. A way to model a discrete TDGs is to use labelled edges: $\boldsymbol{B} = \big(V, U, E, \lambda(G)\big)$, where $\lambda(\boldsymbol{B})$ is the label set which is a collection of natural numbers on each edge (Chen, et al., 2017), (Huang, et al., 2015). Another way is to build snapshots for the TDG (Braha & Bar-Yam, 2009): $\boldsymbol{B} = (V, U, E)$, where E is a set of time-dependent edges. A function $E(t)$ contains all edges in $\boldsymbol{B}$ at time t, leading in a TDG which is a sequence of static graphs $\boldsymbol{B} = \{B_1, B_2, \dots, B_t\}$. The third method is to build copies of vertices in order to convert a TDG to static graphs and take advantage of the well-built existing technologies perfected for static networks. The trick in this method is to do the conversion without loss of information, which is rather difficult (Wu, et al., 2014).

There are two major drawbacks of the discrete model for a dynamic graph. Expectedly, the model cannot represent the state of the graph between two discrete time points, which can lead to inaccuracy. Then one has to consider the memory and processing requirements. If precision is of utmost importance in the model, then the use of a continuous time-modelling is unavoidable. The most prevalent method and the one used in this work is the presence function method, which determines whether an edge is active or not (described in (Riolo, et al., 2001)). The second method for continuous-time modelling is the variable weight option, which treats the TDG as a flow-dependent graph, i.e., the weight it takes to traverse an edge is different depending on the time interval (Foschini, et al., 2011). An overview of the different options for time-modelling for dynamic graphs is presented in Figure 2-10.

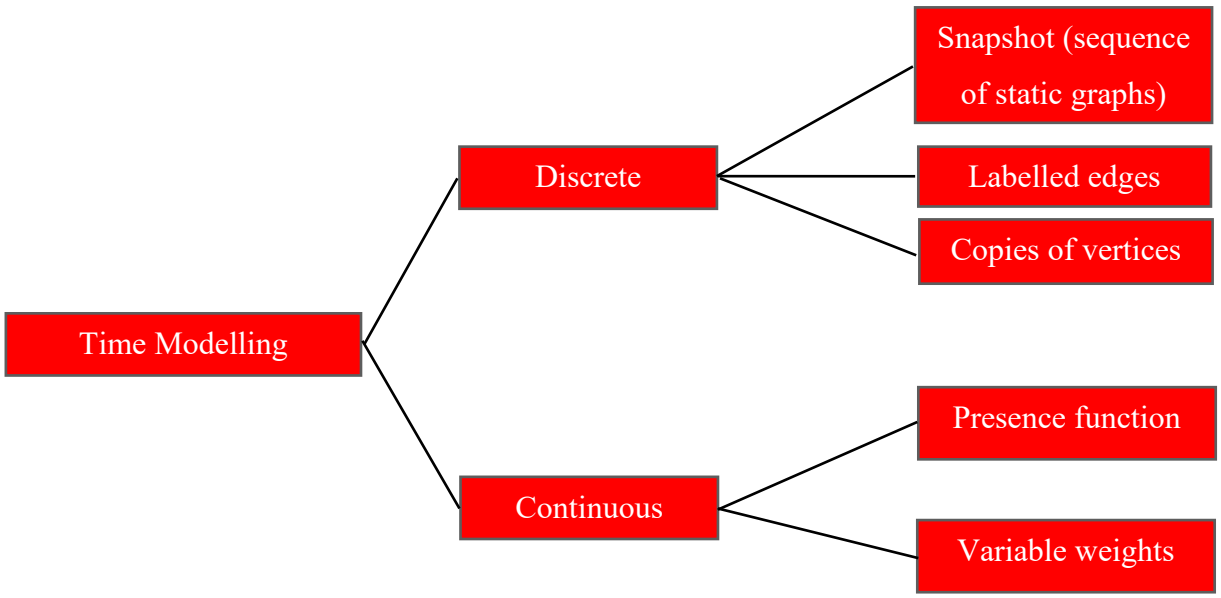

Figure 2-10 Overview of time-modelling techniques for TDGs

2.3 CONNECTED COMPONENTS AND NETWORK ROBUSTNESS

An undirected graph is **connected** if for every pair of nodes, there is a path between them. For example, in Figure 2-11, if edge (4,6) is removed, then the graph is split in two parts {1,2,3,4,5} and {6,7,8,9,70}. The graph is disconnected, since there is no path from one set to the other and two separate communities exist.

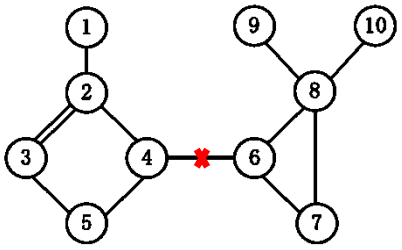

Figure 2-11 Connected components in graph

A **connected component** is defined as a subset of nodes such as:

- Condition 1: Every node in the subset has a path to every other node in the same subset.
- Condition 2: Nodes not belonging to the subset have no path to any node in the subset.

Taking subset {2,4,3,8} in the Figure 2-11 one can notice that there is no path from 2 to 8, since edge (4,6) has been removed, so condition 1 is not met and this subset is not a connected component. Considering subset {2,3,4}, condition 1 is met, but the condition 2 is not, since nodes 1 and 5 have paths leading in the subset, so it is not a connected component.

A directed graph is **strongly connected** if, for every pair of nodes *u* and *v*, there is a directed path from *u* to *v* and from *v* to *u*. A directed graph is **weakly connected** if replacing all directed edges with undirected edges produces a connected undirected graph. For Figure 2-12, the graph is not strongly connected, since there is no path from G to F, however it is weakly connected. The same conditions apply for a strongly connected component or a weakly connected component.

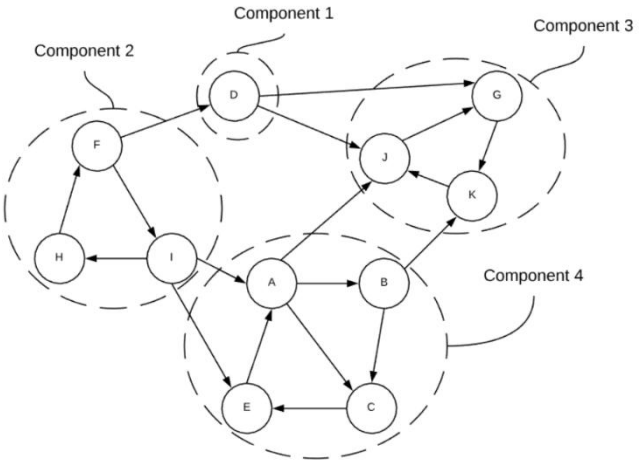

Figure 2-12 Connected components in directed graph

Network robustness is the ability of a network to maintain its general structural properties when it faces failures or attacks leading to removals of edges or nodes. The structural properties of the graph are measured by its connectivity.

Connectivity can be defined as:

- **Node connectivity**: Minimum number of nodes needed to disconnect a graph or a pair of nodes.
- **Edge connectivity**: Minimum number of edges needed to disconnect a graph or a pair of nodes.

Obviously, graphs with large node and edge connectivity are more robust to the loss of nodes and edges.

2.4 MODEL TESTING

During the early stages of the model creation, data access was restricted, so mock-up data were devised and used as input for the development, formulation and testing of the dynamic social network aiming to represent the design process. The nature of the data developed was a basic structure of individuals and tasks and an active time-window for the edges (to retain active or not the presence function). The actual

data collected from industry are then fed in the system and the case studies in the next Chapter are presented.

A bipartite directed network $\boldsymbol{B} = (U, V, E)$ is employed to model the design process. The two bipartite sets (disjoint sets) represent the two node types i.e. U→design tasks and V→participants. The set of edges E, contains directed edges from one node set to the other and the direction indicates whether an individual is providing or extracting information from a task. In cases where individuals do both, a double edge is included in the model. A visualization of such data is in Figure 2-13, where colours represent the different design teams (collaboration networks). Red represents tasks that relate to the start and the end of the design process, green for the ship design tasks, blue for the performance analysis, yellow for the powering and systems design and cyan for the design monitoring and the cost estimation. Tasks are denoted by letters and square markers, individuals are denoted by numbers and circle markers. For this network the number of tasks is 22, the number of individuals is 57 and the connections are 84. The bipartite network approach

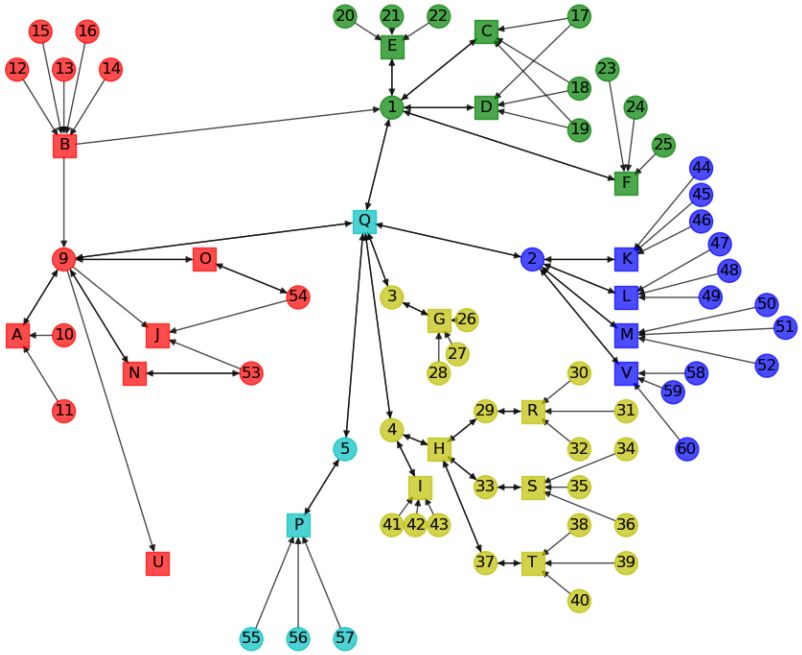

Figure 2-13 Static network for mock-up design process

has been selected, since human actions are a central aspect when analyzing rework. An important note is that any graph $\boldsymbol{G} = (V, E)$ can be seen as a bipartite graph $\boldsymbol{B} = (V, V, E)$, where the links are between the two copies of V. This is used in the coding part of the modelling.

2.4.1 Projections

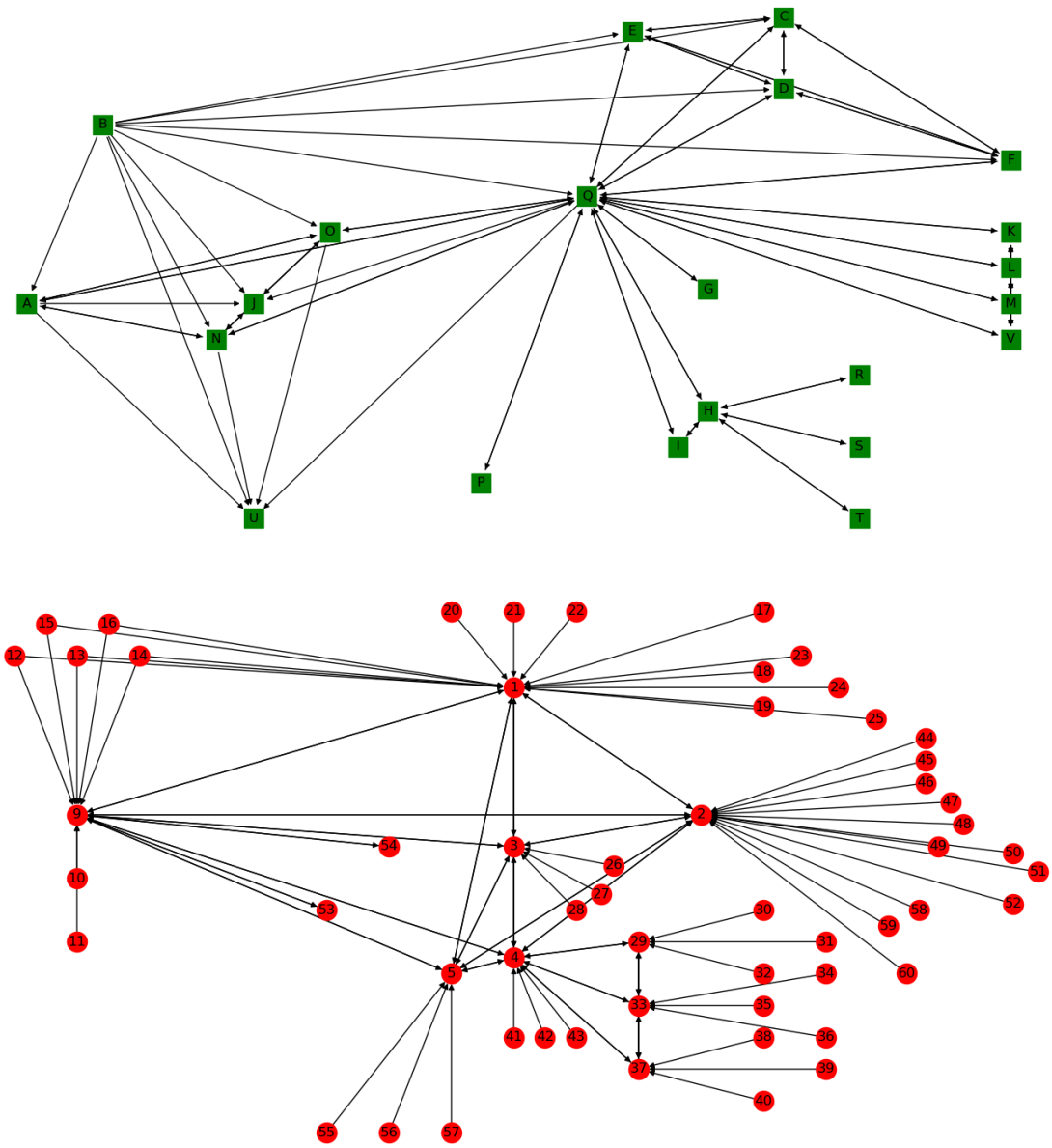

Figure 2-14 U-bipartite projection (up/green) and V-bipartite projection (bottom/red)

Bipartite graph projection is a method for compressing information about bipartite networks. ***U* bipartite projection**: network of nodes in set U, where a pair of nodes is connected if they have a common neighbour in V in the bipartite graph. Similarly, ***V* bipartite projection**: network of nodes in set V, where

a pair of nodes is connected if they have a common neighbour in U in the bipartite graph. Figure 2-14 provides the conversion of the bipartite graph to its two projections. The use of the projection is advantageous for using techniques already developed for classic graphs; however the transformation is one-way. For this reason, all projections are applied to copies of the bipartite graph. Figure 2-14 provides the simplest bipartite projection, i.e., **unweighted projection** and doesn't consider the topology of the network or the frequency of sharing a connection to the elements of the opposing set, as in weighted projection where the frequency of sharing a connection can be added with a thickness distribution on the edges.

A **weighted projected graph** is the projection of the bipartite network ***B*** onto the specified nodes with edge weights proportional to the number of common neighbours between the nodes. Figure 2-15 depicts the weighted graph projections of the mock-up network. Focusing on the U bipartite projection, it is obvious that edges B→Q, O→J and N→J, have the highest weight. Unsurprisingly in the original bipartite network nodes B and Q have two common neighbours i.e. 9 and 1, nodes O and J have 54 and 9, nodes N and J have 53 and 9, while the remaining nodes have only 1 common neighbour. Examining the V bipartite projection, edges 17→1, 18→1 and 19→1 are the edges with the highest number of common neighbours, specifically C and D.

A third bipartite projection, which is particularly useful for this application is the **collaboration weighted graph projection** or Newman's weighted projection. In (Newman, 2001), Newman proposes a measure of collaboration strength based on the numbers of papers co-authored by pairs of scientists and the number of other scientists with whom they have co-authored these papers. Again, a weighted graph is used, but the weights in this case are not proportional to the number of common neighbours. The weights consider the fact that authors that have written many papers together, allegedly, know one another better on average than those that have written few papers together. In a similar fashion, in the ship design process individuals who work together in many tasks know each other better, so the weight assigned on their connecting edge in the projected graph must be greater.

$$w_{ij} = \sum_k \frac{\delta_i^k \delta_j^k}{n_k - 1}, \quad \delta_i^k = \begin{cases} 1, & \exists E(i,k) \\ 0, & \nexists E(i,k) \end{cases} \qquad (2.20)$$

where:

- k represents the tasks
- i, j are individuals
- n_k the number of individuals involved in k
- $E(i,k)$ is the bipartite edge between i and k
- w_{ij} is the weight of the projected edge between i and j

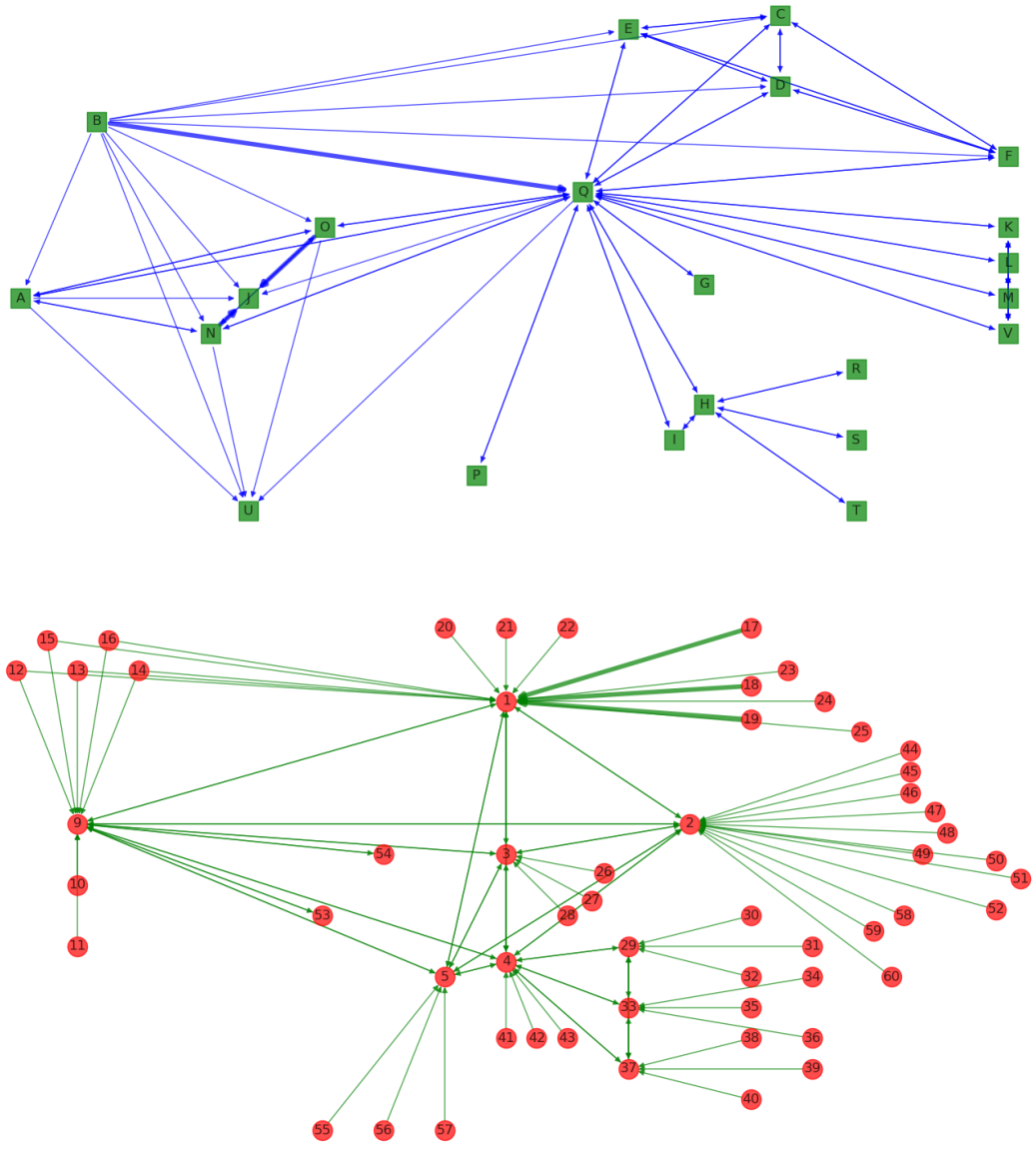

Figure 2-15 Weighted bipartite projections U (up) and V (down). The weight is directly affecting edge width

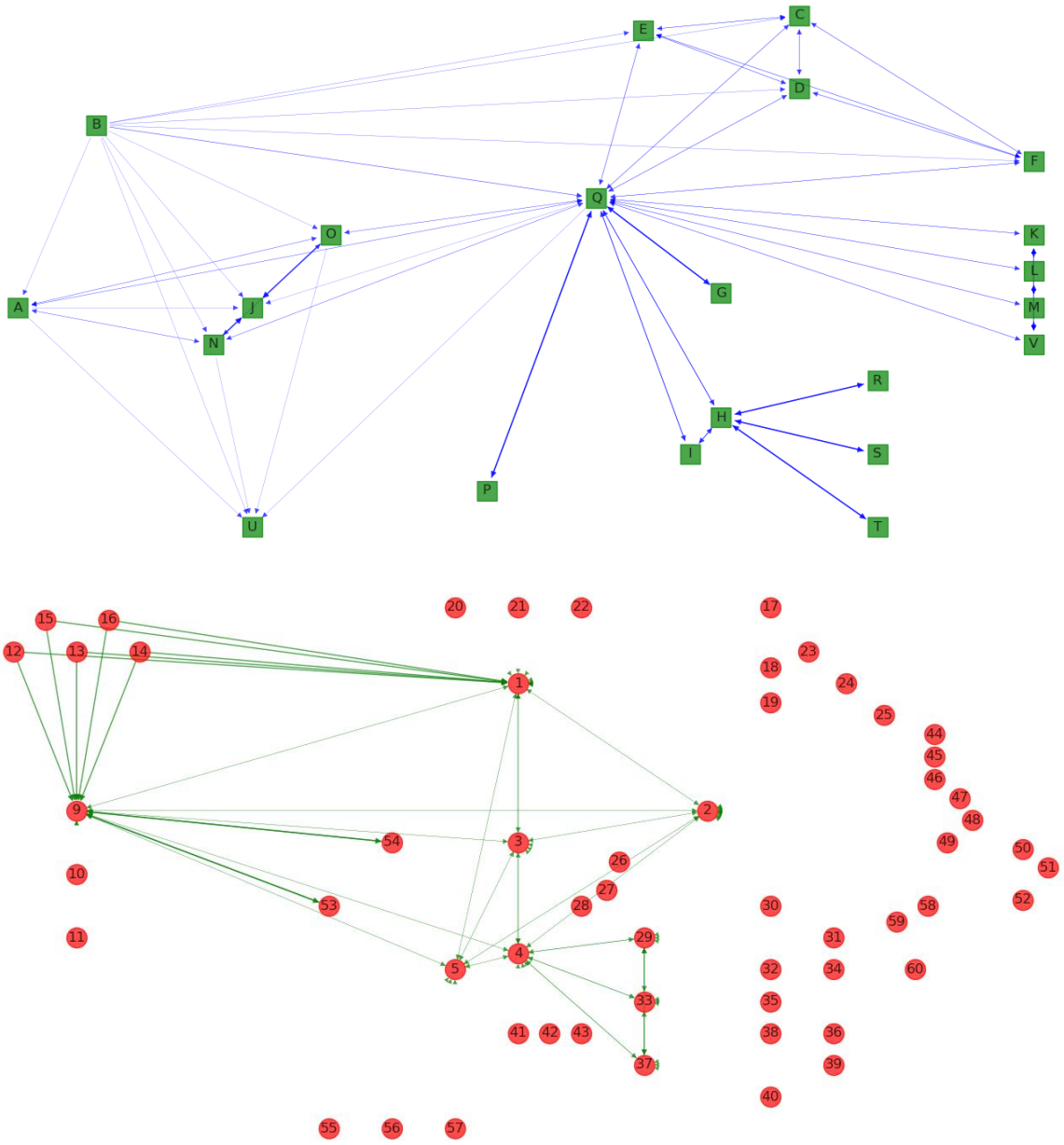

Figure 2-16 Collaboration weighted graph projections, U (up), V (down)

This definition excludes all tasks in which only one individual is involved. In the studied network, there is no such a case. Figure 2-16 shows these two graph projections. In the U-projection the edges with higher weights include H, Q→ and in the directly next level of thickness are N, O, J. It is interesting that these nodes in the bipartite network act as intermediaries between other tasks, e.g., H is needed to connect 4 with R, S and T. This will be revisited when discussing betweenness centrality. In the V-projection a similar behaviour is noticed for 9,1,54,53. However, many nodes that have only one link with a task, e.g.,

21, have zero weight. This is due to the task they relate to, e.g., E: Nodes 21 and 1 have E as a common neighbour but only in the "send signal" direction or **hub** direction, as it is called. In the "receive signal" or **authority** direction E is not a common neighbour of 21 and 1, thus in (2.20) $\delta_E^{21} = 0$ and k takes only value E, so $w_{21,1} = 0$.

2.4.2 Clustering

The next step of the model testing is the evaluation of clustering metrics. To take full advantage of libraries and packages for networks, it is decided to convert the directional graph to undirected. This can be done without loss of directionality data using the edge feature vector. The concept of **edge feature vector** has been used to avoid loss of information regarding the directionality of the edges in the network, that being essential for the developed model. An edge feature vector (EFV) and similarly a **node feature vector** (NFV) include information that may or may not have a direct topological impact. Directionality is a feature that changes the topology and the routing through the network. Some examples of features that do not impact the network structure are the name of the individual or the task, the description of their duty, the years/experience of affiliation within the design team etc. Removing (hiding) the directionality of the network, the result is depicted in Figure 2-17.

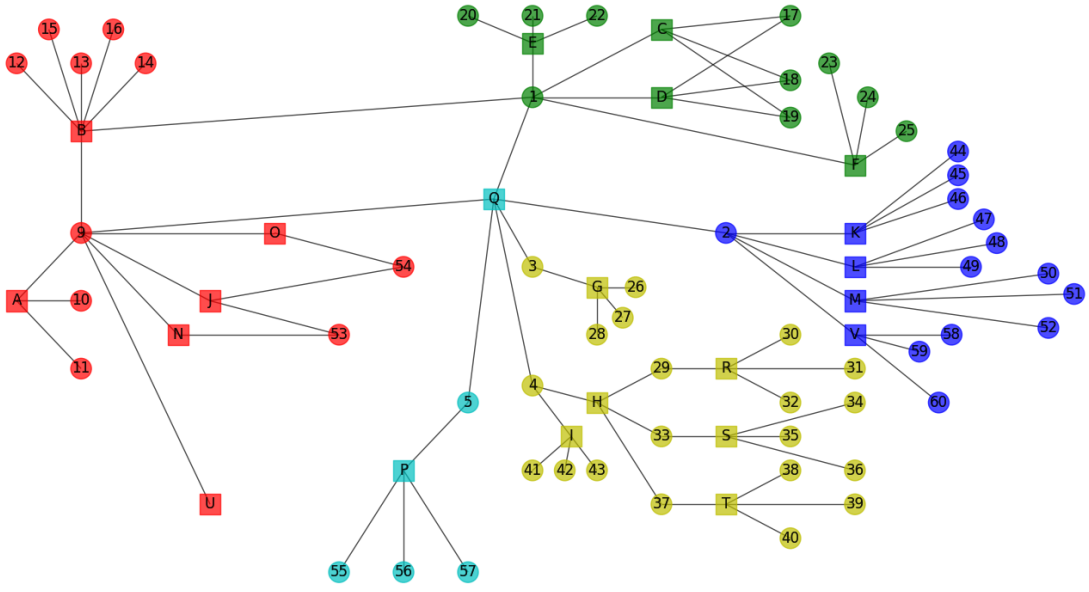

Figure 2-17 Undirected static network

The clustering properties described in section 2.1.1 have been evaluated for the static model of the mock-up data and the results are gathered in Table 2-2. The node degree is also calculated and a degree distribution plot is shown in Figure 2-18. The size of each node is proportional to its degree. The bi-adjacency matrix and the LCC of each node can be directly computed. The bi-adjacency matrix is another interpretation of the graph connectivity, so since the plot of the graph is displayed, the calculation of bi-adjacency matrix is superfluous.

Table 2-2 Clustering metrics for the mock-up data

Property	*Equation*	*Value*
Number of tasks	Set: U	22
Number of individuals	Set: V	57
Number of edges	Set: E	84
Density	$\delta_G = \frac{\lvert E \rvert}{\lvert U \rvert \lvert V \rvert}$	0.066986
Average clustering coefficient	$C(G) = \frac{\sum_{u \in G} c_u}{\lvert U \rvert + \lvert V \rvert}$	0.536316
Average clustering coefficient for set of tasks	$C(U) = \frac{1}{\lvert U \rvert} \sum_{u \in U} c_u$	0.187067
Average clustering coefficient for set of individuals	$C(V) = \frac{1}{\lvert V \rvert} \sum_{u \in V} c_u$	0.671136
Robins-Alexander clustering (transitivity)	$CC_4(G) = 4\frac{C_4}{L_3}$	0.098630

2.4.3 Distance

All distance properties and metrics are implemented on the static network of the mock-up data and added in Table 2-3. An interesting example is $distance(A, U) = 2$. This means that from the start of the network (A represents the first meeting with the ship owner/operator), to the end node of the network (U represents the end of the design phase), the number of edges required to traverse from one to another is 2. This shows one of the limitations of static networks due to the absence of time. Edge (9, U) is activated after all other tasks have been completed, so the distance in this case does not provide any useful information in a static format.

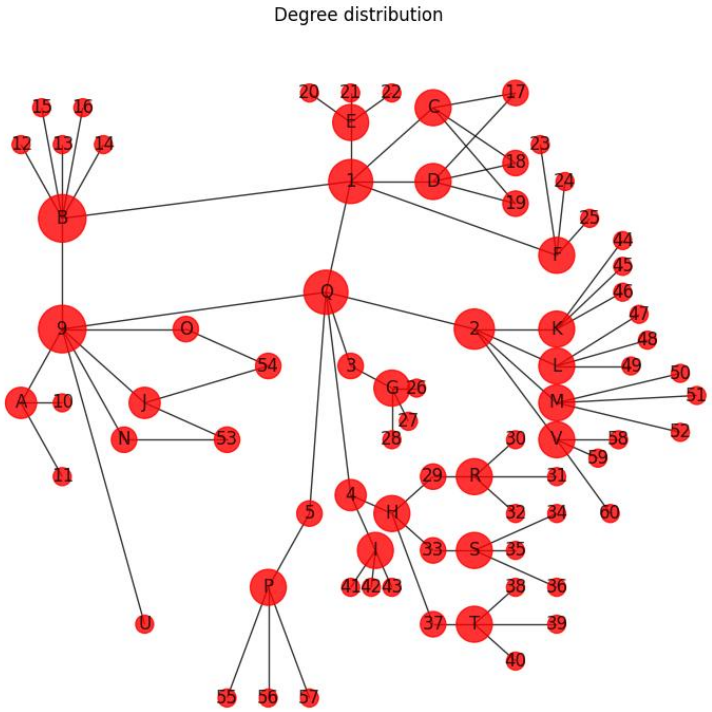

Figure 2-18 Degree distribution plot

Table 2-3 Distance metrics for the mock-up data

Property	*Equation*	*Value*
Average distance	$\bar{d} = \frac{1}{\lvert U \cup V \rvert^2} \sum_{i \in (U \cup V)} \sum_{j \in (U \cup V)} d_{ij}$	5.102240
Diameter	$D(G) = \max(e_i), i \in (U \cup V)$	8
Radius	$R(G) = \min(e_i), i \in (U \cup V)$	4
Periphery	$P(G) = \{i \mid e_i = D(G)\}, i \in (U \cup V)$	{10, 11, 12, 13, 14, 15, 16, 54, 53, 55, 56, 57, 20, 21, 22, 17, 18, 19, 23, 24, 25, 44, 45, 46,

		47, 48, 49, 50, 51, 52, 58, 59, 60, 26, 27, 28, 30, 31, 32, 34, 35, 36, 38, 39, 40}
Center	$K(G) = \{i \mid e_i = R(G)\}, i \in (U \cup V)$	{4}
Number of nodes in the periphery	$\lvert P(G) \rvert$	45
Number of nodes in the center	$\lvert K(G) \rvert$	1

An interesting observation is that the periphery set comprises of a large number of nodes, while the center is only 1 node. This is due to the nature of the static graph. The edges which are at the end of tree formations, such as 50, 51, 52 have distance 8 with nodes 30, 31, 32 and vice versa. In terms of distance, the most central node is node 4 (center), because it requires only 4-edge traversal (hops) to connect with other nodes. The number of nodes belonging on each set seems to be related with the topology of the graph (tree formations and/or circles), so it is of interest to track it and see how it behaves in the dynamic scheme.

