



UNIVERSIDAD
DE GRANADA



Instituto de
Matemáticas
Universidad de Granada

PHD DISSERTATION

DOCTORAL PROGRAM IN MATHEMATICS (B07.56.1)

Mathematical Optimization for the Design of Complex Networks

Gabriel González Domínguez

Directores: Víctor Blanco Izquierdo
*Universidad de Granada. Departamento de Métodos
Cuantitativos para la Economía y la Empresa*

Justo Puerto Albandoz
*Universidad de Sevilla. Departamento de Estadística
e Investigación Operativa*

Editor: Universidad de Granada. Tesis Doctorales
Autor: Gabriel González Domínguez
ISBN: 978-84-1195-887-5
URI: <https://hdl.handle.net/10481/108109>

Resumen

Esta tesis aborda el estudio de métodos matemáticos de optimización orientados al diseño de redes complejas, con un enfoque particular en sus aplicaciones dentro de la ingeniería naval y los sistemas industriales. El objetivo central es desarrollar nuevos modelos de optimización que integren de manera explícita restricciones estructurales y operativas, con el fin de mejorar el diseño de sistemas como tuberías espaciales y bandejas de cables. Algunos de estos modelos también permiten el análisis y la manipulación de sistemas dinámicos.

Las metodologías propuestas se basan en Programación Lineal Entera Mixta (MILP), Programación Cónica Entera Mixta de Segundo Orden (MISOCP) y programación fraccionaria generalizada, las cuales se emplean para formular y resolver diversos problemas asociados al diseño de redes.

La estructura de la tesis es la siguiente:

- El capítulo 1 establece el marco teórico, determina los objetivos y contextualiza la motivación de esta investigación.

A partir de esta base, los capítulos siguientes abordan desafíos específicos dentro de contextos industriales e ingenieriles:

- El capítulo 2 se centra en el enrutamiento óptimo de tuberías espaciales en entornos de ingeniería. Se propone un modelo matemático orientado a minimizar la longitud total de la tubería, considerando tanto los obstáculos espaciales como las restricciones físicas de diseño.
- El capítulo 3 estudia la ubicación óptima de bandejas de cables en espacios navales, un componente clave en la arquitectura de embarcaciones.

Las formulaciones MILP desarrolladas para estos problemas se resuelven mediante técnicas de ramificación y corte, complementadas con heurísticas específicas. Los modelos se validan a través de experimentos computacionales y casos de estudio reales.

- El capítulo 4 analiza la configuración de árboles de longitud mínima bajo topologías fijas, con aplicaciones en el diseño de infraestructura industrial. Se introducen diversas formulaciones matemáticas, incluyendo un modelo MISOCP que incorpora restricciones de tipo cónico de segundo orden para representar de forma precisa las distancias euclidianas.

Posteriormente, la discusión se amplía hacia el diseño esquemático de sistemas de cableado industrial en espacios tridimensionales. En este contexto, se integran restricciones espaciales —como márgenes de seguridad y evitación de

obstáculos— dentro de un marco basado en MILP, el cual se optimiza mediante una descomposición jerárquica que refleja la lógica del cableado industrial y mejora la escalabilidad del enfoque.

- El capítulo 5 introduce el concepto de Factor de Amplificación Máxima (MAF), una métrica innovadora diseñada para identificar subsistemas autoamplificantes en hipergrafos. Se desarrollan modelos de programación no lineal, tanto fraccionarios como enteros mixtos, con aplicaciones al análisis de sistemas dinámicos.
- Finalmente, el capítulo 6 sintetiza los principales hallazgos y esboza posibles líneas de investigación futura.

Esta tesis doctoral contribuye al avance del campo de la optimización matemática aplicada al diseño de redes complejas, mediante la formulación de nuevos modelos, estrategias resolutivas y aplicaciones prácticas. Aborda problemas previamente explorados y propone algoritmos escalables, validados tanto en escenarios industriales como en escenarios aleatorios. El trabajo establece una base sólida para el desarrollo de futuras herramientas de optimización con implicaciones significativas en la planificación de infraestructuras y el diseño de ingeniería moderno.

Abstract

This dissertation explores mathematical optimization methods tailored for the design of complex networks, with a focus on applications in naval engineering and industrial systems. The primary objective is to develop novel optimization models that integrate structural and operational constraints to improve the design of systems such as spatial pipelines and cable trays. Some of these optimization models also allow us to analyze and manipulate dynamic systems. The proposed methodologies leverage Mixed-Integer Linear Programming (MILP), Mixed-Integer Second-Order Cone Programming (MISOCP), and generalized fractional programming to formulate and solve a diverse range of network design problems.

The structure of the thesis is as follows:

Chapter 1 establishes the theoretical framework, identifies the objectives and contextualizes the motivation behind this research.

Building on this foundation, subsequent chapters tackle domain-specific challenges in industrial and engineering contexts.

Chapter 2 focuses on the optimal routing of spatial pipelines in engineering environments. A mathematical model is formulated to minimize total pipeline length while accounting for spatial obstacles and physical design constraints.