From Figure 2-18, a node that could be of interest in terms of how central it appears from its degree distribution is node Q (design process monitoring). Applying the BFS algorithm on Q, returns Figure 2-19 as a directed graph starting from Q. The maximum path length of Q in the network is 5 (distance from nodes 30, 38, 31, 32, 36, 39, 34, 35, 40).

The developed code has the capacity to identify connected components and creates **subgraphs** on these connected subsets. The majority of readily available (python libraries), or developed in the context of this thesis, codes work only for connected graphs. On account of this, the creation and retaining of connected subgraphs and the temporary - omission of the disconnected components are imperative. For the developed network both edge and node connectivity are 1, i.e., a removal of just one node or edge leads to a disconnected graph.

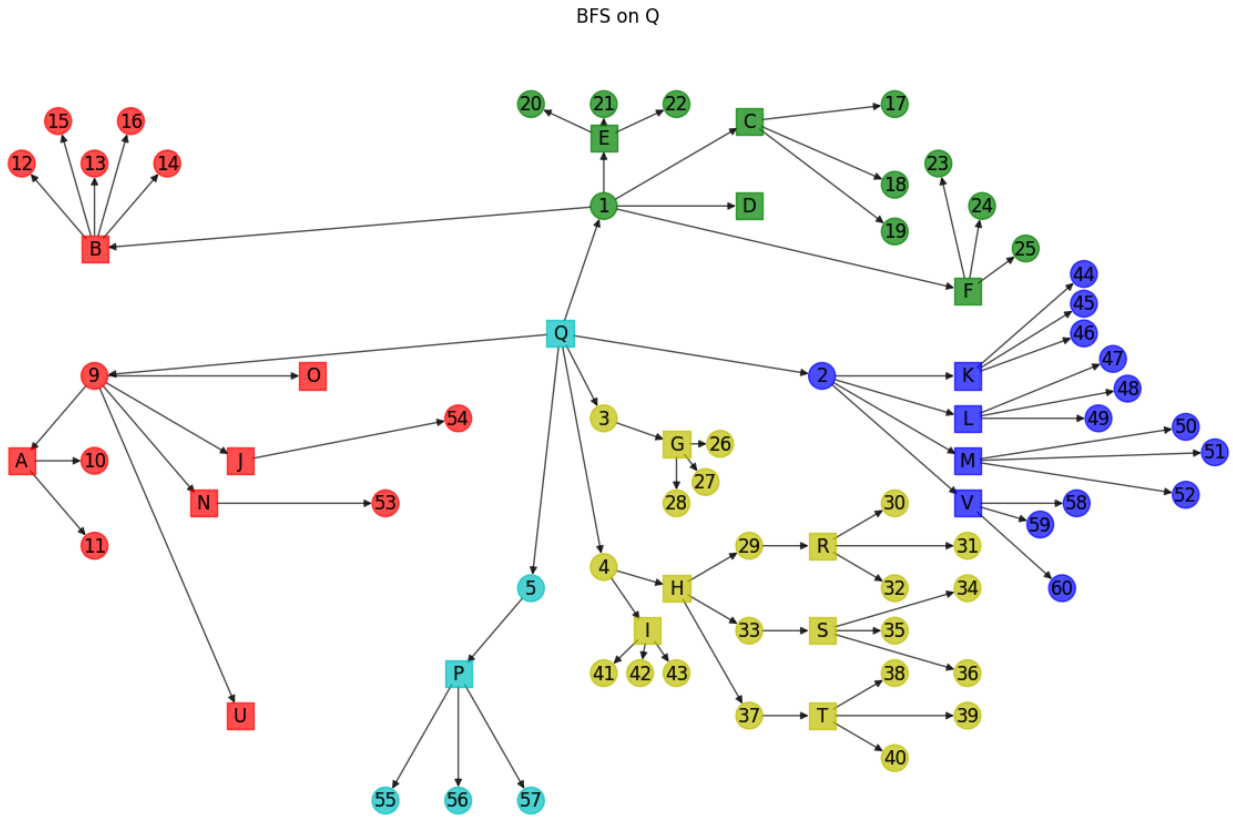

Figure 2-19 BFS algorithm tree applied on node Q

2.4.4 Dynamic graph

The coding process for the creation of the model starts by creating a copy of the static graph and then at time $t = 0$, the presence function is 0 for all edges. The time intervals for which the presence function should be 1 are stored in the EFV, allowing for activation and deactivation of the presence function as time evolves based on each edge's stored time interval. The modelling allows for more than one interval, allowing for activation, deactivation, re-activation and re-deactivation of an edge. The original static graph is retained in order to maintain all static model information that could be useful in the future and for comparative analysis.

The code allows for discretization of the design time in days, weeks or months depending on the user requirements. At each time instance (day, week, month) the projections can be obtained as in the static graph, however this can be done by the user if a particular time instance is important. In that sense, the bipartite projections do not have to be tracked over the whole time spectrum. The same holds true for the

bi-adjacency matrix at each moment in time. The first measures of the network which can be evaluated are the number of active nodes (U, V) and edges (E) and their evolution in time as in Figure 2-20.

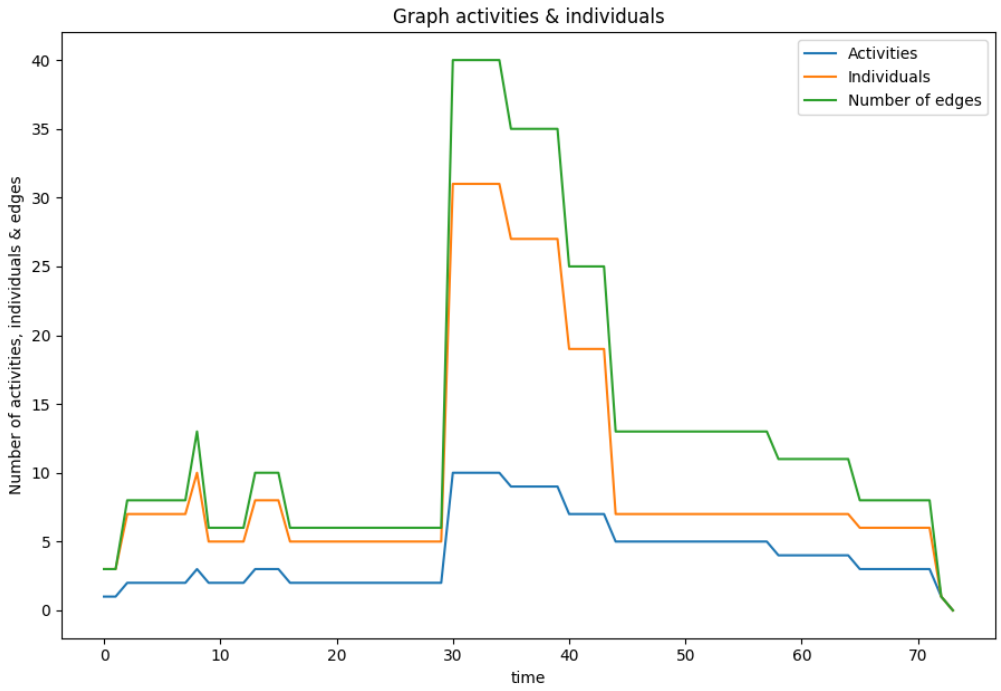

Figure 2-20 Evolution of number of activities, individuals and connections

Clearly at time 30 all 3 measures peak. Examining the network at time instances 25 and 32 (Figure 2-22), verifies this increase in all active nodes and existing edges. The next property that gives some insight on the connectivity of the network is the evolution of density (Figure 2-21). The same behaviour is discovered again, i.e., the density peaks at time 30. Comparing this peak (0.032) with the static graph density (0.067), a high, but expected, discrepancy between the dynamic model and the static model is noticed. Actually, the dynamic graph peak density is less than half the static graph network. Through the density equation, half the density means that the number of edges is half, since the number of nodes is the same. Comparing the number edges between static model (84) and peak dynamic values (40), this is further established.

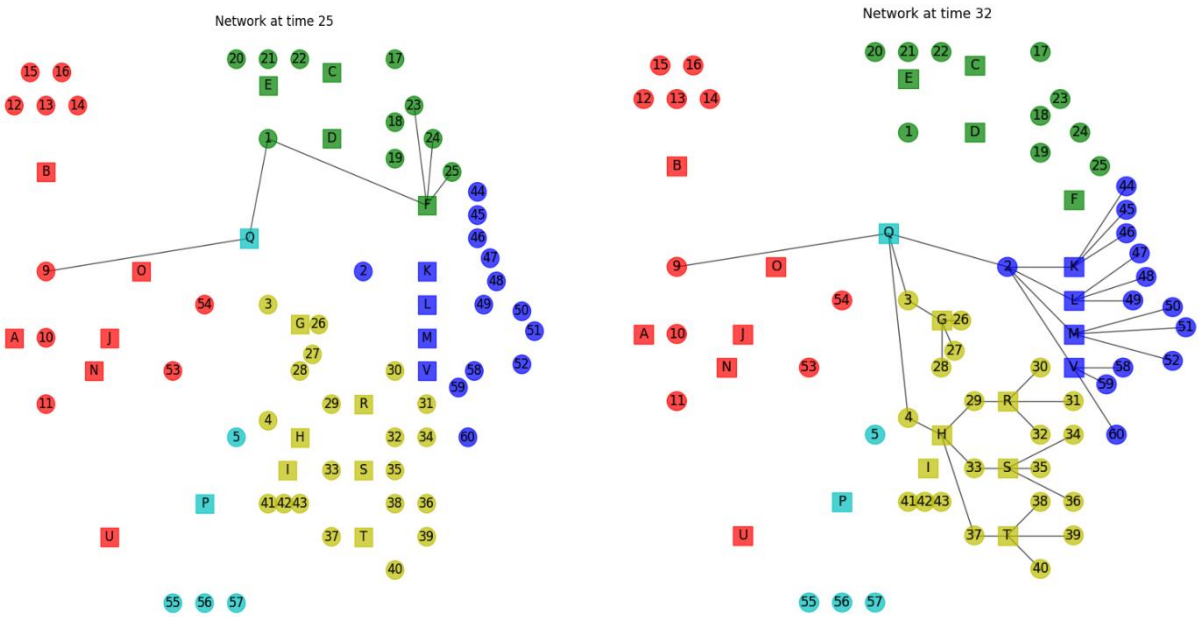

Figure 2-22 Network comparison at time 25 (left) and 32 (right)

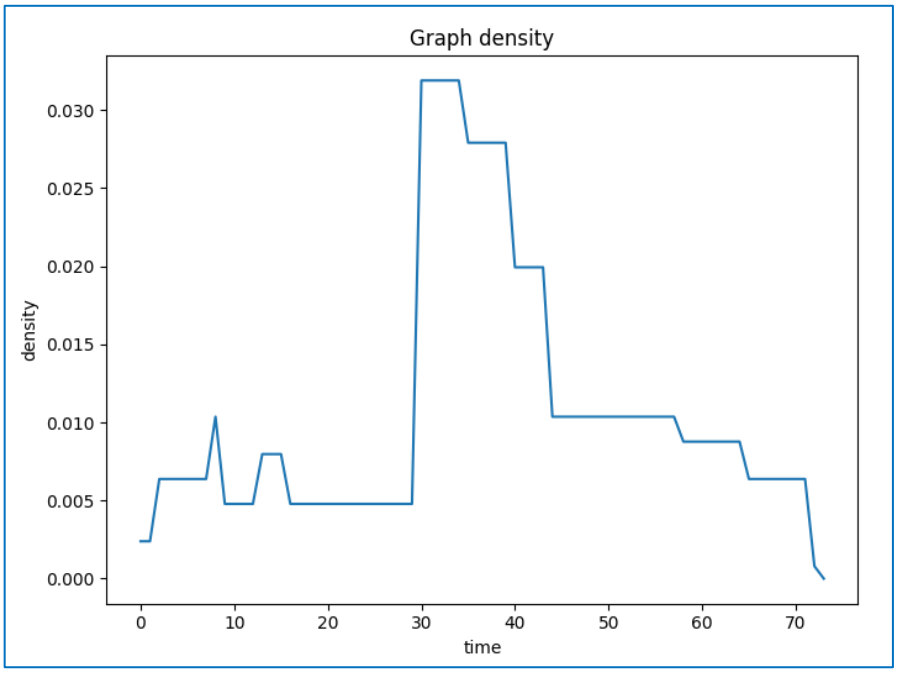

Figure 2-21 Density evolution

The code has the capacity to track the local clustering coefficient (Latapy method) as in (2.2) for each node and provide the corresponding plot. Figure 2-23 shows all global clustering metrics.

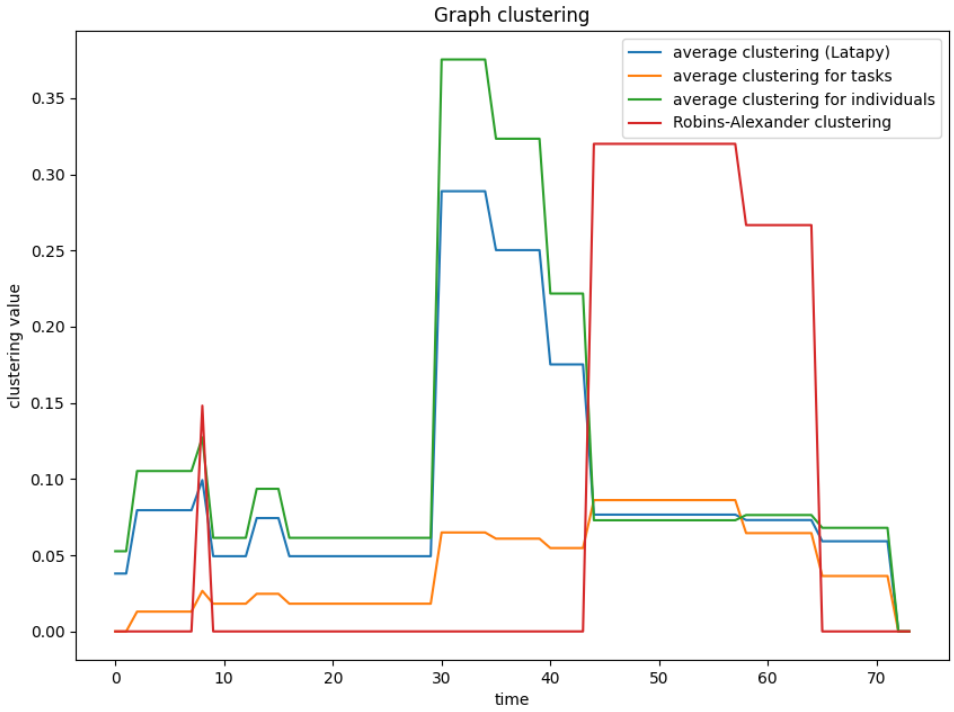

Figure 2-23 Evolution of clustering coefficients in time

The same behaviour as before can be observed, which is to be expected. However, the Robins-Alexander clustering coefficient is not in alignment with the remaining metrics, hence a more careful examination is required and a comparison between times 32 and 50 (Figure 2-24) could provide some insight. First, let's examine Latapy clustering. The clustering coefficient for pairs of nodes is from equation (2.2):

$$c_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}, \quad (u, v) \in U \; or \; V$$

For example, if u=30 and v=32, then $|N(30) \cap N(32)| = |\{R\}|$. However, R is the whole neighbourhood of u and v, i.e., $|N(30) \cup N(32)| = |\{R\}|$, so $c_{30,32} = 1$. This is the case for all pairs of nodes that follow this tree-formation, and this is the reason for the high average clustering coefficient at this instance. Notice that at the time-interval 30-35, this tree-formation is more dominant than ever in the network topology, so high Latapy clustering is expected.

On the other hand, examining the Robins-Alexander clustering from equation (2.5):

$$CC_4(G) = 4\frac{C_4}{L_3}$$

at time instance 32, there is no C_4 formation in the network, so the expected clustering is 0 and this is depicted in the clustering plot. But when examining time instance 50, two paths (9→O→54→J→9, 9→J→53→N→9) exist. As a matter of fact, 8 paths exist, i.e., 9→O→54→J→9 is the same circle with O→54→J→9→O and 54→J→9→O→54 etc., but this is taken into account by the coefficient 4 in the formula. The tree-formation is limited in this case (I, 41, 42, 43), so the Robins-Alexander clustering is dominant in this time-instance. This behaviour reinforces the initial assumption that these two methods of clustering should be used complementary to each other.

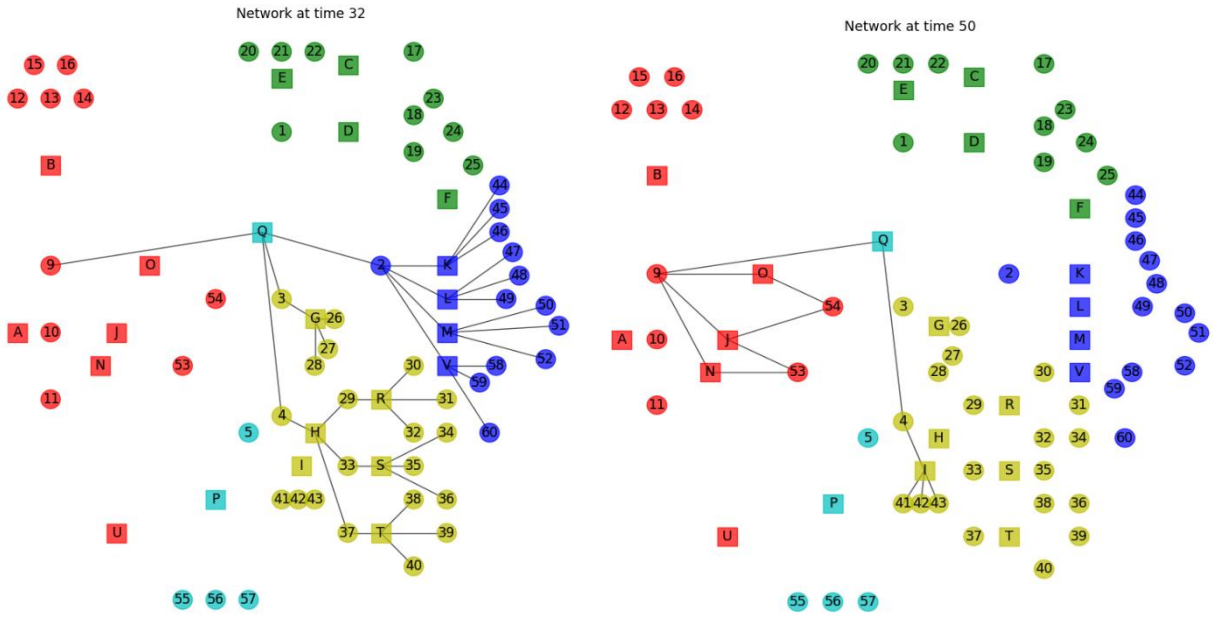

Figure 2-24 Network comparison at time 32 (left) and time 50 (right)

A particular time-instance of interest is at $t = 8$ (Figure 2-25). Here both tree-formations and a circle are formed. From the transitivity value one can deduce that a circle is more dominant than the tree-formations hence leading in greater Robins-Alexander clustering than Latapy clustering. The answer lies in the denominator of transitivity. There exist few paths of length 3 in this network (e.g. 9→B→1→Q, which will lead to a circle C_4 if it connects back to 9, and 9→B→1→C, which remains an L_3). One could assert then that the Robins-Alexander clustering is more sensitive to the addition of one circle C_4, than the Latapy clustering to the addition of a tree component.

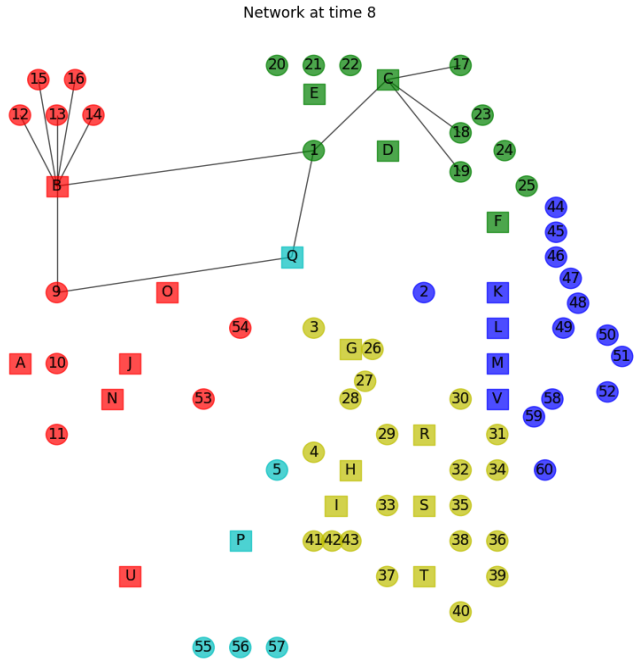

Figure 2-25 Network at time 8

The next step of the testing process includes the evaluation of distance measures. All distance measures, being local (node-wise evaluations) or global, can be evaluated dynamically. Starting from the plot for graph diameter, radius and average distance (Figure 2-26) everything is expected and the behaviour of all three measures is increasing in sections where the graph "stretches" (e.g. time instance 29) and is decreasing in parts where the number of participants and activities is significantly reduced (e.g. time instance 72). For the static model, the diameter is 8 and the radius is 4. This behaviour (diameter being double the radius) is followed in the dynamic representation as well.

Figure 2-27, however, provides valuable insight for the dynamic model. The center of the graph comprises, in most of the project's duration, a single node. There are only two notable exceptions, time interval [2,7] and time instance 72. On the other hand, the periphery of the graph, i.e., nodes that have eccentricity equal to the diameter, are significantly more and reaching their peak, when the number of edges and participants is the largest.

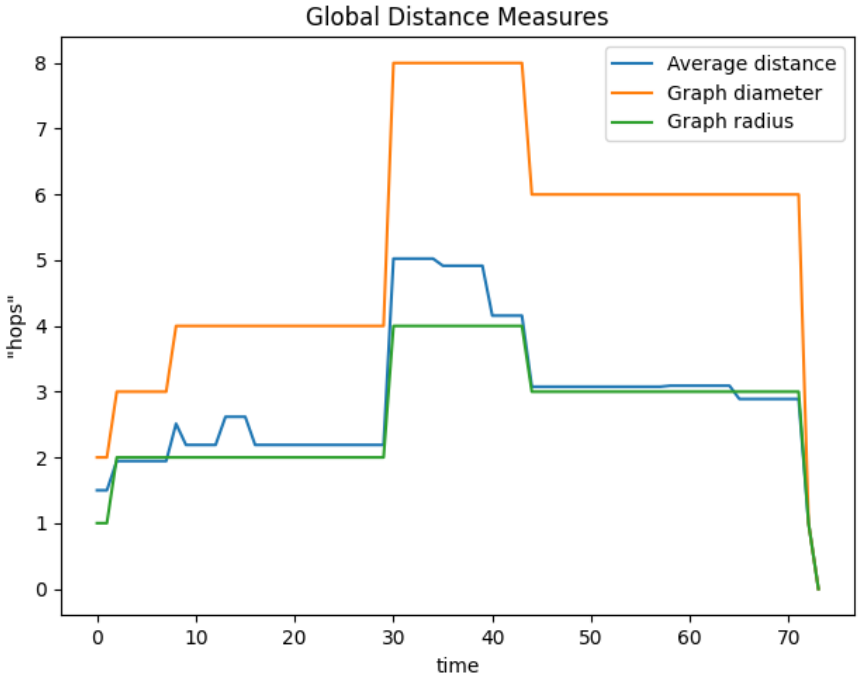

Figure 2-26 Evolution of average distance, diameter and radius

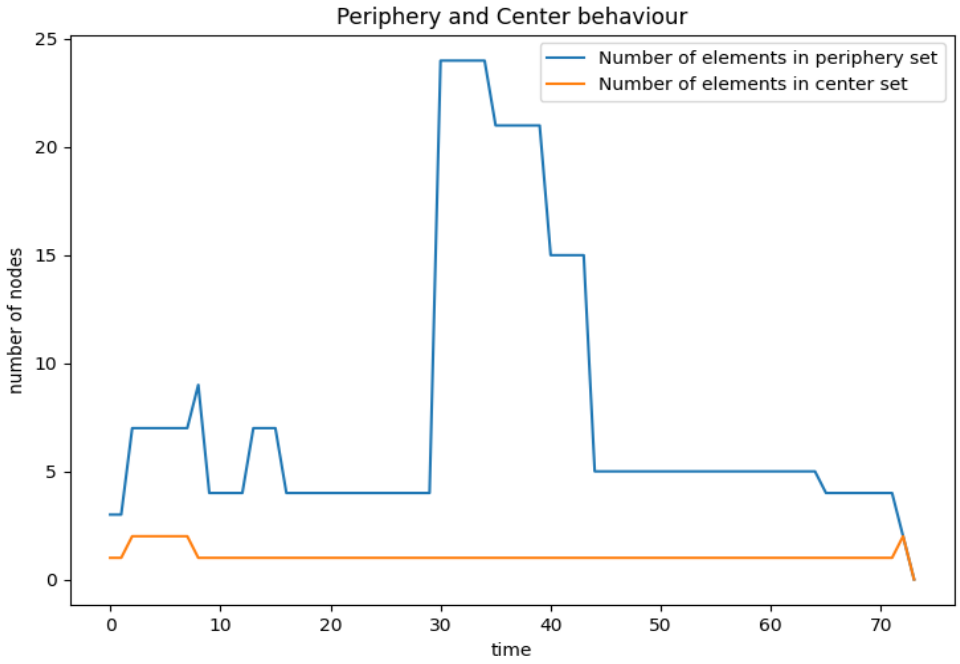

Figure 2-27 Variation of periphery and center population in time

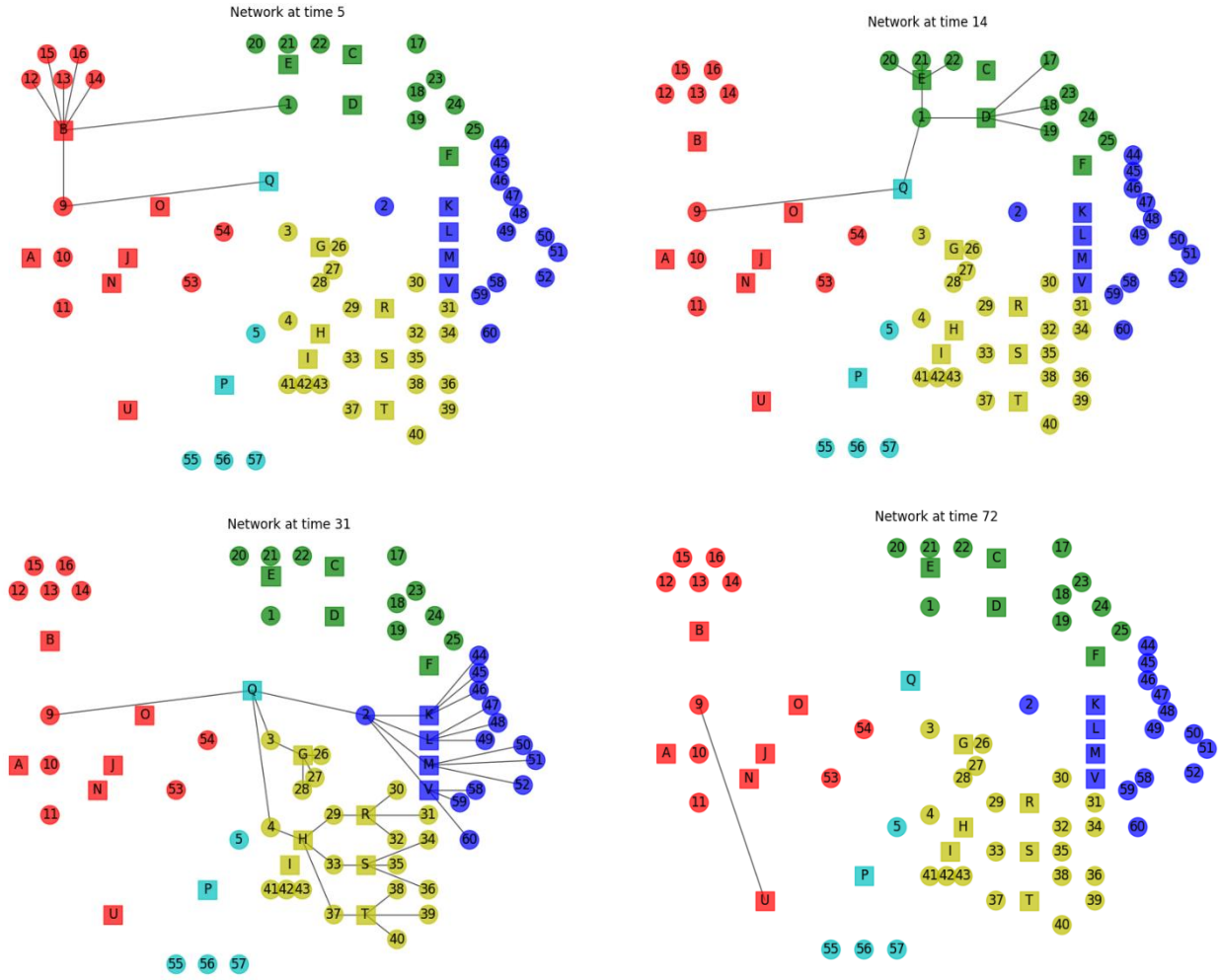

Figure 2-28 Network representation at time instances 5, 14, 31, 72

Examining these time intervals (Figure 2-28), enables a better comprehension of the network behaviour in terms of center and periphery. At time instance 5, the radius is 2 and the nodes that have this value of eccentricity are two K(G)={B,9}, so they form the graph center. The paths that lead to minimum eccentricity (radius) two are 9→B→1 (or 9→B→12,13 etc.) and B→9→Q. The periphery of the graph is set P(G)={1,12,13,14,15,16,Q}, so the number of the vertices forming the periphery set is 7. In this time-instance all nodes belong to either the center or the periphery sets.

At time instance 14, the radius is 2 and only a single node comprises the center: K(G)={1}. The diameter is 4 and 7 nodes form the periphery set: P(G)={9,18,17,19,20,21,22}. In this case, the center is a single node, the periphery is a set of 7 nodes and nodes {Q,E,D} are in between the periphery and center, i.e., $R(G) < e_i < D(G),\ \ i \in \sim\big(K(G) \cup P(G)\big)$.

At time instance 31, the radius is 4 and the center is K(G)={4}. The diameter is 8 and the periphery comprises of 24 nodes: P(G)={44,45,46,47,48,49,50,51,52,58,59,60,26,27,28,30,31,32,34,35,36,38, 39,40}. So out of the 41 active nodes in this time instance, 24 form the periphery and 1 forms the center of the graph.

At the last time instance, time 72, there is a degeneration of the graph to a single edge connecting two vertices. In this case the periphery and center sets coincide, P(G)=K(G)={U,9}, and the radius is equal to the diameter: D(G)=R(G)=1.

More details on the mock-up data are given in Appendix A.

3 CASE STUDIES

This Chapter includes details on the data used for the cases studies (Section 3.1) and application of the developed model on the three case study ship designs (Sections 3.2, 3.3 and 3.4). Then, Section 3.5 discusses the results obtained from various perspectives, in terms of planned and actual task completion, learning variations across the 3 designs, centrality metrics and rework metrics.

3.1 DATA DETAILS & CONFIDENTIALITY

The data used for the systematic experimentation and validation of the model are indicative data for the design of a modern surface combatant. They include activities that cover a period of 11 years (2017-2028). The activities/tasks are all assigned a codename (e.g., Activity 2PSS170M) to avoid displaying sensitive information. In the same manner, the people/individuals are assigned a number (e.g., Individual 98) to avoid personal data breach.

The data were provided after the model had been completed using the mock-up data (as described in Chapter 2). Since the size of the dataset became apparent, restructuring on the code (python) took place to resolve scaling issues when increasing the dataset size by 2 orders of magnitude. Thus an iteration of the model development took place at that time. The data provided were in .csv files, which were converted in Pandas Dataframes for manipulation in Python. The model was re-tuned to work with DataFrames. After a necessary data wrapping/cleaning step the number of activities and individuals for the three ships are gathered in Table 3-1. The first major conclusion can already be drawn from the size of the data. The creation of the first design included many tasks that were not repeated for the second and third ship designs. Despite the size difference of the tasks, the people involved in the design are always of the same order across all ships. Because of the high discrepancy of activities between Ship A and ships 2 and 3, all metrics are normalized (mainly rework metrics, because the structural properties are normalized by their definition).

Table 3-1 Approximate number of people and activities per ship

	SHIP A	SHIP B	SHIP C
Activities	12800	2000	2100

People	100	100	100
Edges (static format)	12700	2000	2000

The dataset includes information on the start and completion dates of each task, the planned start and completion dates of each task (temporal quantification of rework). The planned labour units and the actual labour units (at task completion) are included in the dataset and are used to quantify rework in terms of cost. Labour units[4] are a type of measurement of the personnel and resources (cost & equipment) committed to a task. Other data included for tasks are activity name, activity ID, Project/subproject ID and people involved in them. Data regarding individuals are name and specified code. The data were available in June 2022, therefore the period from mid-2022 to 2024 is treated as future for the data analysis process. Another observation, arising after the data collection, is that the reported individuals are not necessarily the same ones across the project's duration. The high discrepancy between Ship A and ships 2 and 3, is attributed to the concurrent design of all 3. The design tasks performed at the beginning of the project are activities that affect all 3 ships but the company allocates them only on Ship A, thus the different order of magnitude on the activities of Ship A and the two other case studies.

Information for nodes and edges is retained in respective feature vectors. For example, a NFV and an EFV are defined as:

- Node: ['Assigned number', 'Node name', 'Individual personal information (position, status in company etc.)']
- Edge: ['Codename', 'Activity description', 'Planned labour units', 'Labour units at completion', 'Planned start', 'Planned end', 'Start', 'End', 'Task operator', 'Edge directionality']

From the available data an undirected bipartite static network is created, the directionality (single or dual) of information flow being retained within the EFVs. The static network is decomposed into dynamic time-dependent networks, based on the desired time step. The most refined decomposition is in days (available in the naval ship design indicative data); however, all plots are created using weeks as the time unit. The user of the software can choose the analysis period by specifying the limits of the time analysis. For

[4] The detailed definition for labour units cannot be disclosed for confidentiality purposes. It can be considered as a more sophisticated man-hours measurement for the purposes of this thesis.

example, one might be interested only in year 3 of the design process (2020), thus they have the option to isolate this period.

In this Chapter the creation of the plots is based on whole design duration and the dynamic graphs are created weekly, but there are cases where a more local (in terms of time) analysis is required, thus the start and end date are modified, and a daily check is chosen. Setting the period for the whole design duration (2017-2028) with a daily step is the most processing-heavy option, but it runs relatively quickly on a standard PC (processor: AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx - 2.30 GHz, RAM: 16.0 GB), with longest results in 4 (mins) when creating the Ship A (heaviest) dynamic network. Initial times were about 15 (mins) for the first ship when using the original model developed in Chapter 2. Due to the size of the dataset for Ship A, some restructuring of the code (Python) and de-cluttering of the code workflow was required, leading in significantly reduced analysis time (4 mins).

The rest of this chapter displays the case studies, giving for each ship some basic figures to demonstrate the complexity of the static approach, then plots of the activities and individuals evolution across the project duration, plots of density and clustering coefficient across the project duration and a snapshot of the bipartite graph instance with the highest traffic (sum of activities and individuals) for each case. For each ship, the process is repeated twice, one for the projected data from the industrial data and one for the actual (work done so far). In the discussion section, all metrics are gathered and observations from planned vs actual design are made, the impact of Ship A to the subsequent case studies is discussed and highest-centrality and rework nodes in the network are identified.

3.2 SHIP A

3.2.1 Planned design

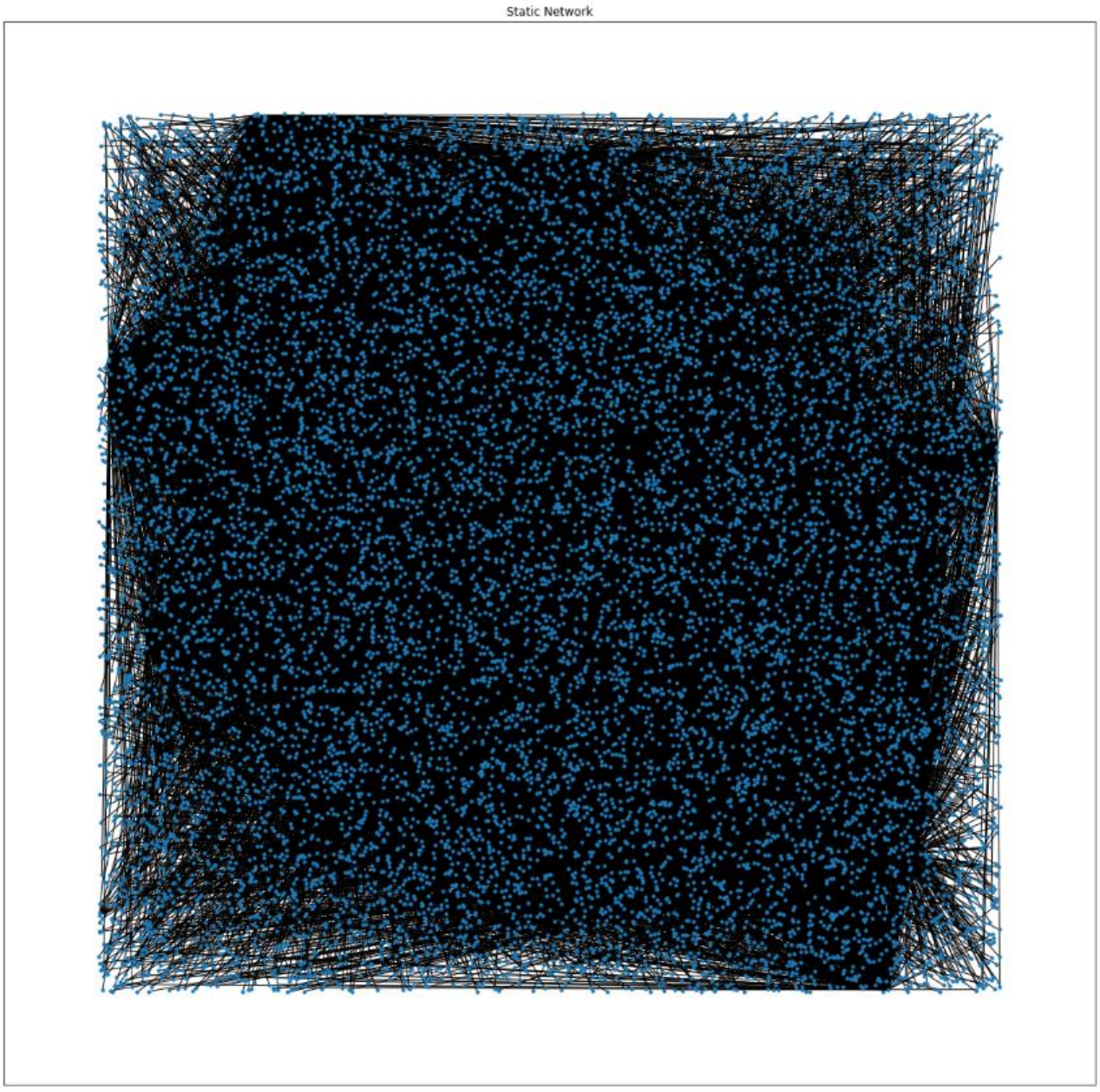

Figure 3-1 Static network for Ship A (planned design)

In this section the presence function of the edges is activated depending on the dates that the tasks are supposed to take place as set before the start of the design. Figure 3-1 depicts the static network of ~12500 edges between the nodes, and already a basic limitation of the static graph approach is obvious. This

visualisation does not provide much insight into the design process apart from showing how dense it is. Even the visualisation techniques developed for this thesis and described in Section 4, are not able to help with this view. A dynamic graph visualisation with a handy user interface (UI) certainly will help resolve problems caused by views like Figure 3-1. All metrics calculated for the static graphs are gathered in Table 3-2.

The density of the static graph is 0.01 (dense graph), Latapy clustering is 0.9922 (highly clustered) and transitivity is 0 (explained from the graph topology). Comparing the values of density and clustering, as well as the numbers of concurrent tasks and individuals involved, it is apparent that the dynamic graph already provides a better understanding of the design process. For example, at any time the tasks are less than 2000 and the individuals are at most 13.

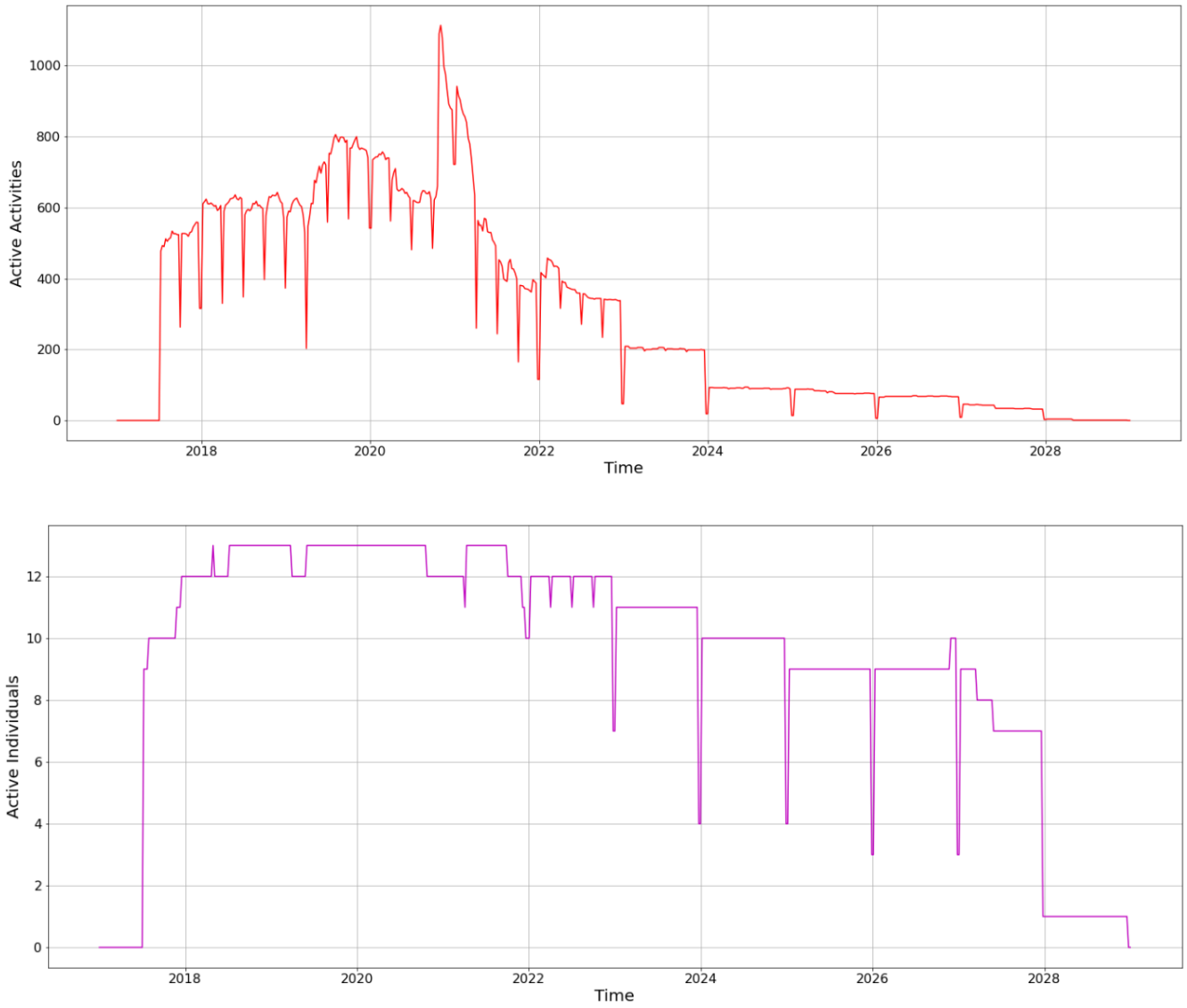

Figure 3-2 Activities and Individuals for Ship A (planned design)

It would be interesting to see what is happening during 2020-2022 period, because as depicted in Figure 3-2 and Figure 3-3, there is a recession and then suddenly an increase in activity (global maximum) and then another sudden recession. The peak activity is displayed in Figure 3-5.

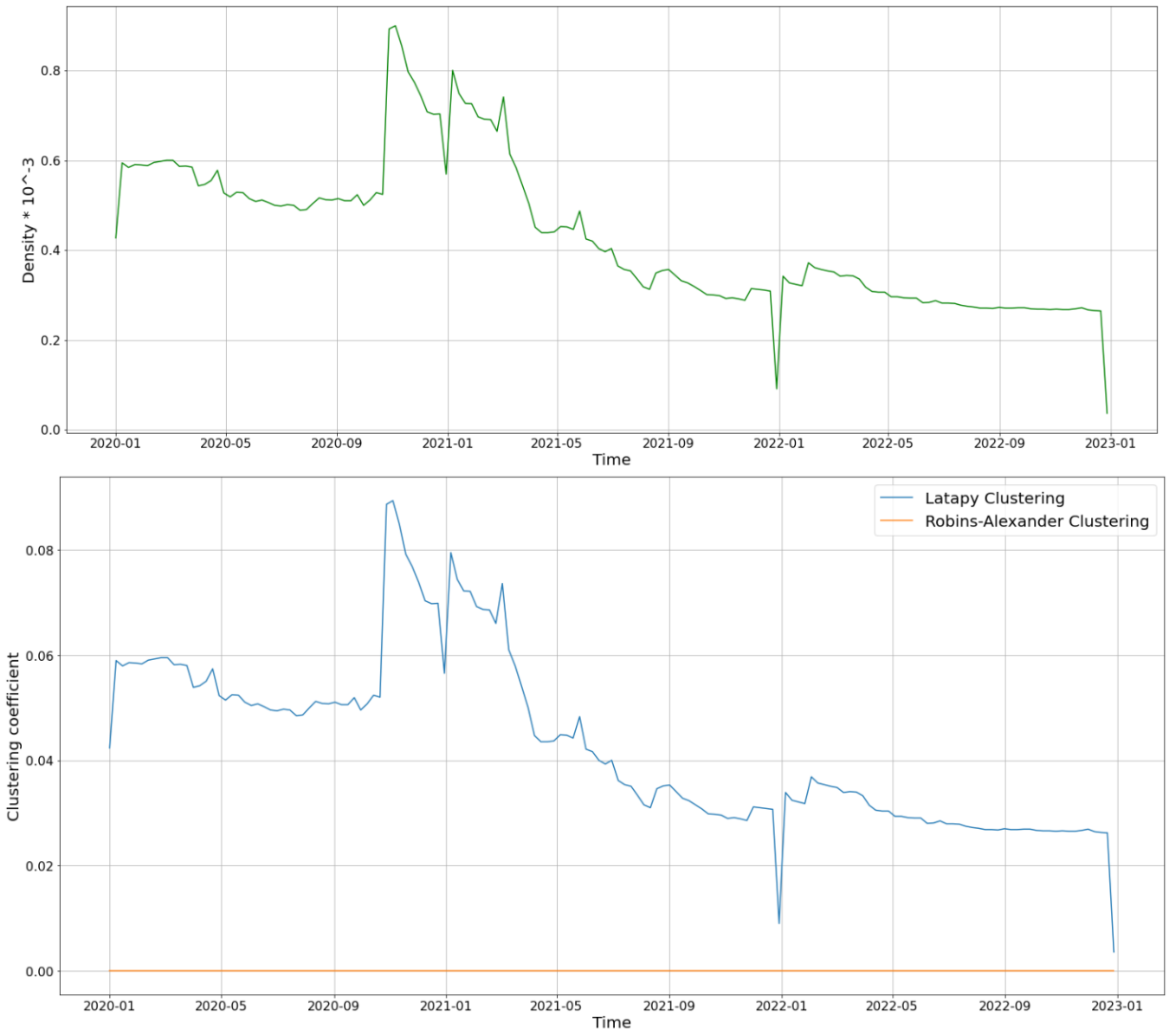

Figure 3-3 Density and clustering for Ship A (planned design 2020-2022)

Figure 3-4 Activities and individuals for Ship A (planned design 2022-2024)

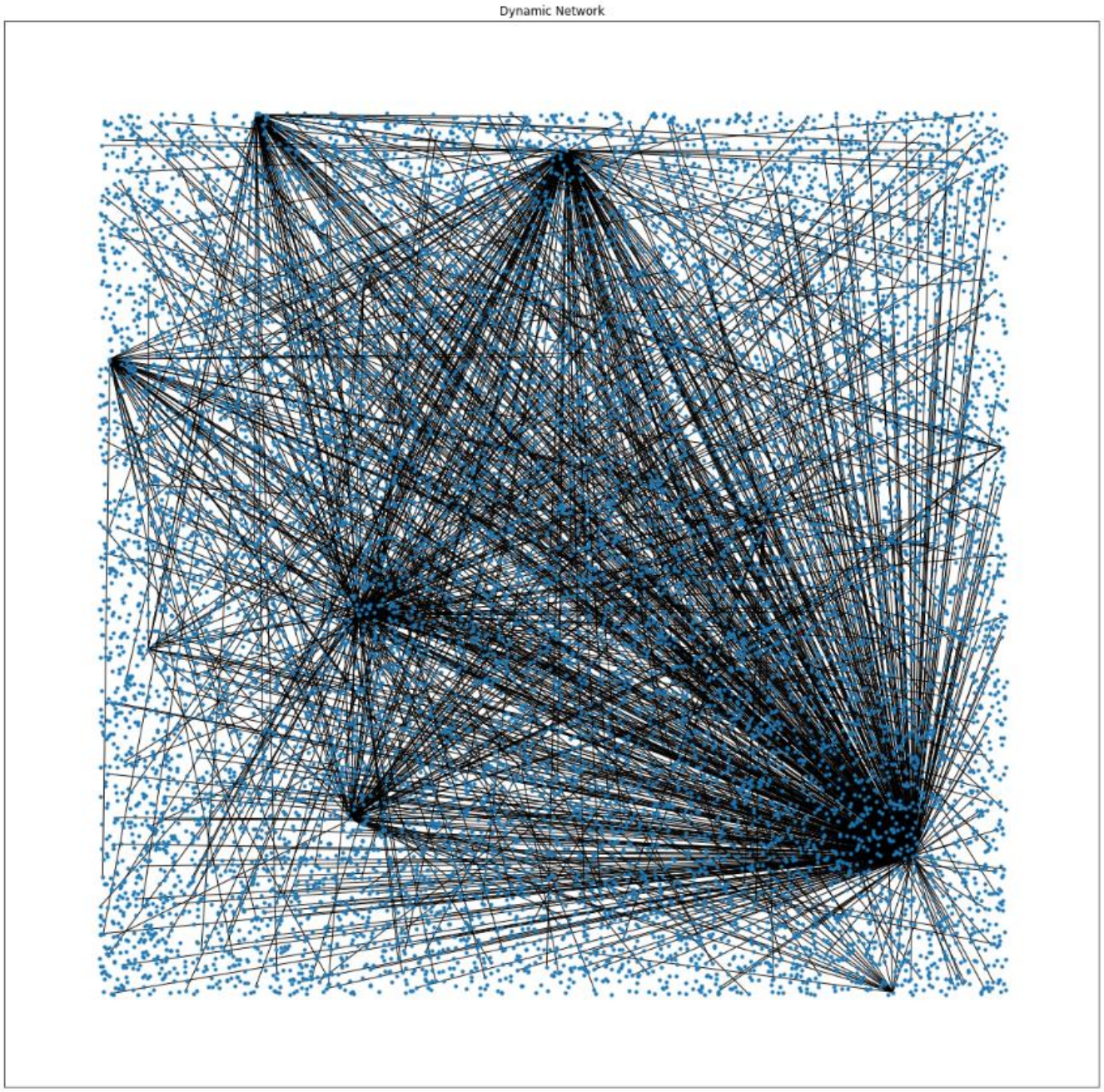

Figure 3-5 Network snapshot at 01/11/2020 for Ship A (highest activity – planned design)

An outcome from the TDG snapshot at date: 01/11/2020 (Figure 3-5), is that the TDG allows for highly clustered areas to be identified and the user using the proposed visualisation techniques (Section 4) can identify visually the more central individuals for the design pipeline (based on degree centrality).

3.2.2 Actual design

The values for the static graph are similar to those of the planned design case. The dynamic graph plots (Figure 3-6 and Figure 3-7) display a more normalised behaviour in terms of how many rapid recessions appear. The 2020-2022 period is displayed in Figure 3-8 and Figure 3-9, while the peak activity network is shown in Figure 3-10.

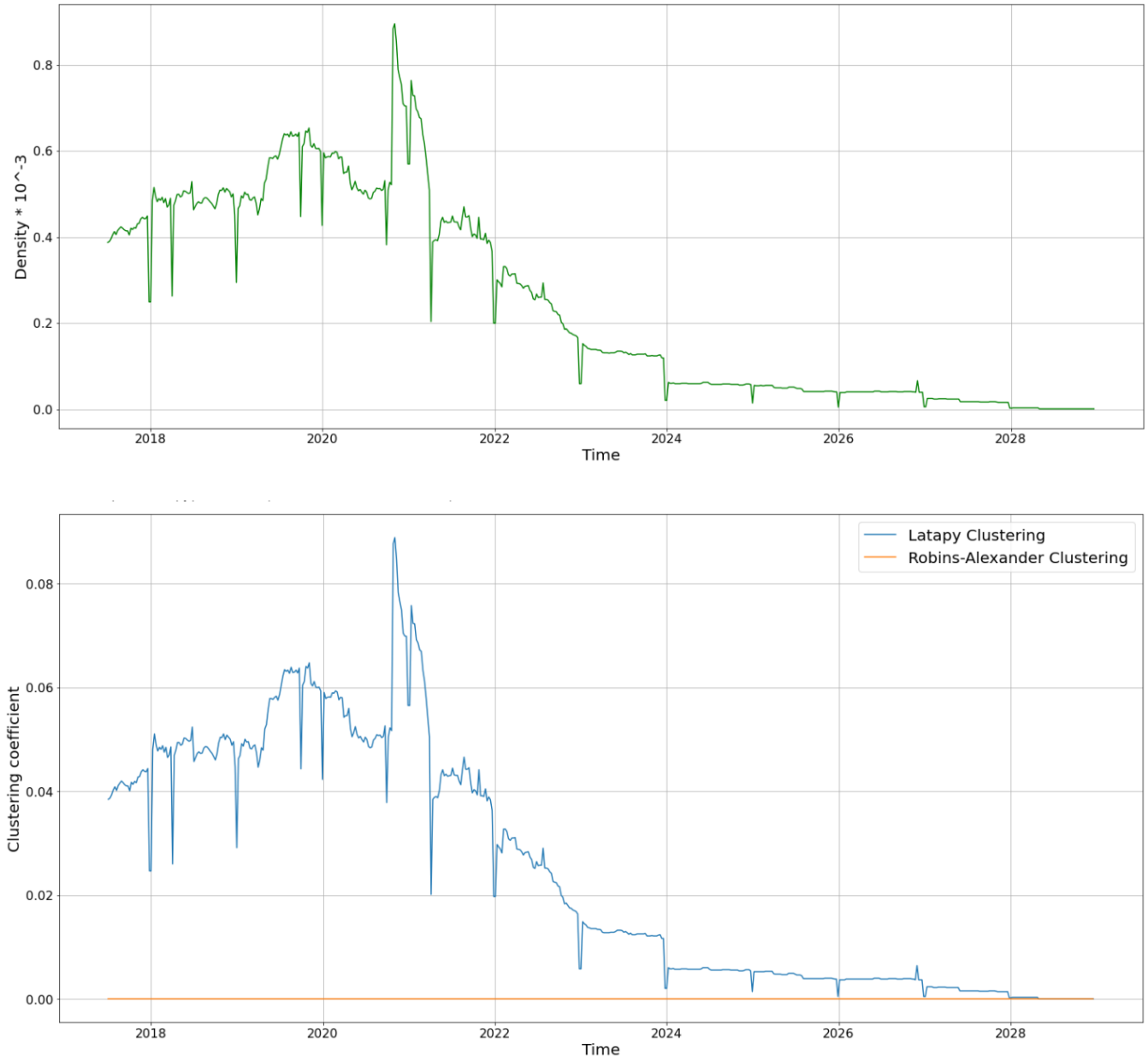

Figure 3-6 Density and clustering for Ship A (actual design)

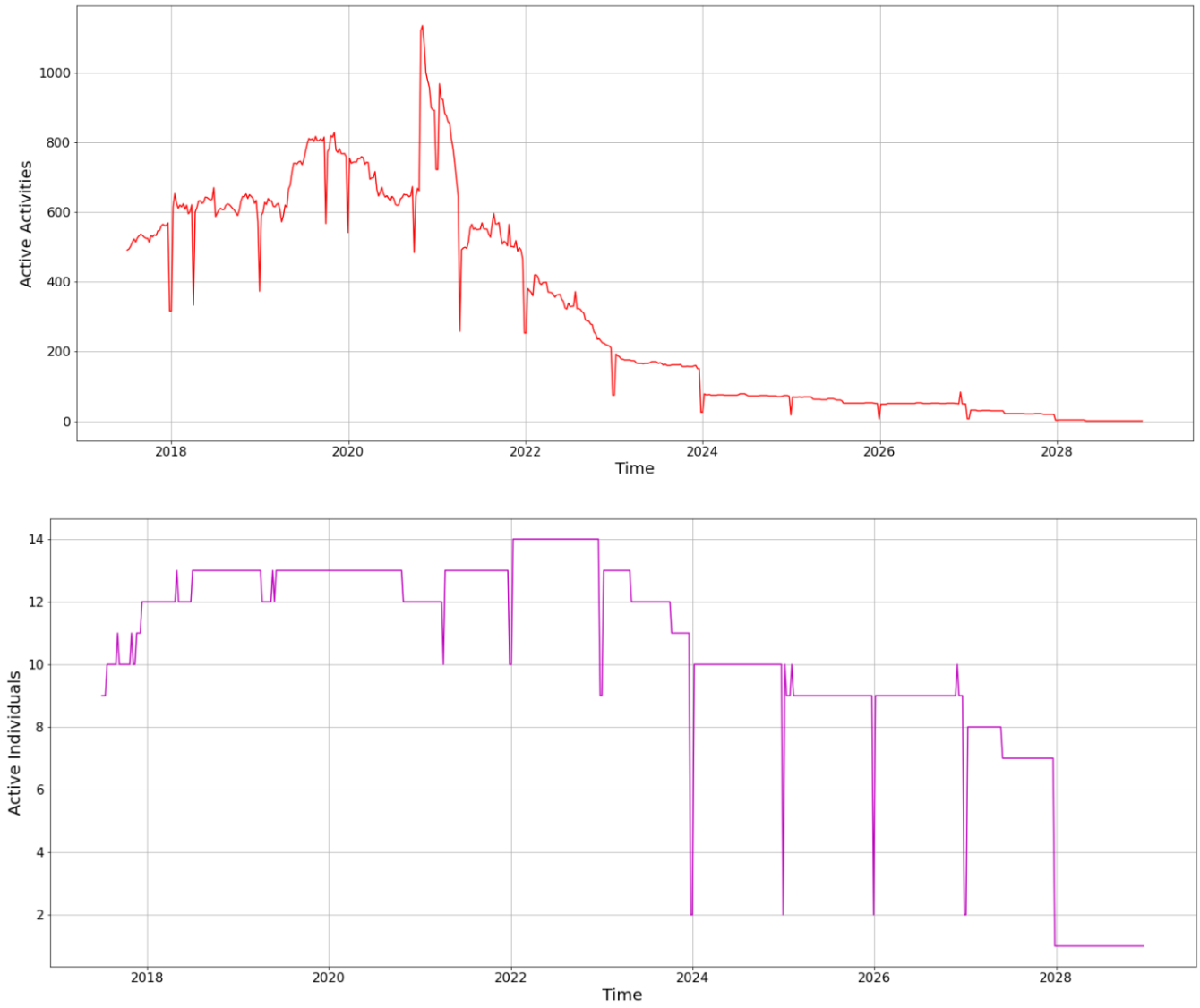

Figure 3-7 Activities and individuals for Ship A (actual design)

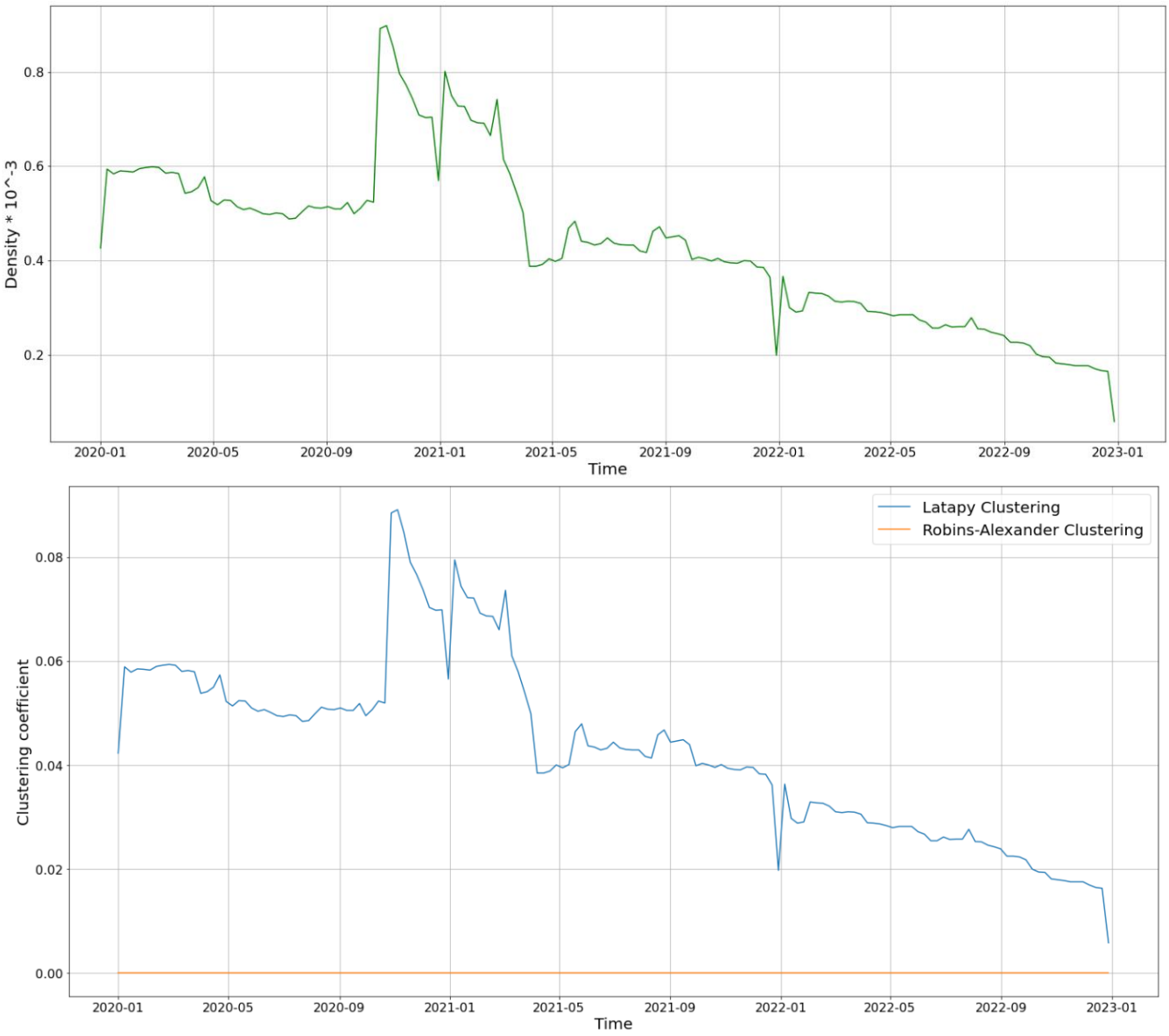

Figure 3-8 Density and clustering for Ship A (actual design 2020-2022)

Figure 3-9 Activities and individuals for Ship A (actual design 2020-2022)

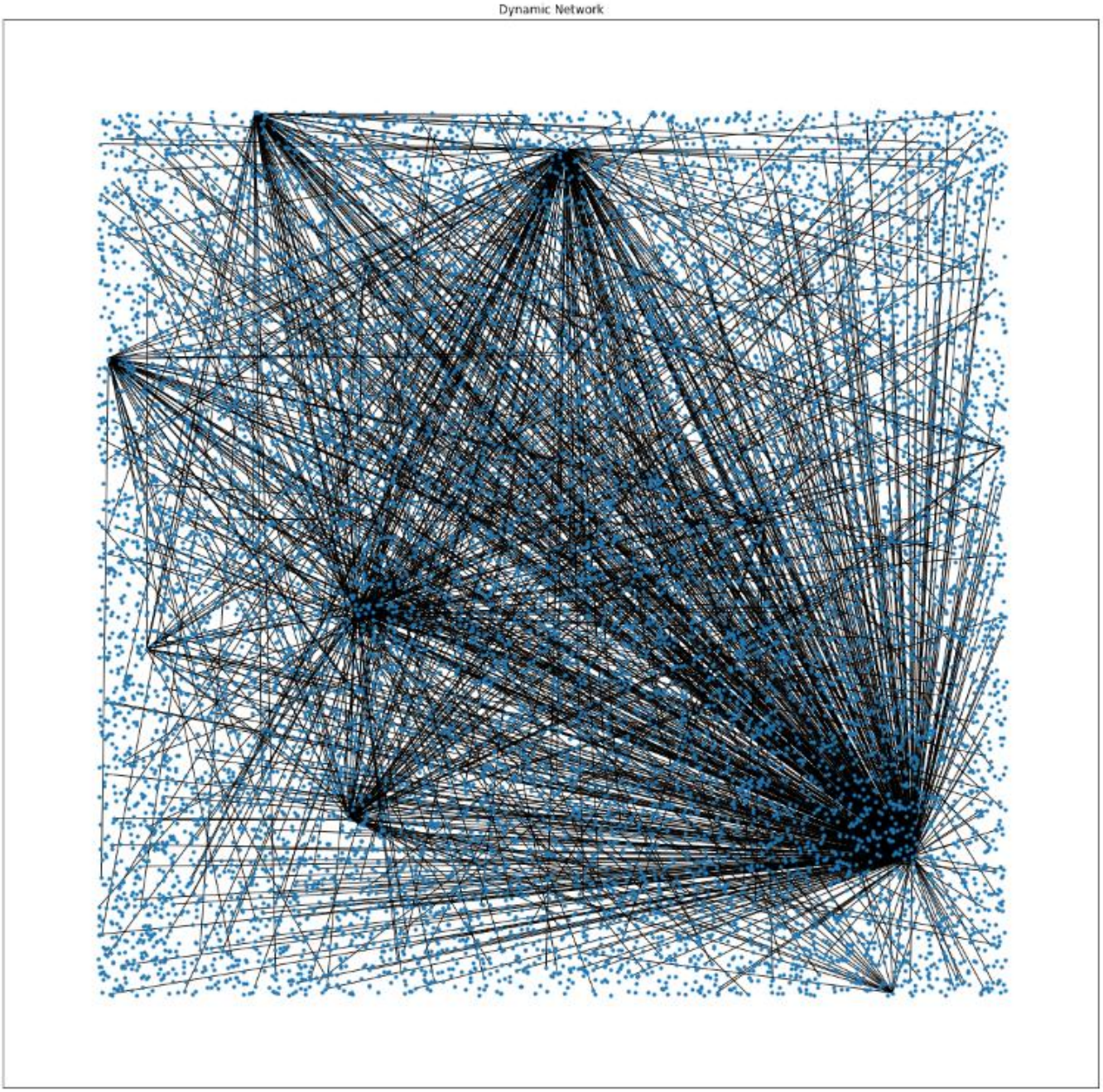

Figure 3-10 Network snapshot at 03/11/2020 for Ship A (highest activity – actual design)

3.3 SHIP B

3.3.1 Planned design

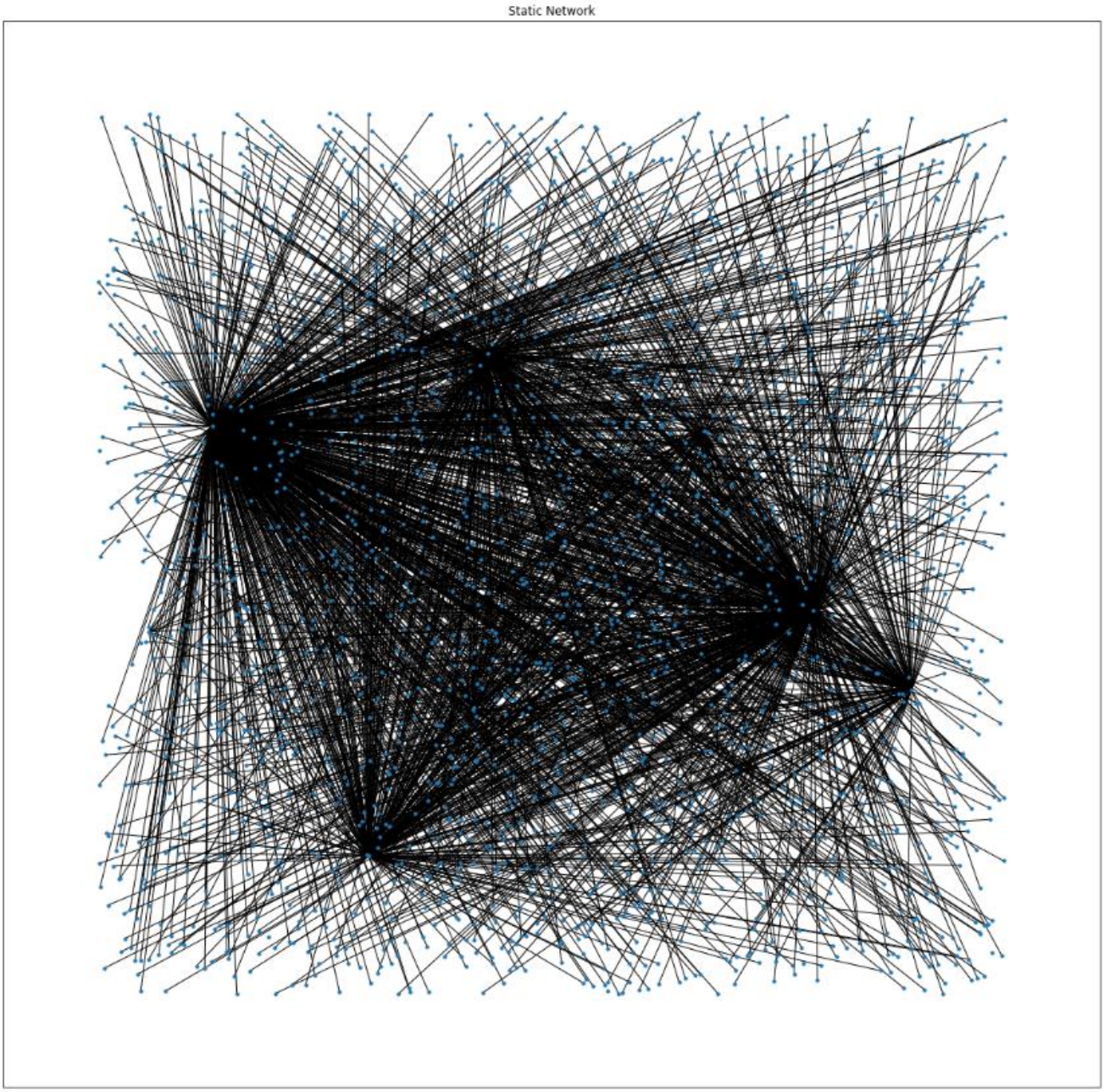

Figure 3-11 Static network for Ship B (planned design)

Figure 3-11 depicts the static network for the second ship for this analysis and comprises ~2000 activities and ~100 individuals.

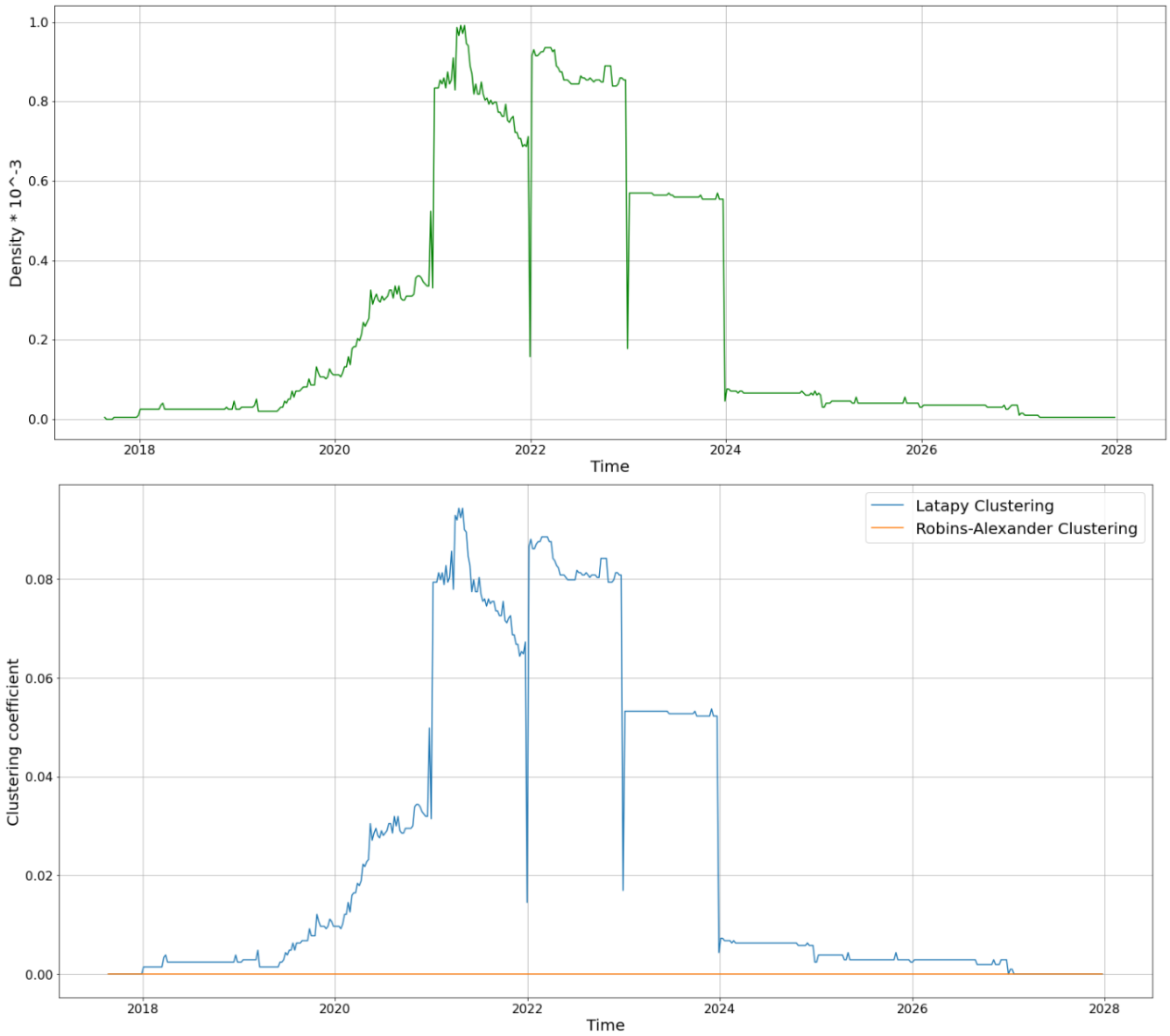

Figure 3-12 Density and clustering for Ship B (planned design)

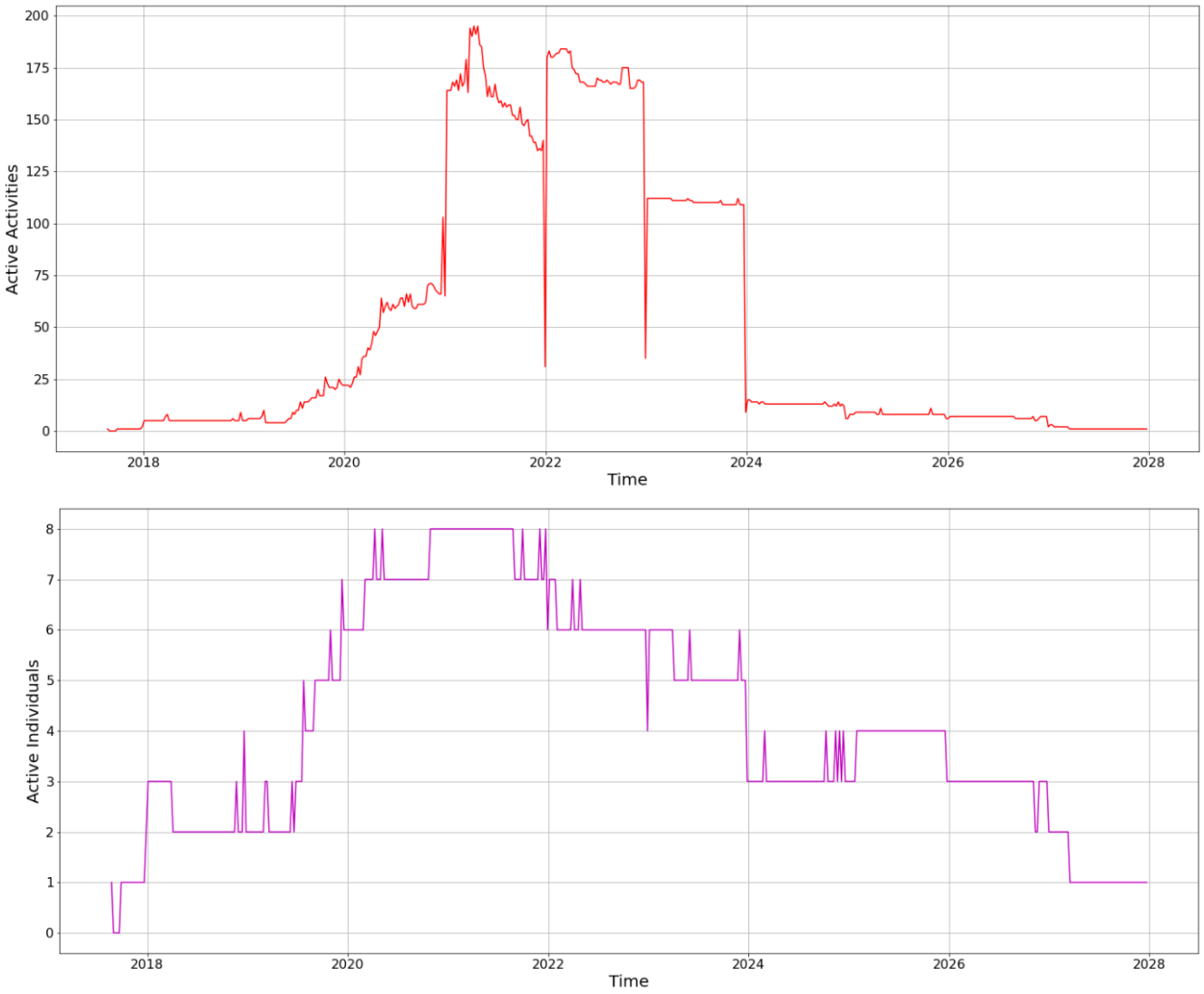

Figure 3-13 Activities and individuals for Ship B (planned design)

For Ship B the design process shape changes a lot compared to Ship A. The region of late 2020 to end of 2023 is particularly interesting and it is displayed in Figure 3-14 and Figure 3-15.

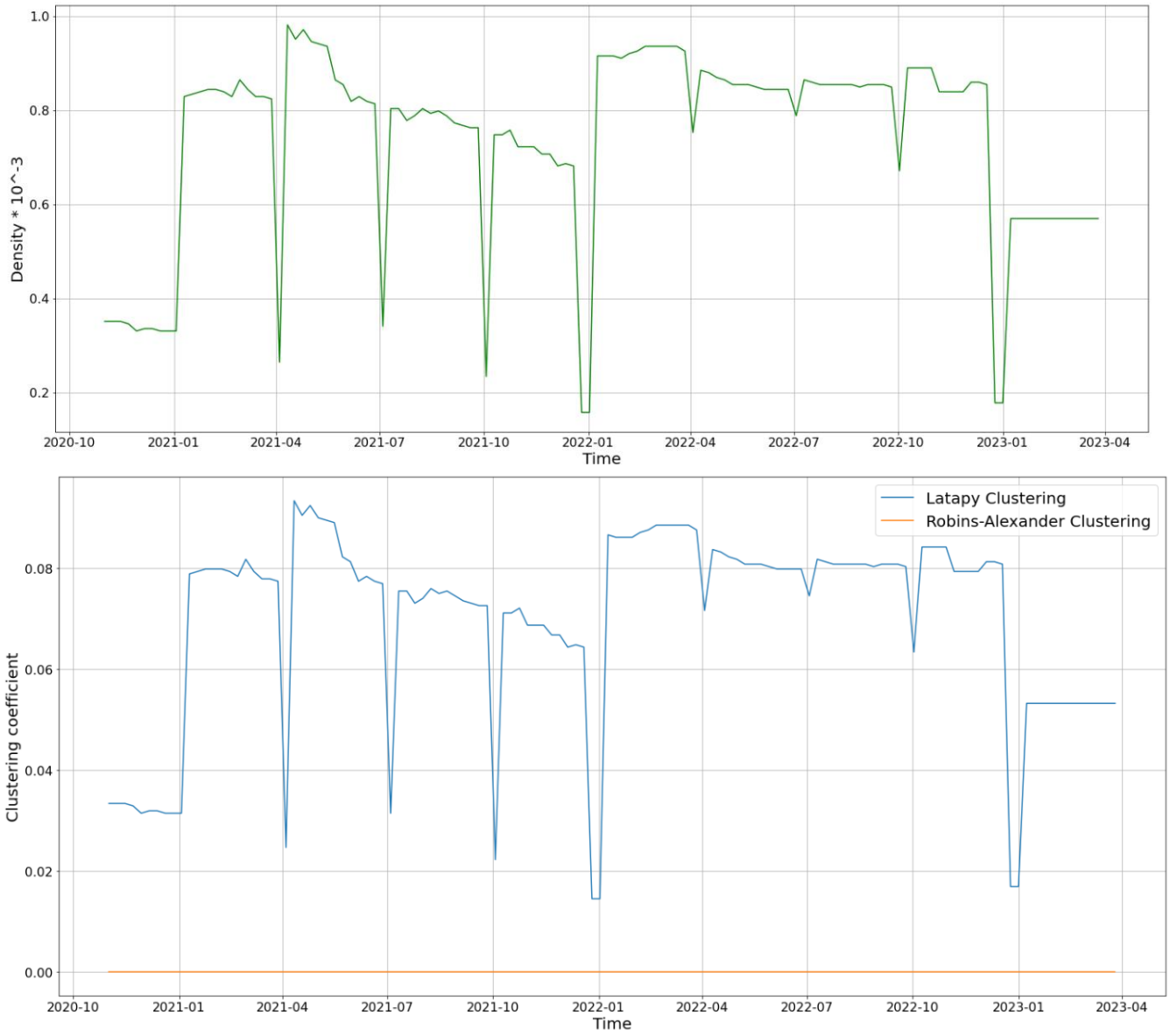

Figure 3-14 Density and clustering for Ship B (planned design 11/2020 – 04/2023)

Figure 3-15 Activities and individuals for Ship B (planned design 11/2020 – 04/2023)

3.3.2 Actual design

The same process and charts are created for the actual design.

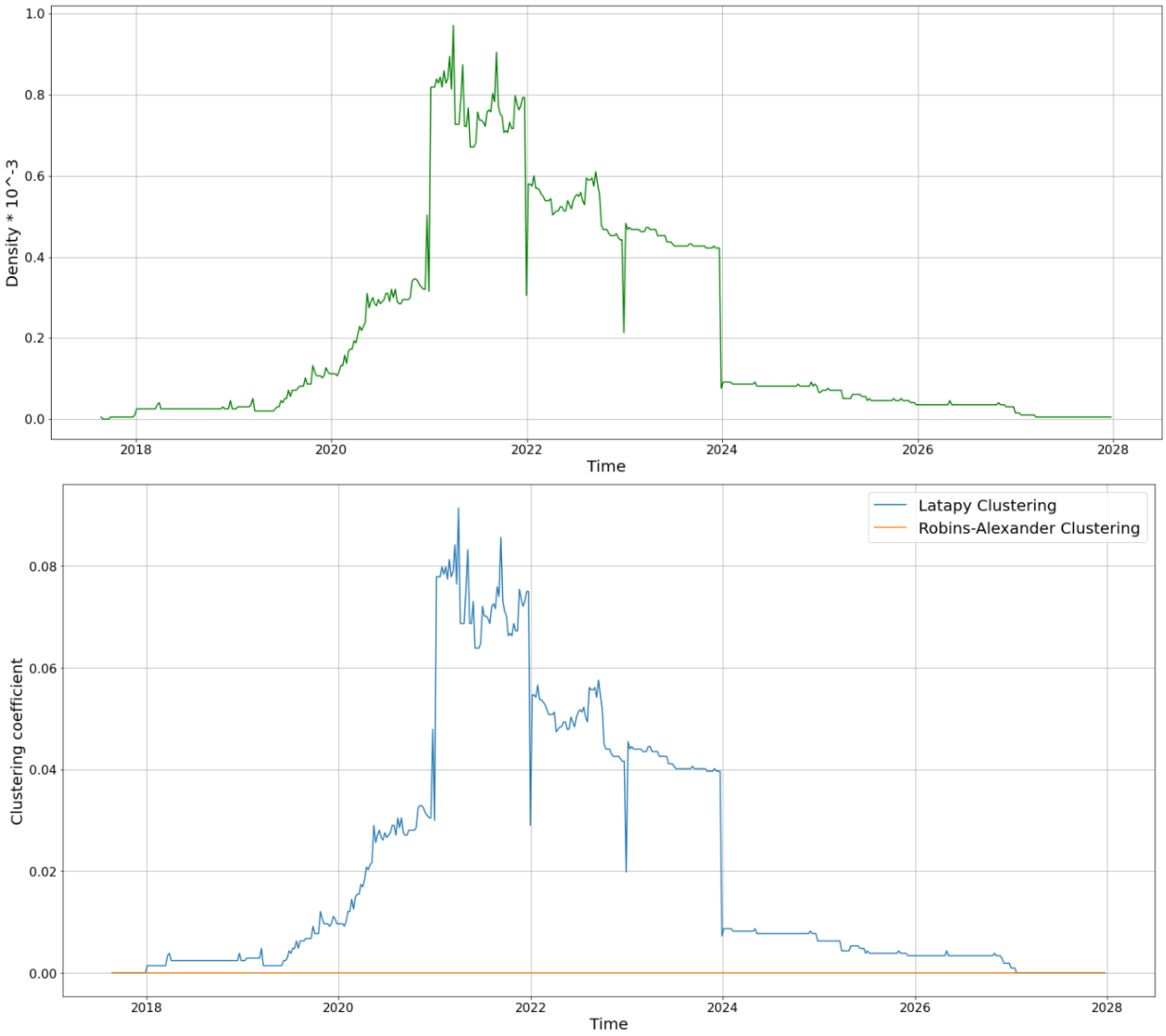

Figure 3-16 Density and clustering for Ship B (actual design)

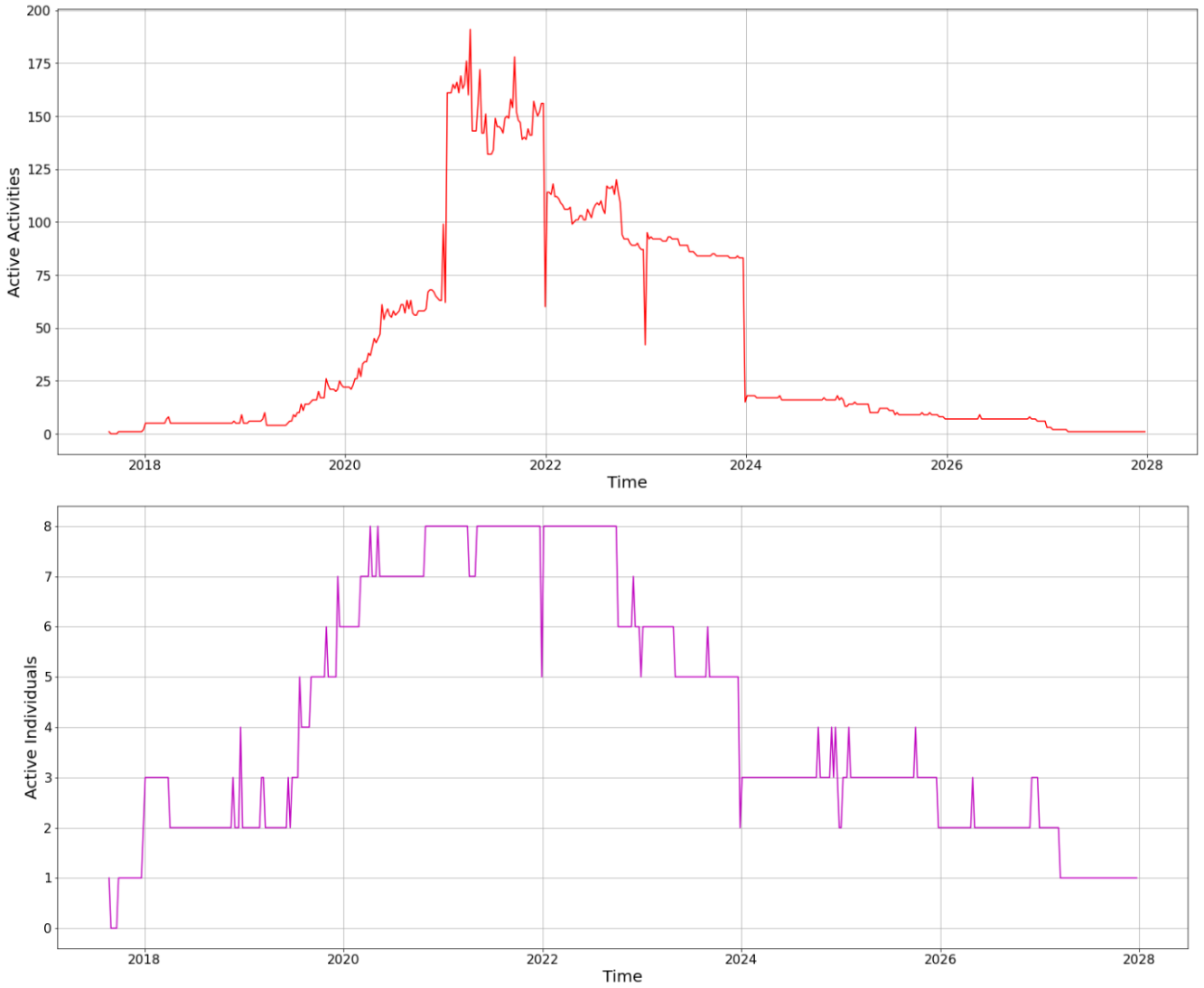

Figure 3-17 Activities and individuals for Ship B (actual design)

Figure 3-18 Density and clustering for Ship B (actual design 11/2020 – 04/2023)

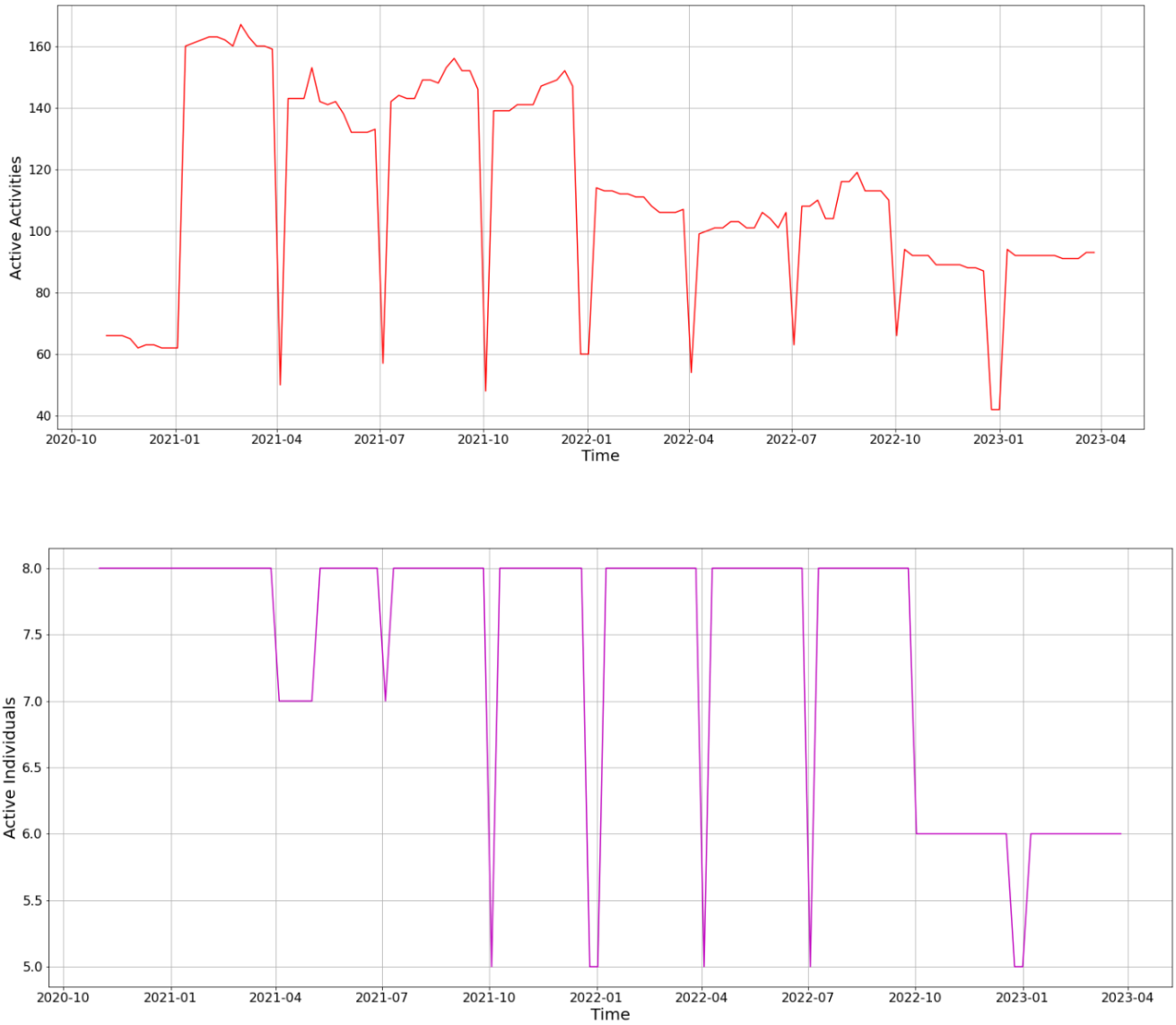

Figure 3-19 Activities and individuals for Ship B (actual design 11/2020 – 04/2023)

3.4 SHIP C

3.4.1 Planned design

Exactly the same process is followed for the third ship of the given dataset. The number of nodes for the static network of Ship C is ~2100 and ~100 individuals.

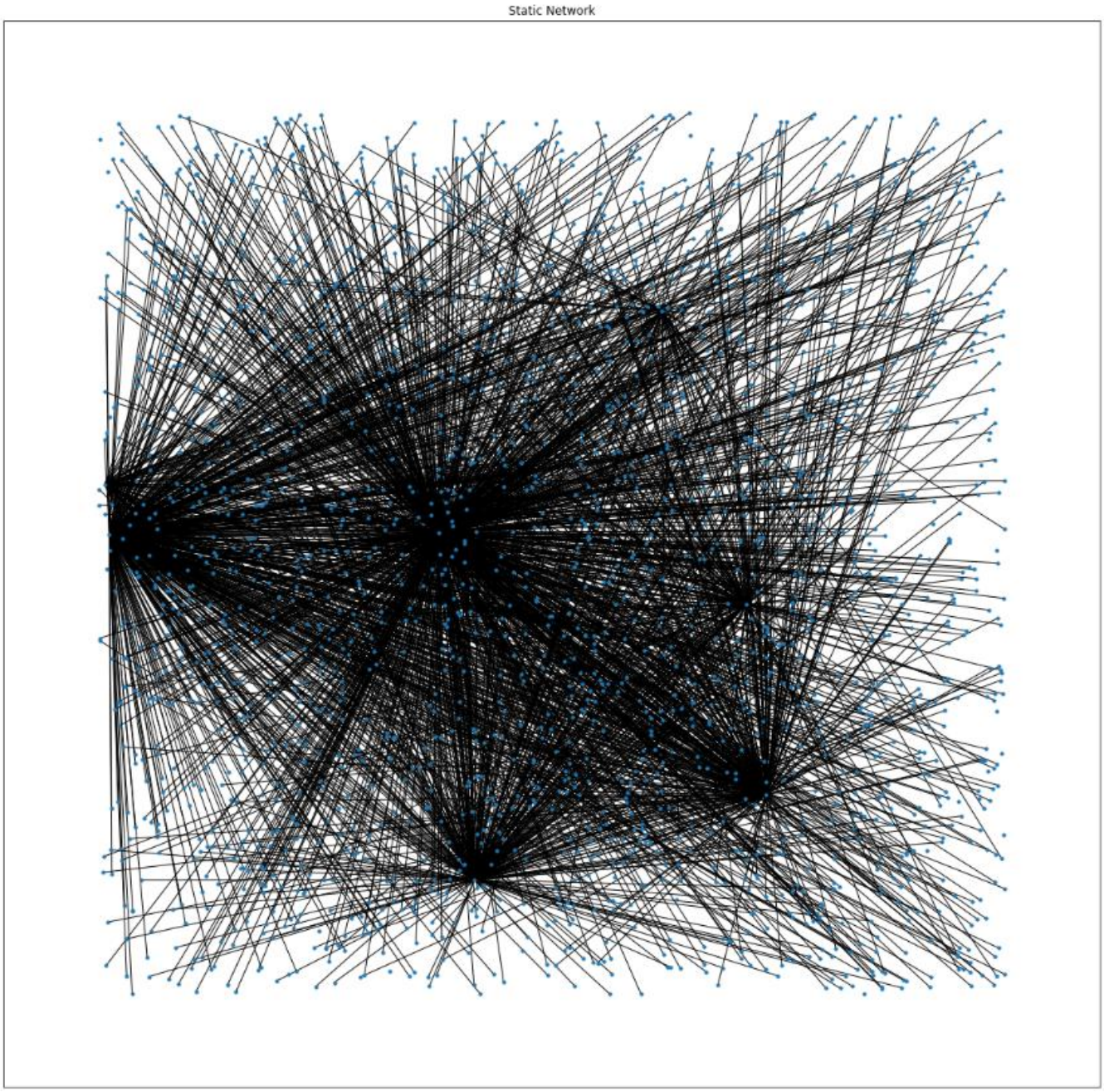

Figure 3-20 Static network for Ship C (planned design)

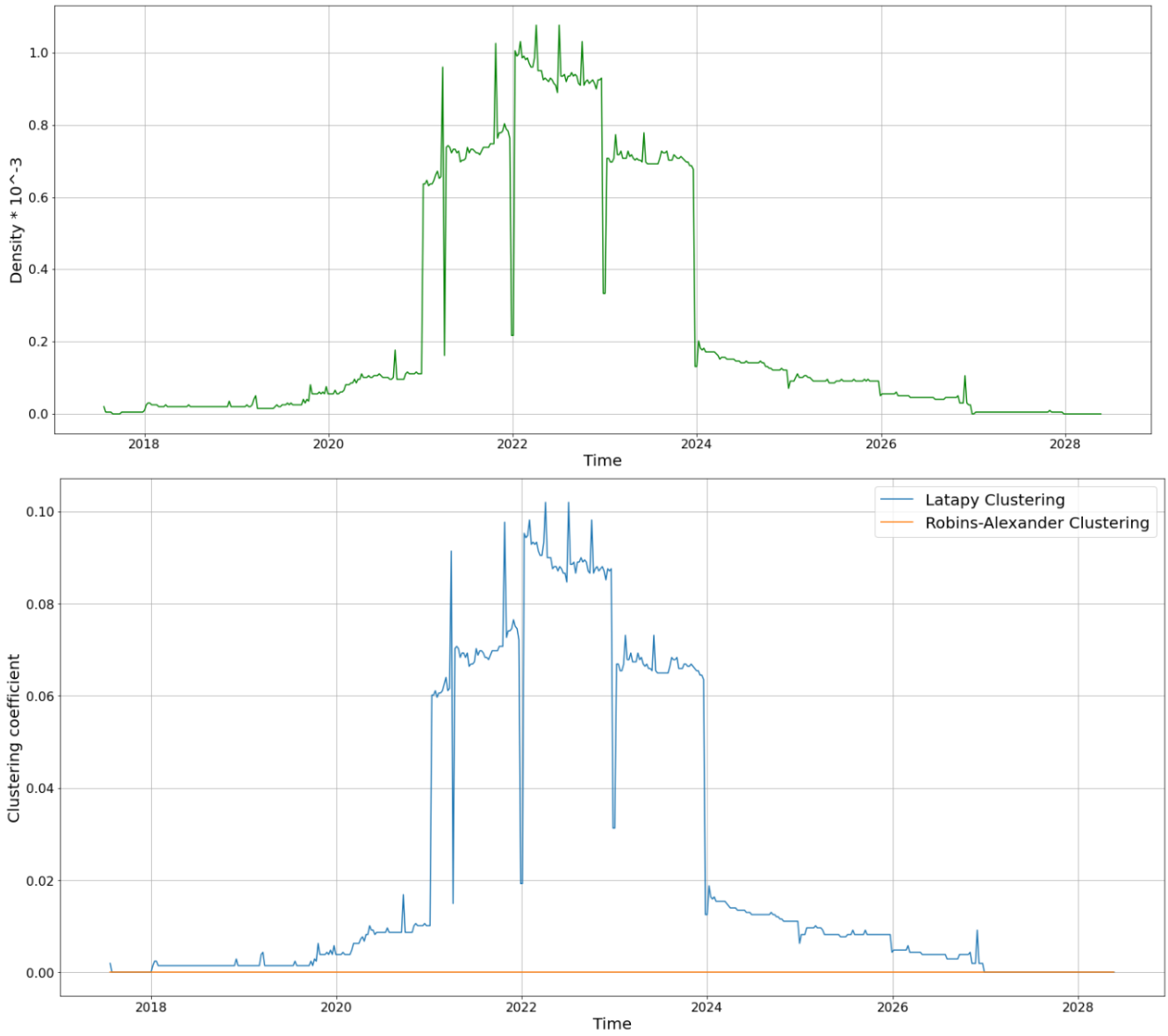

Figure 3-21 Density and clustering for Ship C (planned design)

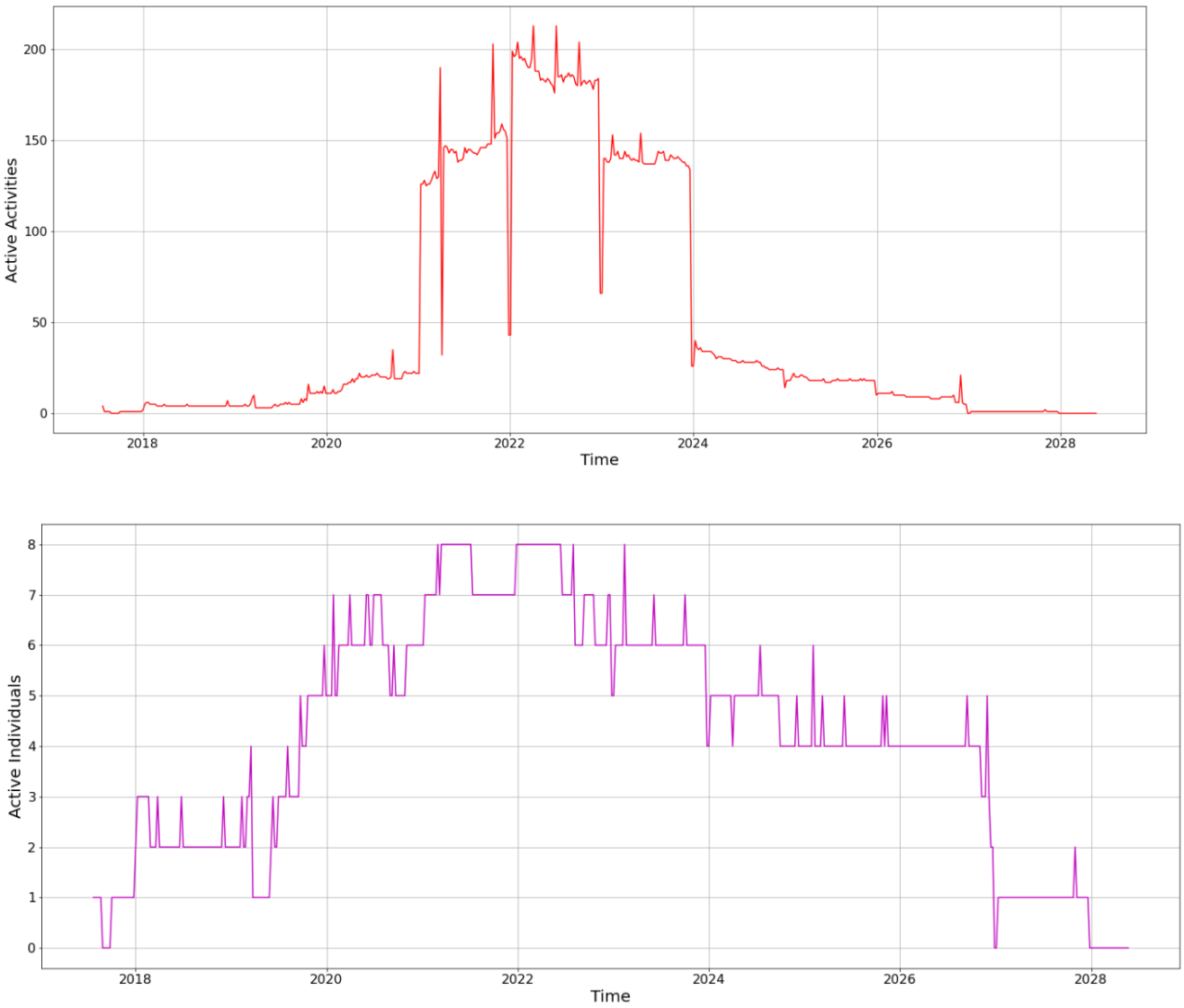

Figure 3-22 Activities and individuals for Ship C (planned design)

The same picture as in Ship B is drawn, so there is no need to go in higher refinement level and smaller scale at this stage.

3.4.2 Actual design

Similar charts are plotted for the actual design of Ship C.

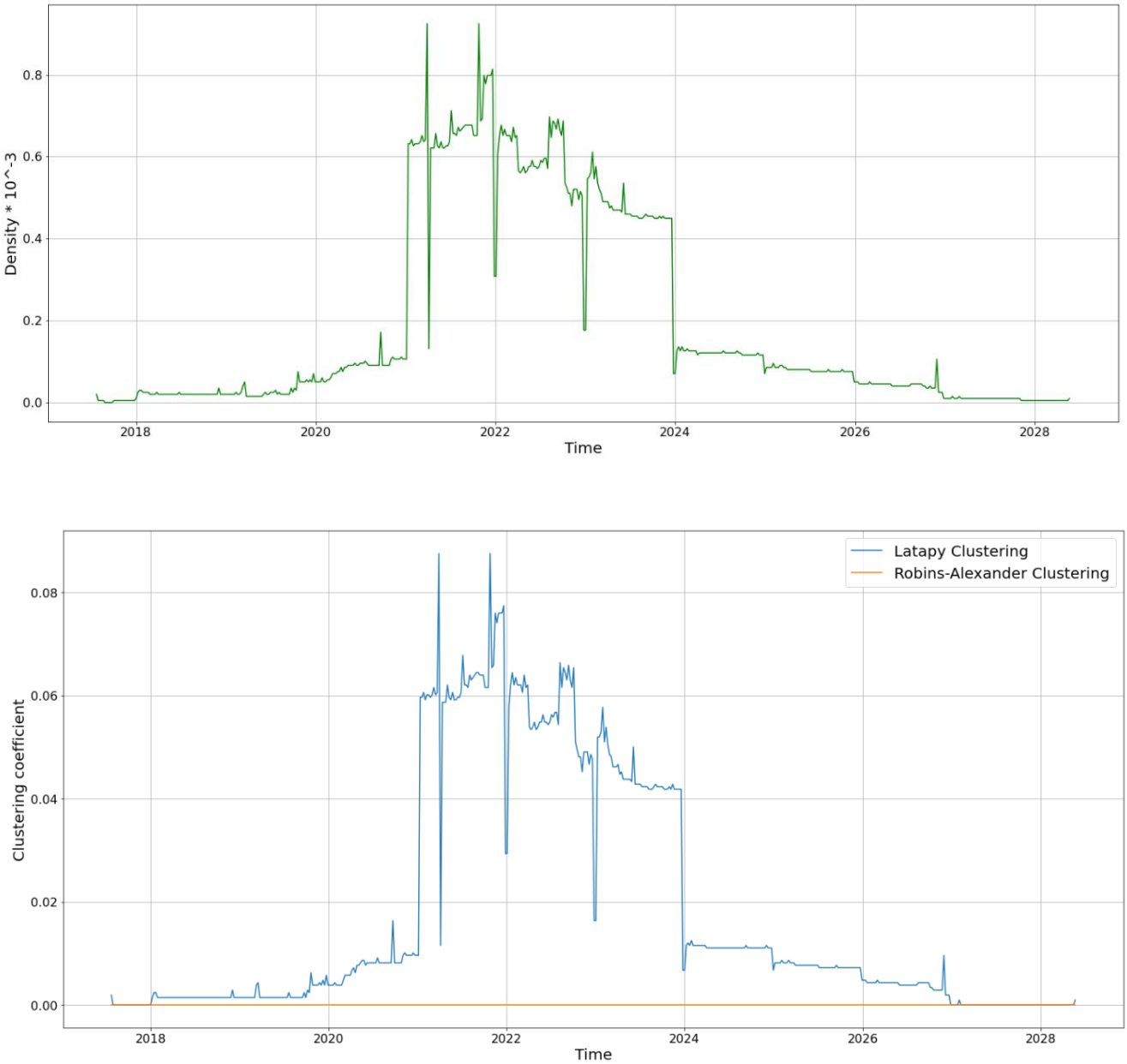

Figure 3-23 Density and clustering for Ship C (actual design)

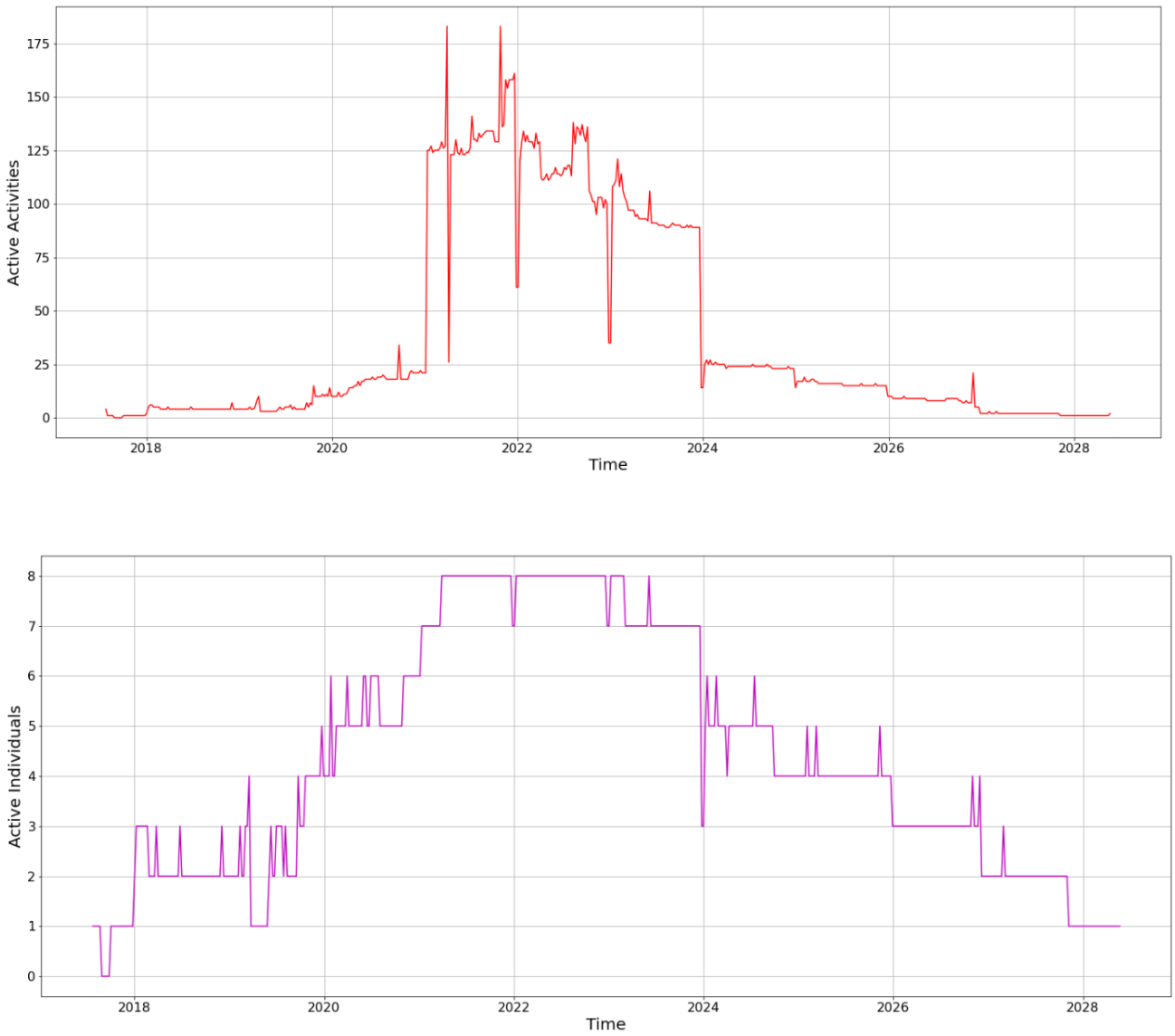

Figure 3-24 Activities and individuals for Ship C (actual design)

3.5 RESULTS - DISCUSSION

The developed model can be employed for any D&M project that can be decomposed in tasks and individuals involved in these tasks and the task activity time window is recorded. The input in the model is a dataframe which in its most basic form should include a list of tasks coupled with individuals, start date and completion date. All additional information can be handled in labels of EFVs and NFVs. Then the user decides the time spectrum of the analysis by setting start and end dates (default option is the

minimum start date from EFVs to the maximum end date from EFVs) and the time step unit. All extra functionalities (printing snapshots of specific time-instances or visualisation modules) are integrated and utilised according to the user's discretion.

The main results for the static graphs for each case study are displayed in Table 3-2. The density values are similar in all ship cases, i.e., ~1% of all the nodes are connected through edges. This is a rather dense network and by checking the Latapy clustering values, it is clear that all six static networks are heavily clustered networks. The Robins – Alexander clustering (transitivity) is 0 for all graphs, therefore no circle paths exist in the networks.

Table 3-2 Basic static metrics across all case studies

	SHIP A		SHIP B		SHIP C	
	Planned	Actual	Planned	Actual	Planned	Actual
Density	0.01	0.01	0.091	0.094	0.091	0.093
Latapy clustering	0.9922	0.9913	0.9516	0.9524	0.952	0.9529
Robins – Alexander clustering	0	0	0	0	0	0

The density values from the dynamic graphs for all ships (Figure 3-3, Figure 3-6, Figure 3-12, Figure 3-16, Figure 3-21, Figure 3-23) and a cross-reference with the behaviour of the average degree of the nodes (Newman, 2010), indicate that the time-dependent graph is sparse; it has much fewer edges than the possible number of edges, which is common in design process networks.

Latapy clustering coefficient generally lies in the region between 0.03 and 0.09 across all 3 case studies, which means that the TDG is at any moment more than 80% de-clustered compared to the static graph. The values of Latapy clustering are in alignment with values recorded for other social networks in (Latapy, et al., 2008).

All metrics display low spikes (sudden decrease in activities and clustering), which are attributed to disruptions from concurrent projects such as other vessel type design or construction reaching critical

milestones. In such cases, the activities are reduced because the focus is on accomplishing the tasks of the competing project (e.g. launch of a patrol vessel). In this respect, the effect of the allocation of resources on different projects immediately affects the design of the ship under study. A multiple-project scenario can provide guidance on how progression in one project affects the completion and efficiency in others.

The behaviour of the number of individuals demonstrates a key aspect of the hiring policy of the company in terms of contractors. From the total of ~100 individuals involved with activities during the design process, at any time the maximum number is 14 for Ship A (Figure 3-7) and 8 for Ships 2 and 3 (Figure 3-17, Figure 3-24), i.e., one order of magnitude less. This reveals the company's contractor policy: contractors are hired to complete specific design tasks and then they are allowed to leave, their work being moved forward by a new team of contractors. The immediate effect of this is that very few individuals are present during the whole of the design process.

The Robins–Alexander clustering value is always 0, both in the static network results and the TDG results across all 3 ships. In (Piccolo, et al., 2018), a similar static bipartite (activities - participants) network was created for an energy plant and the Robins–Alexander clustering was 0.3. A great difference of the two graphs is the size of the activity set, since activities in the naval ship design case range to 1200 (dynamic setting) and 12700 (static graph), while in the power plant design they are 148. This value for the Robins–Alexander clustering coefficient means that there are no cycles of length 4, C_4, in the network. Two individuals involved in the same activity are not involved in any other activity together. The zero Robins–Alexander clustering and the relatively small number of people involved at each time with the activities, indicate that the design process is heavily individual-based and hierarchical (tree-like), i.e., no lateral communication (through a communication activity) between individuals takes place (or recorded). This is also supported by the contractor hiring policy and the lack of communication between different contractors since they are different legal entities.

3.5.1 Planned vs Actual design discussion

The first case study (Figure 1-1) shows that the planned design has many dates in which most of the activities were put on hold compared to what actually happens. Since the data were obtained in mid-2022, it would be interesting to focus on the design cycle during this period. For the results after 2023, there is a small discrepancy between the planned and actual activities, with the planned activities being greater

Figure 3-25 Ship A planned vs actual comparison (bigger discrepancies are circled)

than the actual. Since this region is in the future, they are not actual, but replanned activities. Nevertheless, it is obvious that the end user feels confident that the number of activities will be less than those planned

at the start of the project, even though in the second half of 2020, the greatest differences were encountered with the actual workload being 40-50% higher than the planned. In terms of the individuals assigned on the design process, there are not great differences apart from the 2021-2022 period, where 2 extra designers were involved in the design. The reason for this is that due to COVID-19, the tasks that were completed were less than those planned, so more people got involved to meet the set deadlines.

The impact of COVID-19 is better depicted in Figure 3-26, where during the first UK lockdown (March 2020 – May 2020) after the planned drop, the actual activities are less than planned. After the lockdown seized, immediately the activities needed became more and the planned schedule proved insufficient. To avoid a similar situation during the second UK lockdown (November 2020 – December 2020, remote working in effect), two more designers were added in the design pipeline to cope with the number of activities. This resulted in similar reduction of active tasks as in Lockdown 1, but the end of Lockdown 2 did not cause a similar rapid increase of tasks required. An analogous process was followed during Lockdown 3 (January 2021 – March 2021).

For the second case study (Ship B), the actual and planned timelines have small differences between them at the beginning of the design process calendar (Figure 3-27). The first discrepancies appear in January 2020 when COVID-19 pandemic first appeared, followed by Lockdown 1. Then during Lockdown 2 instead of reducing the participants in the design process, the same number was retained (8). An indication that stems from the results that Lockdown 3 had the highest impact on the design process. There were 8 designers assigned, instead of the planned 6, but still the activities that happened (~110) were much fewer than those planned (~180).

The third case study proved to be more interesting in terms of both the industry policy and COVID-19 impact. The design of the third ship was understaffed even before the first lockdown. For most of the design process until 2020 the designers involved were 2 less than planned originally. The most probable explanation for this is that these designers were involved at the same time with designs of Ship A and Ship B. During lockdowns 2 and 3 the activities planned were ~170 and the actual activities taking place were ~110. After the third lockdown, the focus was shifted to Ship C and 2 more designers than planned were added in the design pipeline.

Figure 3-26 Ship A planned vs actual design (period 2017-2023) and COVID-19 impact

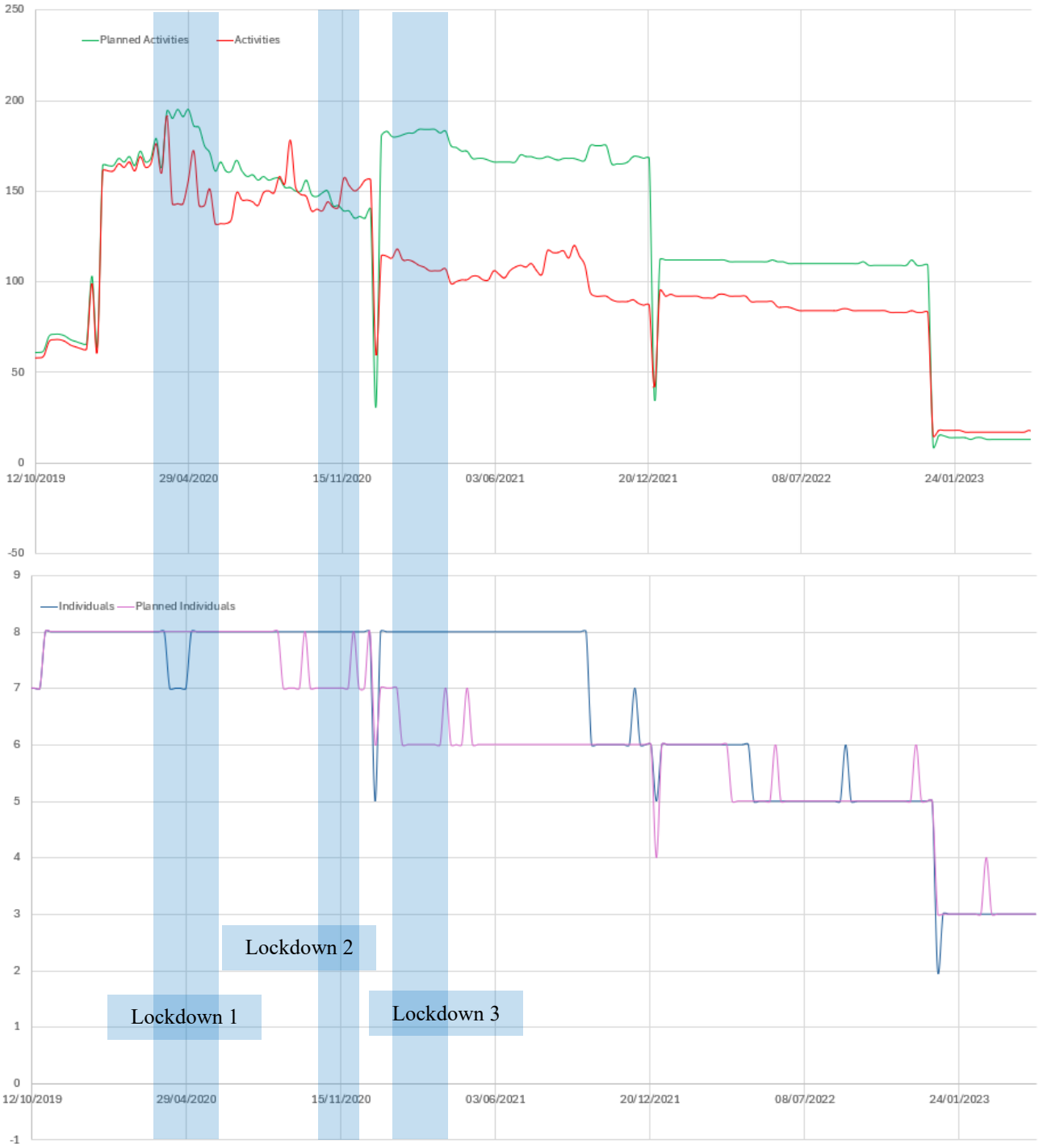

Figure 3-27 Ship B planned vs actual design (period 2019-2023) and COVID-19 impact

Figure 3-28 Ship C planned vs actual design (period 2017-2024) and COVID-19 impact

3.5.2 Learning from Ship A

The next valuable insight stems from a comparison across the 3 ships. How does one design affect the other? Is there any contribution or knowledge transfer from the original design to subsequent designs? A first answer is that the number of tasks mainly and in a smaller extent the number of individuals, are significantly reduced, leading in nominally faster designs. However, the three ships of the case studies are being developed simultaneously and a change in Ship A, immediately changes the design of the subsequent ships. Because Ship A requires intrinsically more tasks (being the first) there is a need of normalising the activities across all designs. The normalisation is performed with the number of total activities in the static graph of each case study and the results are expressed on a percentage of static graph activities (Figure 3-29) (relative activities).

At the early stages of the design timeline relative activities and individuals that relate to ships B and C are significantly less compared to their respective values for Ship A. The activities peak for Ship A which corresponds to a major milestone (batch delivery) in its design process and only then is the focus shifted to ships B and C, peaking in activities and individual involvement. After the COVID-19 period the number of individuals involved with Ship A remains high but the relative activities in which they participate are reduced. The current projection is that the 3 ships during the last 3 years of their design are going to have similar number of relative tasks and are going to be delivered simultaneously. Overall, it is concluded that the variations between the designs of the three ships, Ship A being heavily worked on at the 1st third of the design timeline, ships B and C starting on the 2nd third of the design timeline and all three ships are being handled together at the end, was a strategic design decision applied as best as possible, despite the impact of the pandemic.

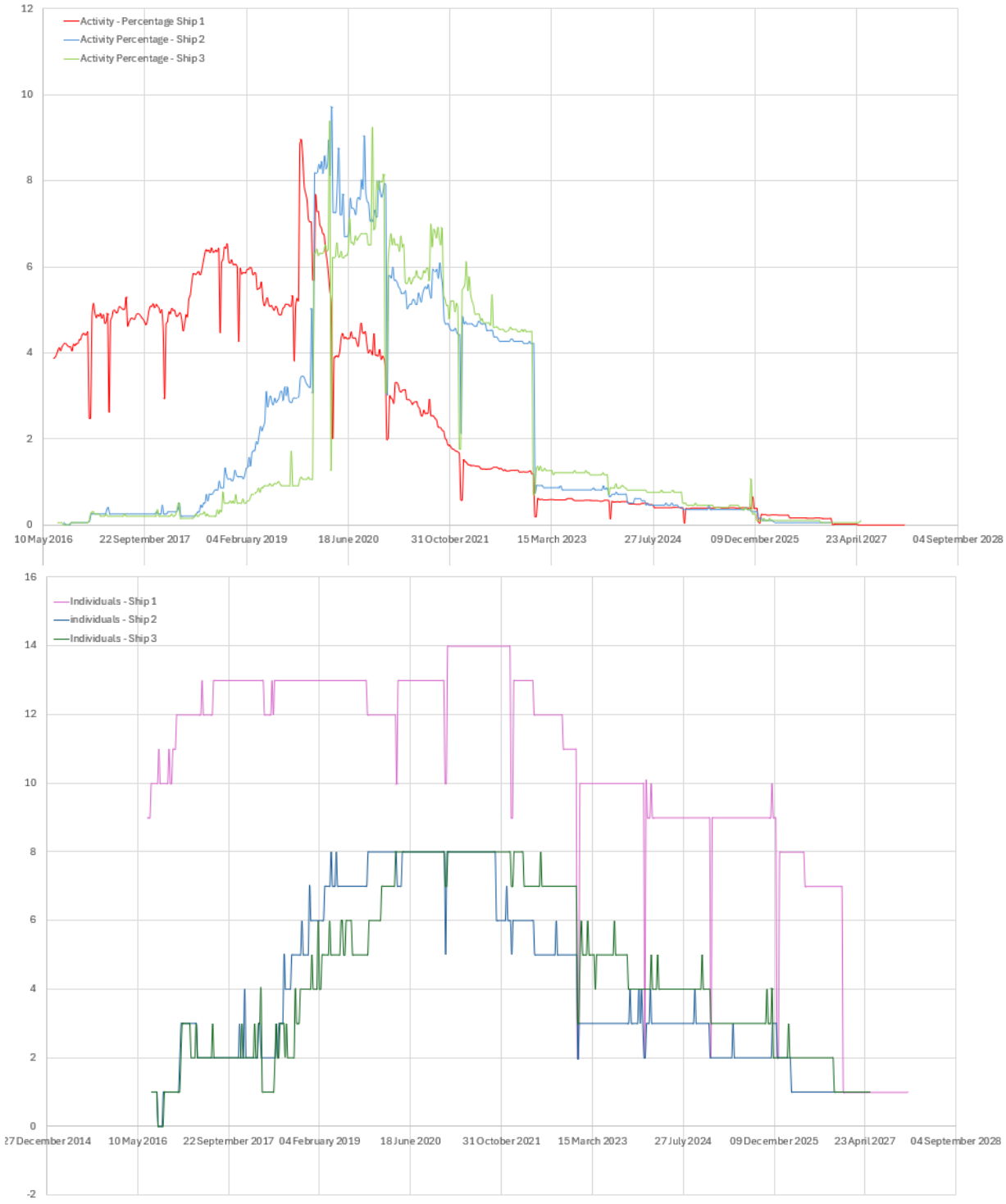

Figure 3-29 Activity percentage and individuals across the three ships

3.5.3 Centrality

So far the analysis was focused on the design process and the evolution of the design process metrics (clustering coefficient, density, number of activities and individuals) in time. This subchapter concerns the topological structure of the TDG and identifies the most central tasks and individuals for the design of the three ships. A key observation is that the centrality differences between planned and actual design networks are very small and for all intents and purposes the central tasks and individuals are the same between planned and actual design processes.

Table 3-3 Maximum centrality values for two example cases of Ship A (actual design)

	Trough (28/9/2020)	Global Peak (2/11/2020)
Active participants	12	12
Active activities	485	1138
$\max(DC(u))$	2MFDA727: 0.01	2MRD3T45:0.01
$\max(DC(v))$	Individual 56: 0.00844	Individual 25: 0.04048
$\max(BC(u))$	2MEIT189: 3.24e-09	2MSEN101: 8.71e-09
$\max(BC(v))$	Individual 56: 6.97e-05	Individual 25: 0.00161
$\max(CC(u))$	MAWNC103: 1.99217	MPLHA635: 1.05468
$\max(CC(v))$	Individual 2: 1.00775	Individual 30: 1.00772
$\max(EC(v))$	MMMSE495: 0.16777	MMMSE911: 0.22111
$\max(EC(u))$	Individual 56: 0.67533	Individual 25: 0.83333
$\max(PR(v))$	2APMC626: 0.05611	2PHFT575: 0.04844
$\max(PR(u))$	Individual 20: 0.32431	Individual 34: 0.43524
$\max(KC(v))$	MMMSE495: 0.15666	MMMSE911: 0.19334
$\max(KC(v))$	Individual 56: 0.52366	Individual 25: 0.67391

In light of this, the centrality results focus on the actual design TDG. Table 3-3 includes the results for the two cases that display the greatest difference between trough and peak for Ship A. Many activities at these time instances display the same value, since they have the same distance from individuals and for similar topology reasons. Those displayed are selected indicatively.

Since the naval ship design process is heavily individual based, it is reasonable to assume that the nodes with higher degree centrality (DC) correspond to individuals. However, the use of the bipartite definition for degree centrality results in activities being more central than individuals. The reason for this is the great difference in the number of elements in the activities and individual sets. The individual set comprises ~100 nodes and the activity set comprises ~12800 nodes. Dividing the degree of each node with the opposing set (equations 2.6a, 2.6b) results in high degree centrality for activity nodes and low degree centrality for individuals. The bipartite degree centrality definition is particularly useful for comparative analysis between the bipartite sets, however in this network a limitation is discovered. A way to avoid this issue would be to neglect, momentarily, the bipartite nature of the network and use the classic degree centrality (normalisation with all the nodes). This approach produces centrality values that are higher for the individual nodes and much lower for the activity nodes, which is reasonable, however it is not recommended. In this work, the bipartite nature is not neglected, but instead the two sets are treated separately, the ranking is kept within sets, and no conclusions across sets are derived, mitigating the impact of the bipartite degree centrality definition.

In the lowest traffic region (28/9/2020), the most central activities were activities:

- 2MFDA727: authorisation of safety requirements for power distribution systems
- 2MEIT189: review and management of support systems
- MAWNC103: logistics for 2020
- 2APMC626: cable management design
- MMMSE495: propeller GB safety

From the individuals set, more central are individuals 56 and 25. Individual 25 is supply chain manager and their main duty is the procurement of equipment. A note-worthy observation in the data is that Individual 56 is not currently employed by the design company and they either had been hired as a contractor or have retired/moved to another company. Therefore, an individual who is central during the trough period of the design process is not participating in the design process as of May 2022.

In the November 2020 peak time-instance), the most central activities are:

- 2MRD3T45: review of metallic pipe fittings
- 2MSEN101: changes in system engineering management
- MPLHA575: delivery of design model and data set
- MMMSE911: modelling studies
- 2PHFT575: escape and evacuation certification

From the individual set, the most central nodes are:

- Individual 25: engineering manager
- Individual 30: head of electrical engineering – reviewer of electrical plans

Similar results can be yielded for all ships at any time instance by the model. The most central tasks for Ship B and Ship C are not the same, however they still have relatively high centrality values. From the individuals set, individual 25 is the most central participant in all 3 case studies.

3.5.4 Rework quantification

Rework is measured for the tasks in a two-fold manner. First the tasks with the highest delays in terms of completion date are ranked, and secondly the tasks with the highest labour unit difference (planned – actual) are ranked. A major conclusion is that the activities that display the highest values of completion date delays are the same across all 3 case studies. From the given dataset only 12% of the activities display recorded delays, however only 4% have recorded extra labour units. This seems non-sensical since the extra time devoted on the sizeable number of delayed activities, should be mirrored by the recorded values of extra labour units. This difference points to insufficient reporting of rework on the dataset. Love's term organisational zemblanity (Love, et al., 2019) arises from comparable cases where a company understands the concept of rework and the negative impact it has on the design project, but fails to do anything about it, treating it as a mere symptom of the design process.

Nevertheless, the model does rank the activities with most labour unit difference between planned and actual designs and some conclusions may be drawn. Table 3-4 shows the activities that display the highest labour differences. Most of them are unrelated to each other, but Project Control appears 3 times. Looking through the data all activities that relate to Project Control have extra labour units assigned to them, so it is safe to say that Project Control tasks are either rework producing or mostly affected by rework in other tasks. The most probable situation is that both are happening simultaneously.

Table 3-4 Activities that display highest labour differences

Activity	Labour unit difference
2MSSE230	6835
2MEMP230	4920
2MSSE214	4552
2PHDZ1O6	4339
2MEMP221	3491
2MEMP222	2932
2PBAM101	1800
2MENC221	1791
2PCTX362	1524
2MPPP250	1448

Similar ranking is performed based on the task completion date delays. The values displayed in Table 3-5 are working days. Investigating the results it is inferred that most of the tasks that display high variation between planned completion date and actual completion date are design changes or equipment changes (e.g. MA0389PC, MPDHS941 from Table 3-5), which is the definition of rework. Both rankings can be used complimentary to one another, nevertheless the results from scheduling ranking are likely to be more accurate, due to the greater data size that includes the relevant information.

Table 3-5 Activities that display the highest completion date delays

Activity	Delay
MMMSEF41	1347

MA0389PC	1331
MPDHS941	959
MMMSE492	624
ZBS22601	600
MMAUX512	497
MMAUX056	464
MMMSEF45	447
MMLIG019	433
MMAUX053	393

The results were discussed with industrial experts and they indicated that they are in line with their experience for most of the candidates that present high rework. Specifically, the Project Controls tasks were heavily impacted from COVID19 and required a lot of man-power immediately after lockdown seizes. Some of the delays in terms of rework, triggered questions (e.g. Diesel generator change), where they did not expect it to take that much time. A work proposal was to change the time discretization to months instead of weeks, however this would hide insights such as the COVID19 impact in their pipeline. Nevertheless, the model allows this and the user can change this parameter.

4 NETWORK VISUALISATION PLATFORM

A secondary aspect of this work is the development of a visualisation tool to enable a better understanding of the nature of the TDG, network topology and centrality metrics. The visualisation tool code does not require yard data to work and can be used independently with a variety of network input formats (.xlsx, .txt etc.) with minor modifications. The codes were developed in Python and are added in Appendix B – Visualisation tool codes.

An augmented visualisation provides a multitude of benefits for the user, most important being the enhanced understanding of complex data. Types of augmented data visualisation include augmented reality (AR), virtual reality (VR), mixed reality (MR) and simpler techniques that can highlight previously hidden aspects of the data. Their major contribution is simplification (e.g. giving annotations) using real-time data (Craig, 2013) allowing for improved decision making, better interactivity and user engagement. Design process networks are highly intricate, especially in D&M applications (Zhang & Kwok, 2018), with numerous stages and interdependencies. An advanced visualisation tool may present this inherent complexity in an organised format, allowing for easier comprehension of the process, identification of bottlenecks and inefficiencies, interactivity and opportunities for design optimisation. When multiple collaboration teams are involved, an advanced visual representation allows the everyone to understand the design flow and the individual roles leading in better collaboration and feedback (Wolfartsberger, 2019). Advanced visualisation tools often provide simulation capabilities (Zhu, et al., 2019), allowing to test various scenarios within the design process network (process changes and resource reallocation), focus on potential future issues and refine the design process before implementation.

After the data are input in a Pandas Dataframe, it is very straightforward to create lists of them in Python and make them compatible with other Python libraries used for plotting. Specifically, a combination of matplotlib (https://matplotlib.org/stable/index.html), pyvis (https://pyvis.readthedocs.io/en/latest/documentation.html) , and plotly (https://plotly.com/python-api-reference/) libraries were used to produce the visualization platform. The proposed visualisation tool includes three types of visualisations for the design process: a) dynamic network plot with data integration, b) an improved display for the temporal metrics and c) a node-wise metric plotter for metrics that are local in time. The first option is displayed in Figure 4-1, where two modes are shown. The first mode presents

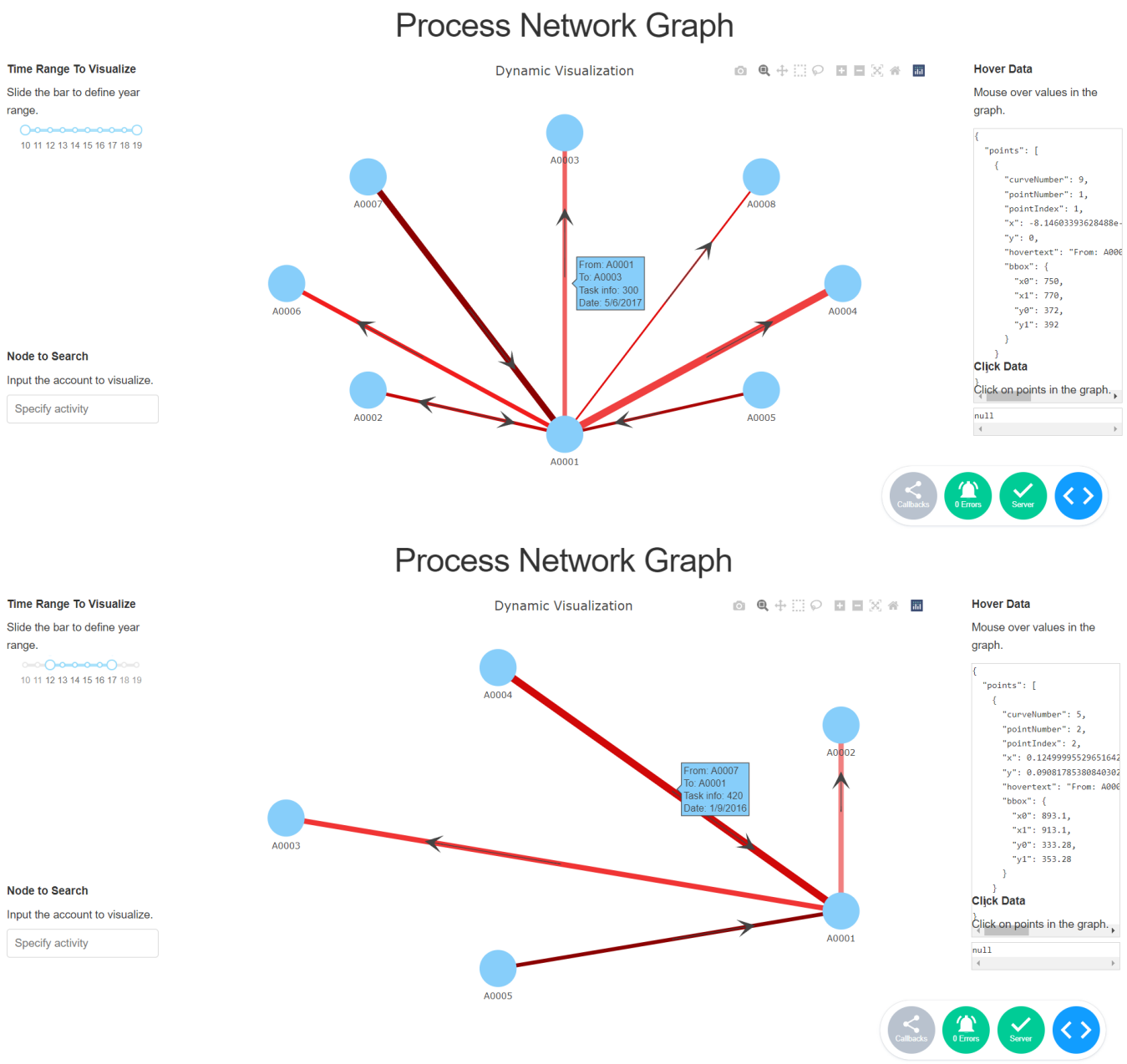

Figure 4-1 Dynamic graph visualization at full time-range (top) and a more restricted time-range (bottom)

all edges activated simultaneously (full time-range), i.e., the static network, while the second mode has changed the activity time window. The user is able to modify the time-unit with the time-bar slider in the "Time Range to Visualize" window. Some restrictions of this visualisation at this stage are that the code uses directed edges, and the nodes do not have a standard position on the graph. An option given to the user is the "Node to Search", which will reconstruct the graph around the selected node (activity or

individual). The “Hover Data” window shows important information stored on the edges and/or the nodes. This can be a description of the activity or any other related info that is available in the dataset and needs to be retained. The user may also hover the cursor above an edge or a node and see available quick access data.

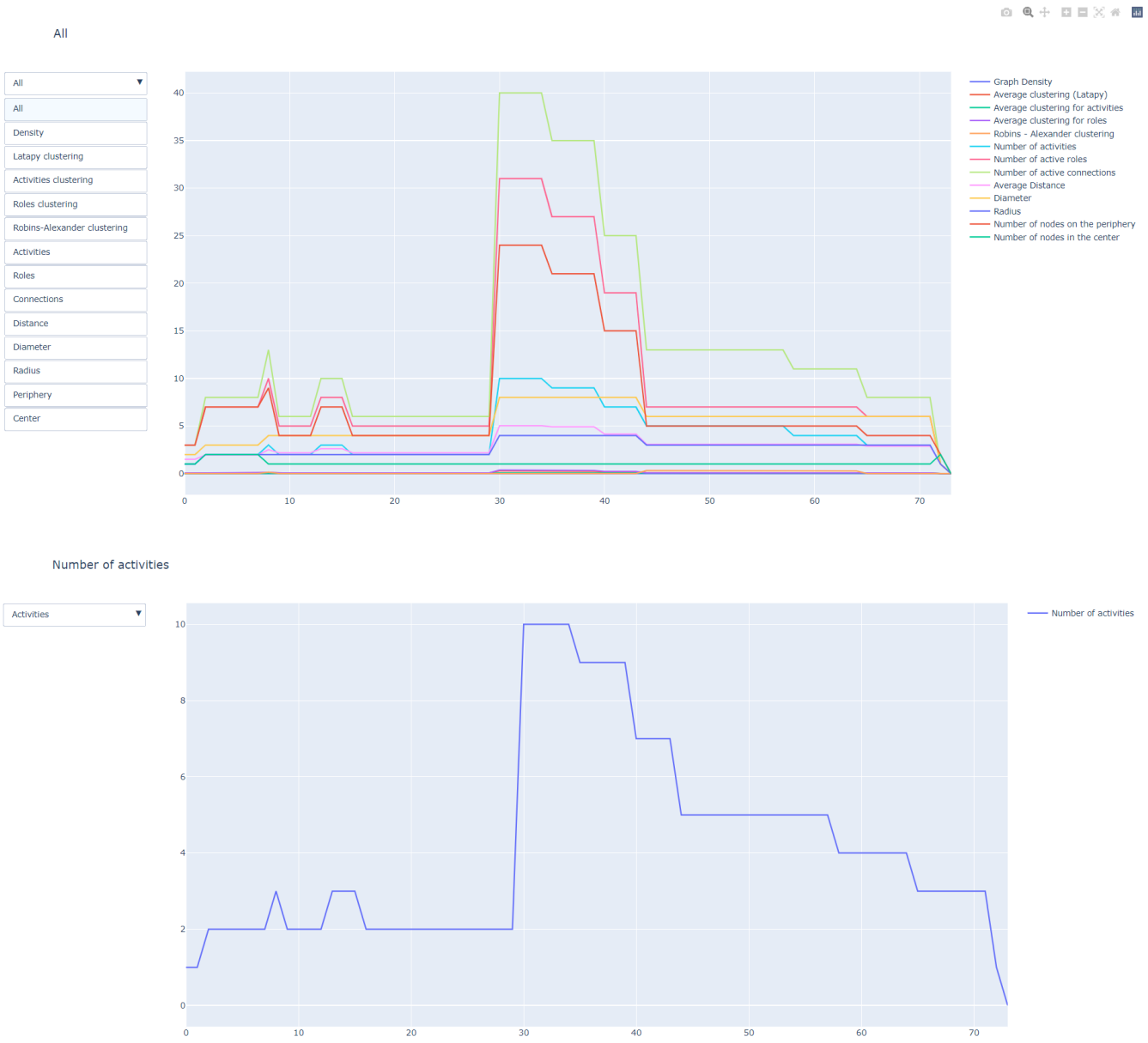

Figure 4-2 Dropdown menu for visualizing metrics

The second visualisation operation is a more user-friendly option for visualising all the plots created for the dynamic network. There are two options available, one being the dropdown menu (Figure 4-2) and the second is a multiple choice plotter (Figure 4-3) for visualising metrics that are comparable between them.

The third operation focuses on properties that are nodewise and should be displayed at every time instance instead of the whole time spectrum. Figure 4-4 shows a degree centrality plot for the network created for

Figure 4-3 Multiple choice plot for visualizing metrics

the mock-up data, with a colourmap, to show the variation of centrality across all network nodes. Similar plots are available for all centrality types and for the LCC (all are node-wise values).

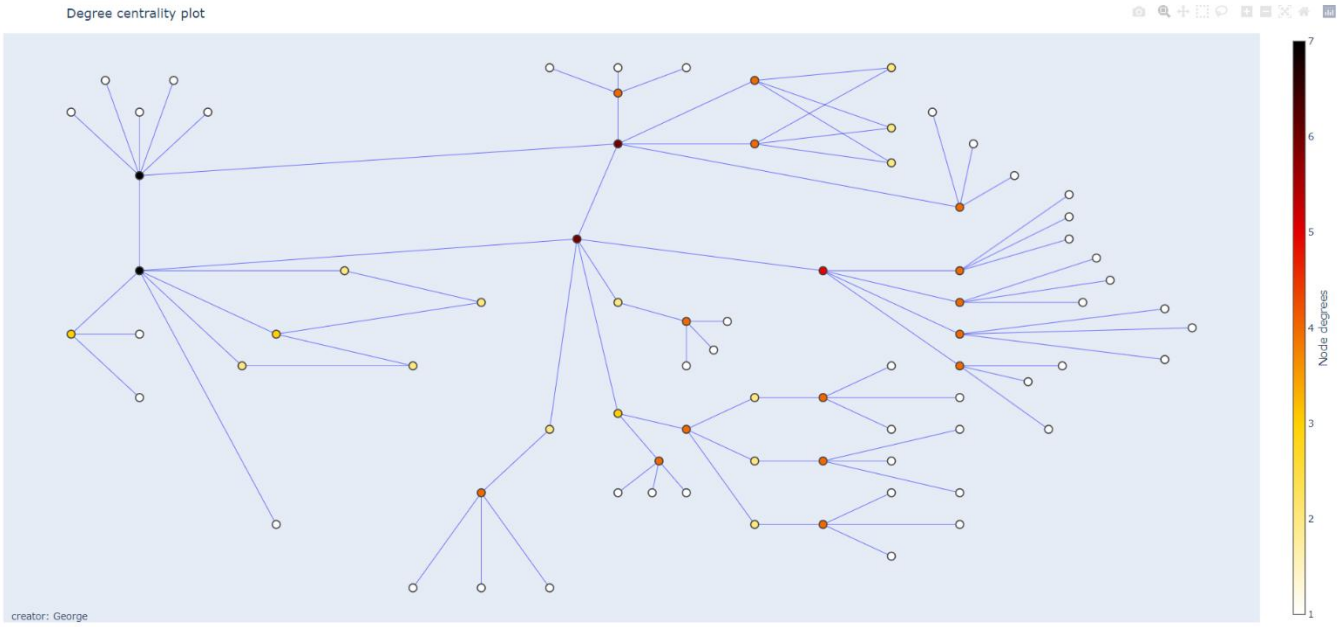

Figure 4-4 Degree centrality plot with colourmap

5 RESEARCH SO FAR AND NEXT STEPS

This Chapter introduces and assesses the methodology and analysis performed in Chapter 3 and overall the research undertaken in the context of this project. Section 5.1 provides the research insights gained from the study, Section 5.2 discusses the implications in the industry and the related stakeholders, Section 5.3 states some of the limitations the current implementation might encounter in the future, while Section 5.4 gives some directions on the continuation of this work. Section 5.5 enhances these directions for more rework-specific avenues and Section 5.6 discusses the possibilities for application to neighbouring industries. Finally, Section 5.7 gives an overall conclusion to this thesis, discussing the value of this research.

5.1 RESEARCH INSIGHTS

This research set out to develop a novel framework for understanding and managing naval ship design processes through the lens of bipartite dynamic social networks. The primary objectives of representing the complex socio-technical nature of design processes, capturing temporal dynamics, and identifying rework patterns have been achieved. The model's application to three industrial case studies has validated its utility and demonstrated its capacity to reveal insights that would remain hidden in traditional process models.

Several key lessons emerged from this research. First, the critical importance of human-task interactions in naval design processes was confirmed, with evidence that these interactions significantly influence design outcomes beyond what technical dependencies alone would predict. Second, the evolutionary nature of design networks became apparent, with distinct phases identifiable through changes in network metrics like clustering coefficients and centrality measures. Third, the potential for early identification of rework triggers through monitoring of network structural changes before they manifest as actual design problems was demonstrated across the case studies. Finally, the value of temporal analysis in identifying critical knowledge nodes and potential communication bottlenecks in the design process was established through the time-dependent graph approach.

This research has made several significant theoretical contributions to both network science and engineering design fields. The integration of social network theory with engineering design represents a

novel theoretical fusion that extends social network theory into a domain traditionally dominated by process-centric frameworks (GANTT, PERT diagrams etc.). This integration provides a new theoretical view for understanding design as a human-informed activity rather than a purely technical one, challenging conventional perspectives that often separate human and technical aspects of design. The development of temporal network dynamics in design processes advances network theory by demonstrating how network properties evolve in engineering design contexts. This extends previous static network models by incorporating the critical dimension of time, allowing for analysis of how network structures form, evolve, and dissolve throughout the design lifecycle. This temporal perspective has revealed patterns of knowledge flow and collaboration that remain invisible in static models. The research has also contributed to theory by developing a formal characterization of rework in terms of network properties.

The visualization approaches developed contribute to information visualization theory by addressing the specific challenges of representing complex temporal interactions between human actors and design activities. These visualization methods extend beyond conventional network diagrams to incorporate temporal navigation and multi-layered information display, enhancing theoretical approaches to complex system visualization.

The primary theoretical frameworks extended by this work include Complex Adaptive Systems theory, Social Network Analysis, and Engineering Design Process theory. However, limitations in these theoretical extensions should be acknowledged. Causal inference between network properties and design outcomes remains challenging, requiring further theoretical development to establish stronger predictive models. The integration of knowledge quality aspects into the network model represents another theoretical challenge, as current approaches focus primarily on structural properties rather than qualitative dimensions of knowledge exchange. Additionally, accounting for external factors beyond the network boundaries presents theoretical difficulties that future work must address. Future theoretical work should address these limitations through development of more sophisticated models that integrate qualitative aspects of knowledge and communication alongside the quantitative network metrics presented here. Expanding the theoretical framework to incorporate concepts from organizational learning and knowledge management would strengthen the model's explanatory power.

5.2 IMPLICATIONS FOR INDUSTRY (SHIPYARDS, STAKEHOLDERS)

The developed model and tools offer several practical applications for shipyards engaged in ship design. Design process management represents a primary application area, with the TDG model allowing design managers to monitor the evolution of design processes in real-time. This capability helps identify potential bottlenecks or communication gaps before they impact project timelines, enabling proactive rather than reactive management approaches. Resource allocation optimization represents another valuable application. By identifying critical nodes in the network, shipyards can strategically allocate expertise and resources to high-centrality activities that have the greatest influence on design outcomes. This targeted approach to resource allocation can improve efficiency compared to traditional methods that may not account for the relative importance of different activities within the overall design network. Risk mitigation is enhanced through the model's ability to identify potential rework triggers. This enables proactive risk management by allowing design teams to implement targeted quality control measures for high-risk activities. Rather than applying uniform quality control across all activities, resources can be focused on areas with the highest potential for generating downstream rework.

Knowledge management within shipyards can be improved through visualization of expertise networks. This helps identify potential vulnerabilities in knowledge transfer, particularly during personnel transitions or when key team members are reassigned. The model makes these dependencies explicit, allowing management to develop strategies for preserving critical knowledge despite changes in team composition.

The benefits of this approach extend to various stakeholders in the naval design ecosystem. For design engineers, the model provides improved understanding of how their work interfaces with others and highlights the importance of specific communication channels in preventing rework. This awareness can foster more collaborative approaches and help engineers prioritize communications that are critical to project success. Project managers gain enhanced ability to monitor project progress through network evolution metrics rather than simply tracking individual tasks. This higher-level view of project dynamics enables more strategic decision-making about resource allocation and intervention points. The ability to visualize the project as an evolving network rather than simply a collection of tasks provides managers with new insights into project health and potential risks.

Naval clients and operators benefit from greater visibility into design process dynamics, potentially leading to more informed decision-making regarding change requests and their potential ripple effects. Understanding how design changes propagate through the network can help clients better time and scope their change requests to minimize disruption to the overall design process. Regulatory bodies can utilize the network model to improve traceability of design decisions and identify critical validation points in the design process. This can enhance regulatory oversight by focusing attention on key decision points that have far-reaching implications for vessel compliance and safety. Supply chain partners can achieve better integration into the design network through explicit modeling of their interfaces with the core design team. This visibility can improve coordination of design and procurement activities, reducing delays and miscommunications that often occur at organizational boundaries. The model makes these interfaces explicit, highlighting the importance of effective communication across organizational boundaries.

5.3 LIMITATIONS FOR THE INDUSTRY CONTEXT

The application of bipartite dynamic social networks to naval design faces several limitations that must be understood within the broader industry context. Data availability and quality represent significant constraints, as the model's effectiveness depends on accurate tracking of task assignments and completions. Many shipyards have data management systems that were not designed to capture the types of interactions needed for detailed network analysis, resulting in incomplete or inconsistent data that may limit the model's accuracy. Organizational boundaries within traditional shipyard structures often create artificial divisions that can impede the collection of network data across departments or subcontractors. These boundaries may result in network models that fail to capture important cross-boundary interactions or create artificial disconnections in what should be a continuous design process. The industry's hierarchical structure and separation of design disciplines presents challenges for implementation of network-based approaches that rely on transparent information flow.

These limitations should be viewed in the context of the industry's ongoing digital transformation efforts. As shipyards increasingly adopt digital design tools and integrated data management systems, the foundations for more sophisticated network analysis are being established. This research positions the naval design community to leverage these advancements as they emerge. The move toward digital twins and integrated design environments creates opportunities for more comprehensive data collection that would enhance the value of network-based approaches.

5.4 MODEL ACHIEVEMENTS AND PROPOSED DIRECTIONS

The model developed is able to represent the design process network of a modern naval combatant with the adoption of a social bipartite TDG. It has been applied on 3 cases studies and can evaluate the basic network properties and node centralities. The model based on the data can identify rework based on two-criteria, task completion delays and labour units (metric used within the processes). Here is the first major contribution that can be made to this research. The clarification of the labour unit metric and more importantly the finding of any correlation between delays and labour units is the immediate step towards continuation of this work. Access to a more recent dataset can lead to better understanding of the results of the model in the period 2023-2024, since for intents and purposes, they are treated as projected data in this work. The possession of the newer dataset and application of the model on it, certainly will give valuable insight on the dependencies a task may have on another. Regarding rework an interesting point of investigation on the dataset is the delay of the start of an activity and identifying the reason for this delay. Which task caused this delay? A dependency is created by such an analysis and the most rework inducing tasks could be identified throughout the network not only through their own delay, but by a metric that includes the delays caused on other activities. In this manner, simulation could be very useful, in the sense of including a highly rework-heavy task artificially, then investigating what its effect is on the whole of the design process. This can be treated as an error propagation or cascading failure analysis on the network (Taylor, 1997). Another perspective is to treat rework as a design iteration and then investigate design iterations in the TDG (Wynn & Clarkson, 2018). Based on data availability cost functions could be used so that a third metric is created for evaluating rework.

Several methodological extensions could enhance the value of the current approach for ship applications. Integration of qualitative knowledge attributes represents a promising direction, extending the model to incorporate aspects such as information quality, certainty, or complexity. Current models focus primarily on the existence of connections rather than their qualitative characteristics, limiting their ability to distinguish between different types of knowledge exchange. Incorporating these qualitative dimensions would provide a richer representation of the design process and enable more nuanced analysis of knowledge flows.

Multi-layer network models offer another methodological extension, developing frameworks that simultaneously represent multiple types of relationships as different layers in a multiplex network. These

relationships might include technical dependencies, communication patterns, and organizational structures, all of which influence design outcomes in different ways. By modeling these as interconnected layers rather than collapsing them into a single network, more sophisticated analysis of their interactions becomes possible. Incorporation of Bayesian approaches would improve handling of uncertainty in design decisions and their propagation through the network. Bayesian networks could model conditional dependencies between design decisions and outcomes, providing a more nuanced representation of how uncertainty propagates through the design process. This probabilistic approach would better represent the reality of design decision-making under incomplete information.

Alternative methodologies were considered but not adopted in this research for various reasons. Pure process models such as Design Structure Matrix (DSM) were rejected due to their inability to adequately represent the human dimension of design activities. While these approaches effectively capture technical dependencies, they typically treat human actors as interchangeable resources rather than as individuals with unique expertise and communication patterns. System dynamics approaches were considered for their ability to model feedback loops and emergent behaviors, but were found to lack the granularity needed to identify specific rework triggers at the task level. These approaches are valuable for macro-level analysis but less suited to the detailed task-level analysis required for rework identification.

The bipartite approach was selected precisely because it bridges these alternative perspectives, though at the cost of increased model complexity. This hybrid approach provides a more complete representation of the socio-technical nature of design but requires more sophisticated analysis techniques and data collection methods. Future work might explore ways to simplify this complexity without sacrificing the essential insights that the bipartite approach provides.

The visualisation platform proposed here is only but the starting point of an advanced visualisation tool. The expansion of the tool in terms of operations and the integration of all operations on a single toolset can be viewed as viable aim for the continuation of the work started here and will allow for better network understanding. It should be used in conjunction with the TDG analysis model and not only for visualisation of results and snapshots of the design. A better integrated visualisation option can help in creating simulations for an error propagation analysis. Combining the existing work with an VR environment would be the ideal target technology for communicating the model results at this stage. The scalability of the visualization approach has been tested with networks of up to 300 nodes, beyond which performance

degradation becomes significant. For larger projects, hierarchical visualization approaches that allow "drilling down" into subnetworks will be necessary. Technical improvements in rendering efficiency and data handling would be required to maintain interactive performance with very large networks. This represents a technical challenge that must be addressed for application to the large ship design projects, which may involve thousands of tasks and hundreds of participants.

5.5 REWORK-SPECIFIC PATHS

Future research specifically focused on rework should address several key areas to advance understanding and management of this critical issue. Development of a more detailed typology of rework patterns based on their network signatures and propagation characteristics would enhance our ability to identify and classify different types of rework. Current approaches often treat rework as a uniform phenomenon, but different types of rework likely have distinct network signatures that could enable more targeted interventions. Moving beyond descriptive analysis to predictive models represents another important direction. Predictive models could forecast potential rework based on early network patterns, enabling proactive interventions before problems manifest. This would require longitudinal studies of multiple design projects to identify reliable early indicators of rework potential and development of statistical or machine learning approaches to leverage these indicators.

Studying intervention effectiveness would provide practical guidance for design managers. Research examining how different management interventions impact network structure and subsequent rework patterns could help identify the most effective strategies for rework reduction. This might involve comparative analysis of different coordination mechanisms, review processes, or communication structures and their effects on network evolution and rework incidence.

Examination of cultural and organizational factors influencing communication patterns and rework propensity represents a final important direction. The social aspects of network formation are influenced by organizational culture, incentive structures, and leadership approaches. Understanding these influences would help organizations create environments that naturally foster effective network structures and reduce rework potential.

5.6 EXTENSION TO OTHER INDUSTRIES

The methodology developed here has potential applications beyond naval design to other complex engineering domains. Aerospace design represents a natural extension, with similar complexity and long development cycles making this methodology applicable to aircraft design processes. The multi-disciplinary nature of aerospace design, involving numerous engineering specialties and extended design timeframes, creates network structures with similar characteristics to naval design. Adaptation would require domain-specific knowledge capture but could leverage the same fundamental network modeling approach. Civil infrastructure projects face comparable challenges in coordinating diverse expertise across extended timeframes. Large-scale projects such as bridges, airports, or urban developments involve numerous stakeholders and design disciplines that must coordinate effectively over multi-year timeframes. The network modeling approach could help identify critical interfaces and potential communication failures in these complex projects.

Software development, particularly in embedded systems where hardware and software interfaces create complex dependencies, represents another potential application domain. The increasing complexity of software systems and their integration with hardware components creates networks of dependencies that share many characteristics with physical engineering domains. The methodology could help software organizations better manage these dependencies and identify potential sources of rework. Adaptation to these domains would require domain-specific knowledge capture methods and calibration of network metrics to reflect different patterns of collaboration and design evolution. Each domain has unique characteristics that would influence network formation and evolution, requiring adaptation of the modeling approach to account for these differences. Cross-domain applications would also provide opportunities to validate the generalization of the network metrics and identify universal patterns in complex design processes. In conclusion, if a process can be represented as a graph and involve iterations due to errors can be treated as a case of the proposed methodology.

5.7 CONCLUSION

This research has demonstrated the value of bipartite dynamic social networks as a novel model for understanding and managing ship design processes. By explicitly representing both the technical and social dimensions of design activities and capturing their evolution over time, the model provides insights

that traditional approaches cannot offer. The integration of social network theory with engineering design concepts has created a more holistic framework for analyzing design processes that acknowledges their fundamentally human/task/interface-nature. The application to industrial case studies has validated the approach's practical utility while also revealing methodological challenges that future research must address. While limitations exist in data availability and integration with existing tools, the approach shows significant promise for improving design process management and reducing rework in naval design. The positive feedback from industrial experts suggests that the approach addresses real needs within the naval design community and offers practical value alongside its theoretical contributions.

Future research should focus on refining the methodological approach, extending the visualization capabilities, and adapting the framework to other domains and phases of naval projects. Particularly promising directions include the development of predictive models for rework, integration of qualitative knowledge attributes, and creation of multi-layer network representations that can simultaneously capture different types of relationships within the design process. Extensions to other phases of the naval vessel lifecycle would expand the approach's value beyond design into construction, operation, and disposal.

The theoretical and practical contributions of this work lay a foundation for a more sophisticated understanding of naval design as a complex socio-technical system, with significant implications for how design processes are structured, managed, and improved in the future. By revealing the complex interplay between human expertise and technical tasks, this research advances both our understanding of design processes and our ability to manage them effectively in an increasingly complex engineering environment.

BIBLIOGRAPHY

Albert, R. & Barabasi, A., 2002. Statistical mechanics of complex networks. *Review of Modern Physics,* 74(1), pp. 47-97.

Albert, R., Jeong, H. & Barabasi, A., 2000. Error and attack tolerance of complex networks. *Nature,* 406(1), pp. 378-382.

Anagnostopoulos, G. & Kaklis, P., 2023. Naval ship design-process analysis through dynamic social networks. *Ship Technology Research,* 71(2), pp. 190-198.

Battiston, S., Bonabeau, E. & Weisbuch, G., 2003. Emergence of Complexity in Financial Networks. *Lecture Notes in Computer Science,* 650(1), pp. 99-110.

Bhadra, S. & Ferreira, A., 2003. Complexity of Connected Components in Evolving Graphs and the Computation of Multicast Trees in Dynamic Networks. In: S. Pierre, ed. *Ad-Hoc, Mobile, and Wireless Networks.* Berlin: Springer, pp. 259-270.

Bonacich, P., 1972. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology,* 2(1), pp. 113-120.

Borgatti, S. P. & Halgin, D., 2014. Analyzing affiliation networks. In: *The SAGE Handbook of Social Network Analysis.* s.l.:SAGE Publications Ltd.

Borgatti, S. P. & Halgin, D. S., 2014. Analyzing Affiliation Networks. In: U. Flick, ed. *The SAGE Handbook of Social Network Analysis.* -: SAGE Publication Ltd, pp. -.

Bornholdt, S. & Schuster, H. G., 2002. *Handbook of Graphs and Networks: From the Genome to the Internet.* 1sy ed. -: Wiley.

Braha, D. & Bar-Yam, Y., 2009. Time-Dependent Complex Networks: Dynamic Centrality, Dynamic Motifs, and Cycles of Social Interactions. In: T. Gross & H. Sayama, eds. *Adaptive Networks.* Massaschusetts: Springer, pp. 39-50.

Chen, X., Zhang , C. & Ge, B., 2017. Temporal Query Processing in Social Network. *Journal of Intelligent Information Systems,* 49(2), pp. 147-166.

Craig, A. B., 2013. *Understanding Augmented Reality: Concepts and Applications.* 1st ed. Waltham, MA: Morgan Kaufmann.

Dahui, W., Li, Z. & Zengru, D., 2005. Bipartite Producer-Consumer Networks and the Size Distribution of Firms. *Physics and Society,* 363(2), pp. 359-366.

Dai, W., Chu, J., Maropoulos, P. G. & Zhao, Y., 2014. Research on Rework Strategies for Reconfigurable Manufacturing System Considering Mission Reliability. *Procedia CIRP,* 25(1), pp. 199-204.

Delling, D. & Wagner, D., 2009. Time-Dependent Route Planning. In: R. H. Mohring, ed. *Robust and Online Large-Scale Optimization.* Berlin: Springer, pp. 207-230.

Euler, L., 1736. Solutio problematis ad geometriam situs pertinentis.. *Commentarii Academiae Scientiarum Imperialis Petropolitanae,* I(8), pp. 128-140.

Festinger, L., 1983. *The Human Legacy.* 1st ed. New York: Columbia University Press.

Foschini, L., Hershberger, J. & Suri, S., 2011. On the Complexity of Time-Dependent Shortest Paths. *Algorithmica,* 68(4), pp. 327-341.

Foschini, L., Hershberger, J. & Suri, S., 2011. *On the Complexity of Time-Dependent Shortest Paths.* -, Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms.

Garlaschelli, D. et al., 2005. The scale-free topology of market investments. *Physica A,* 350(2-4), pp. 491-499.

Giustiniano, L., Cunha, M. P. e. & Clegg, S., 2016. Organizational zemblanity. *European Management Journal,* 34(1), pp. 7-21.

Guillame, J.-L., Latapy, M. & Le-Blond, S., 2004. Statistical analysis of a P2P query graph based on degrees and their time-evolution. *6th International Workshop on Distributed Computing,* 1(1), pp. 126-137.

Guillaume, J. L. & Latapy, M., 2004. Bipartite structure of all complex networks. *Information Processing Letters,* 90(5), pp. 215-221.

Guillaume, J. L. & Latapy, M., 2006. Bipartite graphs as models of complex networks. *Physica A: Statistical Mechanics and its Applications,* 371(2), pp. 795-813.

Han, S., Love, P. & Pena-Mora, F., 2013. A system dynamics model for assessing the impacts of design errors in construction projects. *Mathematical and Computer Modelling,* 57(1), pp. 2044-2053.

Hegazy, T., Said, M. & Kassab, M., 2011. Incorporating rework into construction schedule analysis. *Automation in Construction,* 20(8), pp. 1051-1059.

Huang, S., Fu, A. W.-C. & Liu, R., 2015. Minimum spanning trees in Temporal Graphs. *SIGMOD 15,* 1(1), pp. 419-430.

Hwang, B.-G., Thomas, S. R., Haas, C. T. & Caldas, C. H., 2009. Measuring the Impact of Rework on Construction Cost Performance. *Construction Engineering and Management,* 135(3), pp. 187-198.

Ianmitchi, A., Ripeanu, M., Santos-Neto, E. & Foster, I., 2010. The Small World of File Sharing. *IEEE Transactions on Parallel and Distributed Systems,* 22(7), pp. 1120-1134.

Katz, L., 1953. A new status index derived from sociometric analysis. *Psychometrika,* 18(1), pp. 39-43.

Kim, Y.-C., 1998. A STRUCTURAL ANALYSIS ON FIRM-MARKET AFFILIATION NETWORKS IN THE KOREAN SYSTEM INTEGRATION INDUSTRY. *Development and Society,* 27(2), pp. 101-116.

Kogut, B., Urso, P. & Walker, G., 2007. Emergent Properties of a New Financial Market: American Venture Capital Syndication, 1960–2005. *Management Science,* 53(7), pp. 1181-1198.

Kogut, B. & Walker, G., 2003. Restructuration ou d´esint´egration du r´eseau des firmes allemandes?. *G´erer et Comprendre,* 74(1), pp. 19-34.

Korf, R. E., 1985. Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence,* 1(1), pp. 97-109.

Latapy, M., Magnien, C. & Del Vecchio, N., 2008. Basic notions for the analysis of large two-mode networks. *Social Networks,* 30(1), pp. 31-48.

Latapy, M., Magnien, C. & Del Vecchio, N., 2008. Basic notions for the analysis of large two-mode networks. *Social Networks,* 30(1), pp. 31-48.

Le Fessant, F., Handurukande, S., Kermarrec, A. & Massoulie, L., 2004. Clustering in Peer-to-Peer File Sharing Workloads. *Lecture Notes in Computer Science,* -(-), pp. -.

LeBlond, S., Guillame, J.-L. & Latapy, M., 2005. Clustering in P2P exchanges and consequences on performances. *LNCS, proceedings of the 4-th International workshop on Peer-to-Peer Systems IPTPS'05,* - (-), pp. - .

Love, P. E., 2002. Influence of project type and procurement method on rework costs in building construction projects. *Construction Engineering and Management,* 128(1), pp. 18-29.

Love, P. E., Smith, J., Ackermann, F. & Irani, Z., 2019. Making sense of rework and its unintended consequence in projects: The emergence of uncomfortable knowledge. *International Journal of Project Management,* 37(3), pp. 501-516.

Love, P. et al., 2018. Reduce rework, improve safety: an empirical inquiry into the precursors to error in construction. *Production Planning and Control,* 29(5), pp. 353-366.

Marchetti-Bowick, M., Yin, J., Howrylak, J. A. & Xing, E. P., 2016. A time-varying group sparse additive model for genome-wide association studies of dynamic complex traits. *Bioinformatics,* 32(19), pp. 2903-2910.

Milgram, S., 1967. The Small-world problem. *Psychology Today,* 1(1), pp. 61-67.

Milgram, S., 1967. The Small-World Problem. *Psychology Today,* 2(1), pp. 60-67.

Moinet, A., Starnini, M. & Pastor-Sattoras, R., 2015. Burstiness and Aging in Social Temporal Networks. *Physical Review Letters,* 114(10), p. 108701.

Moreno, J. L., 1934. *Who Shall Survive?.* 1st ed. Washington, D.C: Nervous and Mental Disease Publishing Company.

Newman, M. E., 2001. Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Physical review E,* Volume 64.

Newman, M. E., 2003. The Structure and Function of Complex Networks. *SIAM Review,* 45(2), pp. 167-256.

Newman, M. E., 2006. *The mathematics of networks,* Michican: University of Michigan.

Newman, M. E., 2010. *Networks : an introduction.* 2nd ed. s.l.:Oxford University Press.

Orda, A. & Rom, R., 1990. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM),* 37(3), pp. 607-625.

Page, L., 1997. Method for node ranking in a linked database. *United States Patents,* 1(1), pp. 1-12.

Piccolo, S. A., Lehmann, S. & Maier, A., 2018. Design Process Robustness: A Bipartite Network Analysis Reveals the Central Importance of People. *Design Science,* 4(1), pp. 1-29.

Piccolo, S., Lehmann, S. & Maier, A., 2018. Design process robustness: A bipartite network analysis reveals the central importance of people. *Design Science,* 4(1).

Przytycka, T. M., Singh, M. & Slonim, D. K., 2010. Toward the dynamic interactome: it's about time. *Briefings in bioinformatics,* 11(1), pp. 15-29.

Qiu, X. et al., 2016. Effects of time-dependent diffusion behaviors on the rumor spreading in social networks. *Physics Letters A,* 380(24), pp. 2054-2063.

Riolo, C., Koopman, J. & Chick, S., 2001. Methods and measures for the description of epidemiologic contact networks. *Journal of Urban Health,* 78(3), pp. 446-457.

Robins, G. & Alexander, M., 2004. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory,* Volume 10, pp. 69-94.

Robins, G. & Alexander, M., 2004. Small Worlds Among Interlocking Directors: Network Structure and Distance in Bipartite Graphs. *Computational & Mathematical Organization Theory,* 10(1), pp. 69-94.

Santos, H., Pereira, M. T., Silva, F. J. & Ferreira, L. P., 2018. A Novel Rework Costing Methodology Applied To a Bus Manufacturing Company. *Procedia Manufacturing,* 17(1), pp. 631-639.

Sherali, H. D., Ozbay, K. & Subramanian, S., 1998. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks,* 31(4), pp. 259-272.

Taylor, J. R., 1997. *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements.* 2nd ed. - : University Science Books.

Voulgaris, S. A., Kermarrec, M. & Massoulie, L., 2004. Exploiting semantic proximity in peer-to-peer content searching. *Proceedings. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems,* 1(1), pp. 238-243.

Wang, Y., Yuan, Y., Ma, Y. & Wang, G., 2019. Time-Dependent Graphs: Definitions, Applications, and Algorithms. *Data Science and Engineering,* Volume 4, pp. 352-366.

Wang, Y., Yuan, Y., Ma, Y. & Wang, G., 2019. Time-Dependent Graphs: Definitions, Applications, and Algorithms. *Data Science and Engineering,* 4(1), pp. 352-366.

Wasserman, S. & Katherine, F., 1994. *Social network analysis: Methods and applications.* 1st ed. US: Cambridge University Press.

Watts, D. J. & Strogatz, S. H., 1998. Collective dynamics of 'small-world' networks. *Nature,* 393(1), pp. 440-442.

Wolfartsberger, J., 2019. Analyzing the potential of Virtual Reality for engineering design review. *Automation in Construction,* 104(1), pp. 27-37.

Wu, H. et al., 2014. Path problems in temporal graphs. *Proceedings of the VLDB Endowment,* 7(9), pp. 721-732.

Wynn, D. C. & Clarkson, J. P., 2018. Process models in design and development. *Research in Engineering Design,* 29(1), pp. 161-202.

Yang, L. & Zhou, X., 2017. Optimizing on-time arrival probability and percentile travel time for elementary path finding in time-dependent transportation networks: Linear mixed integer programming reformulations. *Transportation Research Part B: Methodological,* 96(1), pp. 68-91.

Zhang, Y. & Kwok, T.-H., 2018. Design and Interaction Interface using Augmented Reality for Smart Manufacturing. *Procedia Manufacturing,* 26(1), pp. 1278-1286.

Zhu, Z., Liu, C. & Xu, X., 2019. Visualisation of the Digital Twin data in manufacturing by using Augmented Reality. *Procedia CIRP,* 81(1), pp. 898-903.

APPENDIX A – MOCK-UP DATA

Letters denote design tasks and numbers denote individuals participating in these tasks. Support teams are grouped together. The edges are defined by source, destination and activity time interval. The time interval is $t(e) = [t_e^{start}, t_e^{end}]$, $e = 1,2, \dots, N_e$, where N_e is the number of edges.

Appendix table 1 – Node legend

A	Setting owner's or operator's requirements	4	systems direct
B	Comparison with previous designs	5	cost evaluator
C	Basic dimensions and design ratios	9	chief designer
D	General arrangement	10	consultant
E	Lines plan	11	owner/operator
F	Hull and superstructure design	12-16	team that did a previous design
G	Main Engine, generators and powering	17-19	team for dimensions and GA
H	Piping, mechanical and electrical systems	20-22	team for linesplan
I	Creating an equipment list	23-25	team for hull and superstructure
J	Compliance with design standards	26-28	powering team
K	Weight and CG estimation	29	piping designer
L	Hydrostatics and stability	30-32	piping team
M	Maneuvrability and seakeeping	33	weapon systems designer
N	Production Technology	34-36	weapon systems team

O	Maintenance and decommissioning technology	37	mechanical and electrical systems designer
P	Cost estimation	38-40	mechanical and electrical systems team
Q	Design Process Monitoring	41-43	equipment suppliers
R	Piping systems	44-46	weight calculation team
S	Weapon systems	47-49	hydrostatics and stability team
T	Mechanical and electrical systems	50-52	seakeeping team
U	Design process finalisation	53	production director
V	Survivability	54	maintenance and operations director
1	hullform director	55-57	cost evaluation team
2	performance director	58-60	survivability team
3	powering director		

Appendix table 2 – Edge information

Edge Number	*Source*	*Destination*	*Active time interval*	
			Start	*End*
1	11	A	0	1
2	10	A	0	1
3	9	A	0	1

4	A	9	0	1
5	B	9	2	8
6	12	B	2	8
7	13	B	2	8
8	14	B	2	8
9	15	B	2	8
10	16	B	2	8
11	B	1	2	8
12	Q	1	8	30
13	1	Q	8	30
14	1	E	13	16
15	E	1	13	16
16	20	E	13	16
17	21	E	13	16
18	22	E	13	16
19	F	1	16	30
20	1	F	16	30
21	23	F	16	30
22	24	F	16	30
23	25	F	16	30
24	1	C	8	13

25	1	D	13	16
26	C	1	8	13
27	D	1	13	16
28	17	C	8	13
29	17	D	13	16
30	18	C	8	13
31	18	D	13	16
32	19	C	8	13
33	19	D	13	16
34	Q	2	30	44
35	2	Q	30	44
36	K	2	30	44
37	2	K	30	44
38	44	K	30	44
39	45	K	30	44
40	46	K	30	44
41	L	2	30	44
42	2	L	30	44
43	47	L	30	44
44	48	L	30	44
45	49	L	30	44

46	M	2	30	44
47	2	M	30	44
48	50	M	30	44
49	51	M	30	44
50	52	M	30	44
51	V	2	30	44
52	2	V	30	44
53	58	V	30	44
54	59	V	30	44
55	60	V	30	44
56	3	Q	30	35
57	Q	3	30	35
58	3	G	30	35
59	G	3	30	35
60	26	G	30	35
61	27	G	30	35
62	28	G	30	35
63	Q	5	51	72
64	5	Q	51	72
65	P	5	51	72
66	5	P	51	72

67	55	P	51	72
68	56	P	51	72
69	57	P	51	72
70	Q	4	30	51
71	4	Q	30	51
72	I	4	44	51
73	4	I	44	51
74	41	I	44	51
75	42	I	44	51
76	43	I	44	51
77	4	H	30	44
78	H	4	30	44
79	29	H	30	40
80	H	29	30	40
81	29	R	30	40
82	R	29	30	40
83	30	R	30	40
84	31	R	30	40
85	32	R	30	40
86	33	H	30	44
87	H	33	30	44

88	33	S	30	44
89	S	33	30	44
90	34	S	30	44
91	35	S	30	44
92	36	S	30	44
93	H	37	30	40
94	37	H	30	40
95	T	37	30	40
96	37	T	30	40
97	38	T	30	40
98	39	T	30	40
99	40	T	30	40
100	9	N	44	58
101	N	9	44	58
102	53	N	44	58
103	N	53	44	58
104	9	J	44	65
105	53	J	44	65
106	54	J	44	65
107	9	O	44	72
108	O	9	44	72

109	54	O	44	72
110	O	54	44	72
111	9	Q	2	72
112	Q	9	2	72
113	9	U	72	73

APPENDIX B – VISUALISATION TOOL CODES

Preparation script

```
"""
module for preparing the data given
"""

import pandas as pd
import datetime
import scipy
import networkx as nx
import matplotlib.pyplot as plt

from networkx.algorithms import bipartite
from networkx.algorithms.bipartite.basic import density

#%%  import the dataframe
df_source=pd.read_excel("SHIP1.xlsx")
original_elements=df_source.size
original_rows=df_source.shape[0]
original_columns=df_source.shape[1]
# print(df_source.shape)



#%% drop unnecessary columns, rows etc.

# columns
df=df_source.drop(['Project ID','Variance - BL Project Start Date','Variance - BL
Project Finish Date','T26 - IPT Reporting'],inplace=False,axis=1)

# rows
df.dropna(axis=0,inplace=True)

#sort values with BL Start and conform the dates to the same format
df.sort_values(by='BL Project Start',inplace=True)
df['BL Project Start']=pd.to_datetime(df['BL Project Start']).dt.date
df['BL Project Finish']=pd.to_datetime(df['BL Project Finish']).dt.date
df['Start']=pd.to_datetime(df['Start']).dt.date
df['Finish']=pd.to_datetime(df['Finish']).dt.date
df=df.reset_index()
```

```
# convert individuals involved to integers
df['L3 WBS CAM']=df['L3 WBS CAM'].astype(int)



#import the datasheet for the individuals involved in the design
#import the datasheet for the individuals involved in the design
df_ind=pd.read_excel('Individuals.xlsx')
dfi=df_ind.drop(['Number and name'],inplace=False,axis=1)
# dfi.tail()

# %%
# Create static model from which the dynamic will be lighted based on time
G=nx.Graph()

# create activities nodes from activity data
for i in df.index:
    G.add_node(df.iloc[i]['Activity ID'],ActivityName=df.iloc[i]['Activity Name'],


               PlotShape='s',PlotColor='g')

# create individual nodes from individual data
for i in dfi.index:
    G.add_node(dfi.iloc[i]['Number'],IndividualCode=dfi.iloc[i]['Code'],
               IndividualName=dfi.iloc[i]['Name'],PlotShape='o',PlotColor='r')

# G.nodes() -->list of nodes
# G.nodes(data=True) --> list if nodes with attributes
# G._node[65] --> node feature vector for node 65
# G._node[65]['IndividualName'] --> access to specific attribute

# create edges from data
for i in df.index:
    G.add_edge(df.iloc[i]['Activity ID'],df.iloc[i]['L3 WBS CAM'],
               SetTimeDomain=[df.iloc[i]['BL Project Start'],df.iloc[i]['BL Project
Finish']],
               EndTimeDomain=[df.iloc[i]['Start'],df.iloc[i]['Finish']])
# G.edges() --> list of edges
# G.edges(data=True)--> list of edges with attributes
# G.edges(data=name of an attribute --> list of edges with only the selected attribute

# check for bipartite graph and get some bipartite properties, and other logical
expressions
```

```
if bipartite.is_bipartite(G):
    print('The graph is bipartite')
else:
    print('The graph is NOT bipartite, check initial data')

nx.is_directed(G)
print('Number of nodes: ', G.number_of_nodes())
print('Number of edges: ',G.number_of_edges())
```

Dynamic visualisation script

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 26 12:47:33 2020

@author: G. Anagno

"""


import dash
from dash import dcc
from dash import html
import networkx as nx
import plotly.graph_objs as go

import pandas as pd
from colour import Color
from datetime import datetime
from textwrap import dedent as d
import json

# import the css template, and pass the css template into dash
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.title = "Test_network"

YEAR = [2010, 2019]
ACCOUNT = "A0001"

##############################################################################################
#########################################################################
```

```
def network_graph(yearRange, NodeToSearch):

    edge1 = pd.read_csv('edge1.csv')
    node1 = pd.read_csv('node1.csv')

    edge1['Datetime'] = ""  # add empty Datetime column to edge1 dataframe
    accountSet = set()
    for index in range(0, len(edge1)):
        edge1['Datetime'][index] = datetime.strptime(
            edge1['Date'][index], '%d/%m/%Y')
        if edge1['Datetime'][index].year < yearRange[0] or
edge1['Datetime'][index].year > yearRange[1]:
            edge1.drop(axis=0, index=index, inplace=True)
            continue
        accountSet.add(edge1['Source'][index])
        accountSet.add(edge1['Target'][index])

    # to define the centric point of the networkx layout
    shells = []
    shell1 = []
    shell1.append(NodeToSearch)
    shells.append(shell1)
    shell2 = []
    for ele in NodeSet:
        if ele != NodeToSearch:
            shell2.append(ele)
    shells.append(shell2)

    G = nx.from_pandas_edgelist(edge1, 'Source', 'Target', [
                                'Source', 'Target', 'Amt', 'Date'],
create_using=nx.MultiDiGraph())
    nx.set_node_attributes(G, node1.set_index('node')[
                           'Name'].to_dict(), 'Name')
    nx.set_node_attributes(G, node1.set_index(
        Node)['Type'].to_dict(), 'Type')
    # pos = nx.layout.spring_layout(G)
    # pos = nx.layout.circular_layout(G)
    # nx.layout.shell_layout only works for more than 3 nodes
    if len(shell2) > 1:
        pos = nx.drawing.layout.shell_layout(G, shells)
    else:
        pos = nx.drawing.layout.spring_layout(G)
```

```
    for node in G.nodes:
        G.nodes[node]['pos'] = list(pos[node])

    if len(shell2) == 0:
        traceRecode = []  # contains edge_trace, node_trace, middle_node_trace

        node_trace = go.Scatter(x=tuple([1]), y=tuple([1]),
text=tuple([str(NodeToSearch)]), textposition="bottom center",
                                mode='markers+text',
                                marker={'size': 50, 'color': 'LightSkyBlue'})
        traceRecode.append(node_trace)

        node_trace1 = go.Scatter(x=tuple([1]), y=tuple([1]),
                                 mode='markers',
                                 marker={'size': 50, 'color': 'LightSkyBlue'},
                                 opacity=0)
        traceRecode.append(node_trace1)

        figure = {
            "data": traceRecode,
            "layout": go.Layout(title='Dynamic Visualization Giorgos',
showlegend=False,
                                margin={'b': 40, 'l': 40, 'r': 40, 't': 40},
                                xaxis={'showgrid': False, 'zeroline': False,
                                       'showticklabels': False},
                                yaxis={'showgrid': False, 'zeroline': False,
                                       'showticklabels': False},
                                height=600
                                )}
        return figure

    traceRecode = []  # contains edge_trace, node_trace, middle_node_trace
    ################################################################################
################################################################################
    colors = list(Color('lightcoral').range_to(
        Color('darkred'), len(G.edges())))
    colors = ['rgb' + str(x.rgb) for x in colors]

    index = 0
    for edge in G.edges:
        x0, y0 = G.nodes[edge[0]]['pos']
        x1, y1 = G.nodes[edge[1]]['pos']
        weight = float(G.edges[edge]['Amt']) / \
            max(edge1['Amt']) * 10
```

```
        trace = go.Scatter(x=tuple([x0, x1, None]), y=tuple([y0, y1, None]),
                           mode='lines',
                           line={'width': weight},
                           marker=dict(color=colors[index]),
                           line_shape='spline',
                           opacity=1)
        traceRecode.append(trace)
        index = index + 1
    ################################################################################################
################################################################################
    node_trace = go.Scatter(x=[], y=[], hovertext=[], text=[], mode='markers+text',
textposition="bottom center",
                            hoverinfo="text", marker={'size': 50, 'color':
'LightSkyBlue'})

    index = 0
    for node in G.nodes():
        x, y = G.nodes[node]['pos']
        hovertext = "Name: " + str(G.nodes[node]['Name']) + "<br>" + "NodeType: " +
str(
            G.nodes[node]['Type'])
        text = node1['Node'][index]
        node_trace['x'] += tuple([x])
        node_trace['y'] += tuple([y])
        node_trace['hovertext'] += tuple([hovertext])
        node_trace['text'] += tuple([text])
        index = index + 1

    traceRecode.append(node_trace)
    ################################################################################################
################################################################################
    middle_hover_trace = go.Scatter(x=[], y=[], hovertext=[], mode='markers',
hoverinfo="text",
                                    marker={'size': 20,
                                            'color': 'LightSkyBlue'},
                                    opacity=0)

    index = 0
    for edge in G.edges:
        x0, y0 = G.nodes[edge[0]]['pos']
        x1, y1 = G.nodes[edge[1]]['pos']
        hovertext = "From: " + str(G.edges[edge]['Source']) + "<br>" + "To: " + str(
            G.edges[edge]['Target']) + "<br>" + "Task info: " + str(
            G.edges[edge]['Amt']) + "<br>" + "Date: " + str(G.edges[edge]['Date'])
```

```
        middle_hover_trace['x'] += tuple([(x0 + x1) / 2])
        middle_hover_trace['y'] += tuple([(y0 + y1) / 2])
        middle_hover_trace['hovertext'] += tuple([hovertext])
        index = index + 1

    traceRecode.append(middle_hover_trace)
    ################################################################################################
################################################################################
    figure = {
        "data": traceRecode,
        "layout": go.Layout(title='Dynamic Visualization', showlegend=False,
hovermode='closest',
                            margin={'b': 40, 'l': 40, 'r': 40, 't': 40},
                            xaxis={'showgrid': False, 'zeroline': False,
                                   'showticklabels': False},
                            yaxis={'showgrid': False, 'zeroline': False,
                                   'showticklabels': False},
                            height=600,
                            clickmode='event+select',
                            annotations=[
                                dict(
                                    ax=(G.nodes[edge[0]]['pos'][0] +
                                        G.nodes[edge[1]]['pos'][0]) / 2,
                                    ay=(G.nodes[edge[0]]['pos'][1] +
G.nodes[edge[1]]['pos'][1]) / 2, axref='x', ayref='y',
                                    x=(G.nodes[edge[1]]['pos'][0] * 3 +
                                       G.nodes[edge[0]]['pos'][0]) / 4,
                                    y=(G.nodes[edge[1]]['pos'][1] * 3 +
G.nodes[edge[0]]['pos'][1]) / 4, xref='x', yref='y',
                                    showarrow=True,
                                    arrowhead=3,
                                    arrowsize=4,
                                    arrowwidth=1,
                                    opacity=1
                                ) for edge in G.edges]
                            )}
    return figure


################################################################################################
################################################################################
# styles: for right side hover/click component
styles = {
    'pre': {
```

```
        'border': 'thin lightgrey solid',
        'overflowX': 'scroll'
    }
}

app.layout = html.Div([
    # Title
    html.Div([html.H1("Process Network Graph")],
             className="row",
             style={'textAlign': "center"}),
    # define the row
    html.Div(
        className="row",
        children=[
            # left side two input components
            html.Div(
                className="two columns",
                children=[
                    dcc.Markdown(d("""
                            **Time Range To Visualize**

                            Slide the bar to define year range.
                            """)),
                    html.Div(
                        className="twelve columns",
                        children=[
                            dcc.RangeSlider(
                                id='my-range-slider',
                                min=2010,
                                max=2019,
                                step=1,
                                value=[2010, 2019],
                                marks={
                                    2010: {'label': '10'},
                                    2011: {'label': '11'},
                                    2012: {'label': '12'},
                                    2013: {'label': '13'},
                                    2014: {'label': '14'},
                                    2015: {'label': '15'},
                                    2016: {'label': '16'},
                                    2017: {'label': '17'},
                                    2018: {'label': '18'},
                                    2019: {'label': '19'}
                                }
```

```
                    ),
                    html.Br(),
                    html.Div(id='output-container-range-slider')
                ],
                style={'height': '300px'}
            ),
            html.Div(
                className="twelve columns",
                children=[
                    dcc.Markdown(d("""
                    **Node to Search**

                    Input the node to visualize.
                    """)),
                    dcc.Input(id="input1", type="text",
                              placeholder="Specify activity"),
                    html.Div(id="output")
                ],
                style={'height': '300px'}
            )
        ]
    ),

    # middle graph component
    html.Div(
        className="eight columns",
        children=[dcc.Graph(id="my-graph",
                            figure=network_graph(YEAR, Node))],
    ),

    # right side two output component
    html.Div(
        className="two columns",
        children=[
            html.Div(
                className='twelve columns',
                children=[
                    dcc.Markdown(d("""
                    **Hover Data**

                    Mouse over values in the graph.
                    """)),
                    html.Pre(id='hover-data', style=styles['pre'])
                ],
```

```
                        style={'height': '400px'}),

                    html.Div(
                        className='twelve columns',
                        children=[
                            dcc.Markdown(d("""
                            **Click Data**

                            Click on points in the graph.
                            """)),
                            html.Pre(id='click-data', style=styles['pre'])
                        ],
                        style={'height': '400px'})
                ]
            )
        ]
    )
])

# callback for left side components


@app.callback(
    dash.dependencies.Output('my-graph', 'figure'),
    [dash.dependencies.Input('my-range-slider', 'value'),
dash.dependencies.Input('input1', 'value')])
def update_output(value, input1):
    YEAR = value
    NODE = input1
    return network_graph(value, input1)



@app.callback(
    dash.dependencies.Output('hover-data', 'children'),
    [dash.dependencies.Input('my-graph', 'hoverData')])
def display_hover_data(hoverData):
    return json.dumps(hoverData, indent=2)


@app.callback(
    dash.dependencies.Output('click-data', 'children'),
    [dash.dependencies.Input('my-graph', 'clickData')])
def display_click_data(clickData):
```

```
    return json.dumps(clickData, indent=2)


if __name__ == '__main__':
    app.run_server(debug=True)
```

Metrics codes

```
import dash
from dash import Dash,dcc,html,Input,Output
from matplotlib.axis import YAxis
from matplotlib.pyplot import cla
import plotly.express as px
from plotly.express import data
import plotly.graph_objects as go
import pandas as pd
import numpy as np

import main

#data
time_period=main.time_period
density=main.density
avg_clust=main.avg_clust
avg_clustU=main.avg_clustU
avg_clustV=main.avg_clustV
trans=main.trans
tasks=main.tasks
roles=main.individuals
connections=main.connections
avg_dis=main.avg_dis
diameter=main.diameter
radius=main.radius
numel_per=main.numel_per
numel_cen=main.numel_cen

resultsSet=list(zip(time_period,density,avg_clust,avg_clustU,avg_clustV,trans,tasks,rol
es,connections,
    avg_dis,diameter,radius,numel_per,numel_cen))
resultNames=['Timeline','Density','Latapy clustering','Activity clustering','Role
clustering',
```

```
    'Robins-Alexander','Activities','Roles','Connections','Average
Distance','Diameter','Radius',
    'Nodes in periphery', 'Nodes in center']
df=pd.DataFrame(resultsSet,columns=resultNames)
# print(df)

fig=go.Figure()


fig.add_trace(go.Scatter(x=main.time_period,y=main.density,name='Graph Density'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.avg_clust,name='Average clustering
(Latapy)'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.avg_clustU,name='Average clustering
for activities'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.avg_clustV,name='Average clustering
for roles'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.trans,name='Robins - Alexander
clustering'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.tasks,name='Number of activities'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.individuals,name='Number of active
roles'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.connections,name='Number of active
connections'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.avg_dis,name='Average Distance'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.diameter,name='Diameter'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.radius,name='Radius'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.numel_per,name='Number of nodes on
the periphery'))
fig.add_trace(go.Scatter(x=main.time_period,y=main.numel_cen,name='Number of nodes in
the center'))

fig.update_layout(
    updatemenus=[go.layout.Updatemenu(
        active=0,
        buttons=list(
            [dict(label='All',
                  method='update',
                  args=[{'visible':[True,True,True,True,True,True,True,True,True,True,T
rue,True,True]},
                        {'title':'All',
                         'showlegend':True}]),
             dict(label='Density',
                  method='update',
```

```
                  args=[{'visible':[True,False,False,False,False,False,False,False,Fals
e,False,False,False,False]},
                        {'title':'Graph Density',
                        'showlegend':True}]),
            dict(label='Latapy clustering',
                  method='update',
                  args=[{'visible':[False,True,False,False,False,False,False,False,Fals
e,False,False,False,False]},
                        {'title':'Average clustering (Latapy)',
                        'showlegend':True}]),
            dict(label='Activities clustering',
                  method='update',
                  args=[{'visible':[False,False,True,False,False,False,False,False,Fals
e,False,False,False,False]},
                        {'title':'Average clustering for activities',
                        'showlegend':True}]),
            dict(label='Roles clustering',
                  method='update',
                  args=[{'visible':[False,False,False,True,False,False,False,False,Fals
e,False,False,False,False]},
                        {'title':'Average clustering for roles',
                        'showlegend':True}]),
            dict(label='Robins-Alexander clustering',
                  method='update',
                  args=[{'visible':[False,False,False,False,True,False,False,False,Fals
e,False,False,False,False]},
                        {'title':'Robins - Alexander clustering',
                        'showlegend':True}]),
            dict(label='Activities',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,True,False,False,Fals
e,False,False,False,False]},
                        {'title':'Number of activities',
                        'showlegend':True}]),
            dict(label='Roles',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,True,False,Fals
e,False,False,False,False]},
                        {'title':'Number of active roles',
                        'showlegend':True}]),
            dict(label='Connections',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,False,True,Fals
e,False,False,False,False]},
```

```
                        {'title':'Number of active connections',
                        'showlegend':True}]),
            dict(label='Distance',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,False,False,Tru
e,False,False,False,False]},
                        {'title':'Average Distance',
                        'showlegend':True}]),
            dict(label='Diameter',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,False,False,Fal
se,True,False,False,False]},
                        {'title':'Diameter',
                        'showlegend':True}]),
            dict(label='Radius',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,False,False,Fal
se,False,True,False,False]},
                        {'title':'Radius',
                        'showlegend':True}]),
            dict(label='Periphery',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,False,False,Fal
se,False,False,True,False]},
                        {'title':'Number of nodes belonging in the graph periphery',
                        'showlegend':True}]),
            dict(label='Center',
                  method='update',
                  args=[{'visible':[False,False,False,False,False,False,False,False,Fal
se,False,False,False,True]},
                        {'title':'Number of nodes belonging in the graph center',
                        'showlegend':True}])
            ]
        )
    )]
)

fig.show()
```

```
import dash
from dash import Dash,dcc,html,Input,Output
from matplotlib.axis import YAxis
from matplotlib.pyplot import cla
import plotly.express as px
from plotly.express import data
import plotly.graph_objects as go
import pandas as pd
import numpy as np

import main

#data
time_period=main.time_period
density=main.density
avg_clust=main.avg_clust
avg_clustU=main.avg_clustU
avg_clustV=main.avg_clustV
trans=main.trans
tasks=main.tasks
roles=main.individuals
connections=main.connections
avg_dis=main.avg_dis
diameter=main.diameter
radius=main.radius
numel_per=main.numel_per
numel_cen=main.numel_cen

resultsSet=list(zip(time_period,density,avg_clust,avg_clustU,avg_clustV,trans,tasks,rol
es,connections,
    avg_dis,diameter,radius,numel_per,numel_cen))
resultNames=['Timeline','Density','Latapy clustering','Activity clustering','Role
clustering',
    'Robins-Alexander','Activities','Roles','Connections','Average
Distance','Diameter','Radius',
    'Nodes in periphery', 'Nodes in center']
df=pd.DataFrame(resultsSet,columns=resultNames)
# print(df)

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app=Dash(__name__,external_stylesheets=external_stylesheets)
app.title="Metrics Evolution in time"
```

```
# options=['Graph Density','Clustering Metrics','Graph Sets','Distance
Measures','Center - Periphery']
# value='Graph Density'
app.layout=html.Div([
    html.Div([
        dcc.Graph(id='my_graph')
    ],className='ten columns'),

    html.Div([
        html.Br(),
        html.Div(id='output_data'),
        html.Br(),
        html.Label(['Choose plot: '],style={'font-weight':'bold','text-
align':'center'}),

        dcc.Dropdown(id='my_dropdown',
            options=[
                {'label':'Density','value':'Density'},
                {'label':'Latapy clustering','value':'Latapy clustering'},
                {'label':'Activities clustering','value':'Activity clustering'},
                {'label':'Roles clustering','value':'Role clustering'},
                {'label':'Robins-Alexander clustering','value':'Robins-Alexander'},
                {'label':'Activities','value':'Activities'},
                {'label':'Roles','value':'Roles'},
                {'label':'Connections','value':'Connections'},
                {'label':'Distance','value':'Average Distance'},
                {'label':'Diameter','value':'Diameter'},
                {'label':'Radius','value':'Radius'},
                {'label':'Periphery','value':'Nodes in periphery'},
                {'label':'Center','value':'Nodes in center'}],

            optionHeight=35,
            value='Density',
            disabled=False,
            multi=True,
            searchable=True,
            placeholder='Select property'
        ),

    ], className='two columns')

])

@app.callback(
```

```
    Output(component_id='my_graph',component_property='figure'),
    [Input(component_id='my_dropdown',component_property='value')]

)


def build_graph(column_chosen):
    ddf=df
    fig=px.line(ddf,x='Timeline',y=column_chosen,height=800)
    fig.update_layout(yaxis={'title':'Selected features'},
        title={'text':'Graph properties in
time','font':{'size':28},'x':0.5,'xanchor':'center'})

    return fig


if __name__=='__main__':
    app.run_server(debug=True)
```

Node-wise plots module

```
"""
Created by G.Anagno on 01/03/2022

Includes visualisation tools and functions
"""

# from __future__ import annotations
from cgi import test
from turtle import bgcolor
from matplotlib.pyplot import figure
from numpy import size
from pyvis.network import Network

import networkx as nx
from networkx.algorithms import bipartite

import pandas as pd
import json
from main import ordered_vector
import static_graph_data as data
```

```
G=data.create_graph()
# [str(i) for i in node_list]
# two bipartite sets
U = bipartite.sets(G)[0]  # tasks
V = bipartite.sets(G)[1]  # roles
E=G.edges()

# %% create pyvis_graph for interactive graphs

'''
for the shapes: ellipse, circle, box have the node label inside
dot,star, triangle, triangleDown,square,diamond have the node label outside
'''
task_color="blue"
task_shape="square"
role_color="red"
role_shape="dot"
edge_color="yellow"

vis_g=Network(height="1000px",width="70%",bgcolor="#222222",
font_color="white",directed=False)

for i in U:
    vis_g.add_node(i,color=task_color,shape=task_shape)

for i in V:
    vis_g.add_node(i,color=role_color,shape=role_shape)

for i in E:
    vis_g.add_edge(i[0],i[1],color=edge_color)
# vis_g.from_nx(G)

"""
algorithms: barnes hut, force atlas2based, hrepulsion for plots

the cleanest seems to be the force atlas2based algorithm
"""
# vis_g.hrepulsion()

vis_g.show_buttons(filter_=["nodes","edges"])

# vis_g.show('graph.html')
```

```
# %%create attribute based plot
import plotly.graph_objects as g_obj

#centrality plot
edge_x=[]
edge_y=[]

# print(G.edges(data=True))
# print(G.nodes(data='pos'))

#create horizontal and vertical vectors for each edge
for i in G.edges():
    x0,y0=G.nodes[i[0]]['pos']
    x1,y1=G.nodes[i[1]]['pos']
    edge_x.append(x0)
    edge_x.append(x1)
    edge_x.append(None)
    edge_y.append(y0)
    edge_y.append(y1)
    edge_y.append(None)

edge_trace=g_obj.Scatter(x=edge_x,y=edge_y,line=dict(width=0.5,
color='blue'),hoverinfo='none',mode='lines+markers')

#get position lists of nodes in x,y
node_x=[]
node_y=[]
for i in G.nodes():
    x,y=G.nodes[i]['pos']
    node_x.append(x)
    node_y.append(y)

node_trace=g_obj.Scatter(x=node_x,y=node_y,mode='markers',hoverinfo='text',
marker=dict(showscale=True,
        # colorscale options
        # 'Greys' | 'YlGnBu' | 'Greens' | 'YlOrRd' | 'Bluered' | 'RdBu' |
        # 'Reds' | 'Blues' | 'Picnic' | 'Rainbow' | 'Portland' | 'Jet' |
        # 'Hot' | 'Blackbody' | 'Earth' | 'Electric' | 'Viridis' |
        colorscale='Hot',reversescale=True,color=[],size=10,colorbar=dict(
        thickness=15,title='Node degrees',xanchor='left',titleside='right'),
```

```
        line_width=1.5))

#find adjacencies (degree centrality)
nodelist=G.nodes()
node_adjacencies=[]
node_text=[]

for i,adjs in enumerate(G.adjacency()):
    node_adjacencies.append(len(adjs[1]))
    node_text.append('Node: '+str(adjs[0])+', # of neighbors: '+str(len(adjs[1])))

node_trace.marker.color=node_adjacencies
node_trace.text=node_text

#create the plot

plot=g_obj.Figure(data=[edge_trace,node_trace],layout=g_obj.Layout(title='Degree
centrality plot',
    titlefont_size=16,showlegend=False,hovermode='closest',margin=dict(b=20,l=5,r=5,t=4
0),
    annotations=[dict(text='creator:
George',showarrow=False,xref='paper',yref='paper',x=0.005,y=-0.002)],
    xaxis=dict(showgrid=False,zeroline=False,showticklabels=False),
    yaxis=dict(showgrid=False,zeroline=False,showticklabels=False)))

plot.show()


# %% dynamic plot with timebar

#import libraries
import dash
from dash import dcc,Dash,html
from dash import html
from colour import Color
from datetime import datetime
from textwrap import dedent as ddt

# Import the css template into dash
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = Dash(__name__, external_stylesheets=external_stylesheets)
app.title = "Dynamic Visualisation"

############ data analysis #################
```

```
# set timeline
timeline=ordered_vector(0,73+1)
focus_node='A'

def network_for_interactive(timeline,central_node):

    start=timeline[0]
    end=timeline[-1]
    timeRange=[start,end]

    #data read from excel
    nodelist=pd.read_csv('node_data.csv')
    edgelist=pd.read_csv('edge_data.csv')

    edgelist['Date']="" #extra column for future use
    node_set=set()    #get nodes from edges (should be the same as the nodelist in
number)

    for i in range(0,len(edgelist)):
        edgelist['Date'][i]=edgelist['Start date'][i]
        if edgelist['Date'][i]<timeline[0] or edgelist['Date'][i]>timeline[-1]:
            edgelist.drop(axis=0,index=i,inplace=True)
            continue
        node_set.add(edgelist['Source'][i])
        node_set.add(edgelist['Target'][i])

    #define center for the visual plot
    n_list=[]
    n_list1=[]
    n_list1.append(central_node)
    n_list.append(n_list1)
    n_list2=[]
    for i in node_set:
        if i != central_node:
            n_list2.append(i)
    n_list.append(n_list2)
    #separated central node from the rest nodes

    #create graph
    Graph=nx.from_pandas_edgelist(edgelist,'Source','Target',['Source','Target','weight
','type',
        'Start date','End date'],create_using=nx.MultiDiGraph())
```

```
    nx.set_node_attributes(Graph,nodelist.set_index('Node
ID')['Name'].to_dict(),'Name')
    nx.set_node_attributes(Graph,nodelist.set_index('Node ID')['Set'].to_dict(),'Set')
    nx.set_node_attributes(Graph,nodelist.set_index('Node
ID')['Affiliation'].to_dict(),'Affiliation')

    #generate coordinates for nodes (spring layout,circular layout, shell layout)
    if len(n_list2)>1:
        pos=nx.drawing.layout.shell_layout(Graph,n_list)
    else:
        pos=nx.drawing.layout.spring_layout(Graph)

    for i in Graph.nodes():
        Graph.nodes[i]['pos']=list(pos[i])


#one node case
    if len(n_list2)==0:
        traceRecode=[]
        # go.Scatter(mode="markers", x=namevariants, y=namestems,
marker_symbol=symbols,
        #                    marker_line_color="midnightblue",
marker_color="lightskyblue",
        #                    marker_line_width=2, marker_size=15,
        #                    hovertemplate="name: %{y}%{x}<br>number: %{marker.symbol}<
extra></extra>"))

        node_trace=g_obj.Scatter(x=tuple([1]),y=tuple([1]),text=tuple([str(central_node
)]),textposition="bottom center",
            mode='markers+text',marker={'size':50,'color':'LightSkyBlue'})
        traceRecode.append(node_trace)

        node_trace1=g_obj.Scatter(x=tuple([1]),y=tuple([1]),mode='markers',
            marker={'size':50,'color':'LightSkyBlue'},opacity=0)
        traceRecode.append(node_trace1)

        figure={"data":traceRecode,"layout":g_obj.Layout(title='Process
Plot',showlegend=False,
            margin={'b':40,'l': 40, 'r': 40, 't': 40},
            xaxis={'showgrid': False, 'zeroline': False,'showticklabels': False},
            yaxis={'showgrid': False, 'zeroline': False,'showticklabels':
False},height=600)}

        return figure
```

```
    # normal mode
    traceRecode=[]
    colors=list(Color('lightcoral').range_to(Color('darkred'),len(Graph.edges())))
    colors=['rgb'+str(x.rgb) for x in colors]

    #for each edge get position, type and weight

    edge_weights=Graph.edges(data='weight')
    edge_types=Graph.edges(data='type')
    # print(list(A)[0][2])

    j=0
    k=0
    for i in Graph.edges():

        x0,y0=Graph.nodes[i[0]]['pos']
        x1,y1=Graph.nodes[i[1]]['pos']
        weight=list(edge_weights)[k][2]  #there is a change here for normalisation
        type=list(edge_types)[k][2]    #get single or double
        trace=g_obj.Scatter(x=tuple([x0, x1, None]), y=tuple([y0, y1,
None]),mode='lines',
                            line={'width': weight},marker=dict(color=colors[j]),
                            line_shape='spline',opacity=1)
        traceRecode.append(trace)
        j=j+1
        k=k+1

    #prepare hovertext and display for each node
    node_trace=g_obj.Scatter(x=[],y=[],hovertext=[],text=[],mode='markers+text',textpos
ition="bottom center",
                            hoverinfo="text",marker={'size':50,'color':'darkred'})


    j=0
    for i in Graph.nodes():
        x,y=Graph.nodes[i]['pos']
        name=Graph.nodes[i]['Name']
        type_of_node=Graph.nodes[i]['Set']
        affl=Graph.nodes[i]['Affiliation']
        hovertext="Name: "+str(name)+"<br>"+"Node type:
"+str(type_of_node)+"<br>"+"Affiliation: "+str(affl)
        text=nodelist['Node ID'][j]
        node_trace['x']+=tuple([x])
```

```
        node_trace['y']+=tuple([y])
        node_trace['hovertext']+=tuple([hovertext])
        node_trace['text']+=tuple([text])
        j=j+1

    traceRecode.append(node_trace)

    #prepare hovertext and display for each edge
    mid_edge_trace = g_obj.Scatter(x=[], y=[], hovertext=[], mode='markers',
hoverinfo="text",
                                   marker={'size': 20,'color':
'LightSkyBlue'},opacity=0)
    # print(mid_edge_trace['x'])

    j=0
    for i in Graph.edges:
        x0,y0=Graph.nodes[i[0]]['pos']
        x1,y1=Graph.nodes[i[1]]['pos']
        hovertext="From: "+str(Graph.edges[i]['Source'])+ "<br>"+ "to:
"+str(Graph.edges[i]['Target']
        )+"<br>"+"edge type: "+str(Graph.edges[i]['type'])    #also add extra info if
required
        mid_edge_trace['x'] +=tuple([(x0+x1)/2])
        mid_edge_trace['y']+=tuple([(y0+y1)/2])
        mid_edge_trace['hovertext']+=tuple([hovertext])
        j=j+1

    traceRecode.append(mid_edge_trace)

    #figure setup
    figure={"data":traceRecode,
            "layout":g_obj.Layout(title='Interactive
Plot',showlegend=False,hovermode='closest',
            margin={'b': 40, 'l': 40, 'r': 40, 't': 40},xaxis={'showgrid': False,
'zeroline': False,'showticklabels': False},
            yaxis={'showgrid':False,'zeroline':False,'showticklabels':False},height=600
,clickmode='event+select',
            #
annotations=[dict(ax=(G.nodes[edge[0]]['pos'][0]+G.nodes[edge[1]]['pos'][0])/2,
            #                   ay=(G.nodes[edge[0]]['pos'][1] +
G.nodes[edge[1]]['pos'][1]) / 2, axref='x', ayref='y',
            #                   x=(G.nodes[edge[1]]['pos'][0] * 3 +
G.nodes[edge[0]]['pos'][0]) / 4,
```

```
            #                   y=(G.nodes[edge[1]]['pos'][1] * 3 +
G.nodes[edge[0]]['pos'][1]) / 4, xref='x', yref='y',
            #                   showarrow=True,arrowhead=3,arrowsize=4,arrowwidth=1,opa
city=1) for edge in G.edges]
    )}

    return figure




####### app Layout ############

#styles for information tab
styles={'pre':{'border':'thin lightgrey solid','overflowX':'scroll'}}


app.layout=html.Div([
    #title
    html.Div([html.H1('Visualisation
Tool')],className="row",style={"textAlign":"center"}),

    #define the row
    html.Div(
        className="row",
        children=[
                html.Div(
                className="twelve columns",
                children=[dcc.Graph(id="my-
graph",figure=network_for_interactive(timeline, focus_node))]),


            #left side
            html.Div(
                className="twelve columns",
                children=[
                    dcc.Markdown(ddt(""" **Time Period displayed**

                    Define time range in timebar.
                    """)),
                    html.Div(
                        className="twelve columns", #check this if mistake
```

```
                        children=[ #for autogenerated marks ignore step, for no marks
type marks=None
                            dcc.RangeSlider(timeline[0],timeline[-
1],1,value=[timeline[0],timeline[1]],
                            id='my-range-
slider',dots=False,allowCross=False,included=True),
                            html.Br(),
                            html.Div(id='output-container-range-slider')],
                        style={'height':'300px'}),

                    html.Div(
                        className="two columns",
                        children=[
                            dcc.Markdown(ddt(""" **Focus on Node**

                            Select node to center graph.
                            """)),
                            dcc.Input(id="input1",type="text",placeholder="Task or
role"),
                            html.Div(id="output")],
                        style={'height':'300px'})
                        ]),
            #middle of the plot (graph)


            #add extra pieces like click on node for full data etc.
        ]
    )
])


### callback part ###############
@app.callback(
    dash.dependencies.Output('my-graph','figure'),[dash.dependencies.Input('my-range-
slider','value'),
    dash.dependencies.Input('input1','value')])
def update_output(value,input1):
    timeline=value
    focus_node=input1
    return(network_for_interactive(timeline,focus_node))


if __name__ == '__main__':
    app.run_server(debug=True)
```