Chapter 3 addresses the optimal placement of cable trays in naval environments, a critical aspect of ship architecture.

For both pipeline routing and cable tray layout, MILP formulations are solved using branch-and-cut techniques, complemented by custom heuristics. The models are validated through computational experiments and tested on real-world engineering instances.

Chapter 4 starts exploring the configuration of minimum-length trees under fixed topologies, with applications in industrial infrastructure layout. Several mathematical formulations are presented, including a MISOCP model that incorporates second-order cone constraints to accurately handle Euclidean distances.

Following this presentation and analysis, the discussion extends to the schematic layout of industrial wiring systems in three-dimensional space. Spatial constraints—such as safety margins and obstacle avoidance—are incorporated within a MILP-based framework, enhanced by a hierarchical decomposition that reflects industrial wiring logic and improves scalability.

Chapter 5 introduces the Maximal Amplification Factor (MAF), a novel metric designed to identify self-reinforcing subsystems in hypergraphs. This chapter develops both fractional and mixed-integer nonlinear programming models applicable to the analysis of dynamical systems.

Chapter 6 synthesizes the findings and outlines avenues for future research.

This dissertation advances the field of mathematical optimization for complex network design by introducing new models, solution strategies, and practical applications. It formulates previously unaddressed problems and delivers scalable algorithms validated on both industrial and dynamical systems. The work lays a foundation for future developments in intelligent, domain-aware optimization tools, with broad implications across infrastructure planning and modern engineering design.

Contents

Resumen	iii
Abstract	v
1 Introduction	3
1.1 Objectives	9
2 Network Flows in Pipelines Routing	11
2.1 Introduction	11
2.2 The Pipeline Routing Problem in Naval Design	13
2.3 A multicommodity network flow based model	17
2.3.1 An exact solution method for (PRPS)	21
2.4 Two matheuristic algorithms for the PRPS	22
2.4.1 Dimensionality Reduction	22
2.4.2 Decomposition-based algorithms	23
2.5 Computational experience	24
2.5.1 Random instances	24
2.5.2 Case study	27
2.6 Conclusions	31
A. Pseudocode of the algorithms described in Section 2.4	33
A1. Pseudocode for the dimensionality reduction algorithm in Section 2.4.1	33
A2. Pseudocode for the decomposition-based algorithm described in Section 2.4.2	34
3 Pipeline and Cable Tray Placement	37
3.1 Introduction	37
3.1.1 Motivation	37
3.1.2 Literature Review	38
3.1.3 Contributions	40
3.1.4 Organization	40
3.2 Main elements of the PCTLTP	40
3.2.1 Services	40
3.2.2 Solution space	41
3.2.3 Cost structure	41
3.2.4 The Pipelines and Cable Trays Location Problem	42
3.3 A mathematical programming model for the PCTLTP	42

3.3.1	An exact relax-and-cut procedure for solving PCTLP	45
3.4	A general heuristic approach	46
3.4.1	Strategies for Phase I	46
3.4.2	Strategies for Phase II	49
3.5	Computational experience	51
3.5.1	Random instances	52
3.5.2	Corridor instances	55
3.5.3	Case study	57
3.6	Conclusions	60
4	Topology-Constrained Tree Location	63
4.1	Introduction	63
4.2	Fixed Topology Minimum-Length Trees with Neighborhoods	67
4.3	The continuous-domain FTTNP	69
4.3.1	Graph Transformation	69
4.3.2	The Steiner tree problem on $\tilde{T}$	70
4.3.3	Mathematical Optimization Model	71
4.4	The discrete-domain FTTNP	74
4.4.1	Mathematical Optimization Model	75
4.4.2	A variable fixing strategy for geometric graphs	77
4.5	Wiring Diagram Problem	78
4.5.1	A graph-based discretization of the space	80
4.5.2	Mathematical Optimization Model	81
4.6	Experiments	84
4.6.1	Instances	84
4.7	Conclusions	89
5	Self-Amplifying Hypergraph Structures	91
5.1	Introduction	91
5.2	Preliminaries	93
5.3	Computing the Maximal Amplification Factor	99
5.4	Optimal Subhypergraphs under maximal amplifying factors	101
5.5	Incorporating features to the mathematical model	108
5.6	Computational Experiments	111
5.7	Conclusions	113
6	Conclusions	115
6.1	Summary of Contributions	115
6.2	Future Research	117
6.3	Final Remarks	118
	Bibliography	121

Chapter 1

Introduction

This chapter presents a comprehensive overview of the mathematical and algorithmic foundations that underpin the optimization of complex networks. These networks, often arising in domains such as logistics, telecommunications, and dynamic systems, demand efficient and scalable models to address inherent complexity. We build the theoretical groundwork required to model, analyze, and optimize these systems. We begin with core principles of operations research and mathematical programming, followed by a discussion of linear and fractional programming models. We then present exact solution algorithms and turn our focus to network models, including classical graph-based representations and hypergraphs. We introduce core flow-based optimization problems, including the multicommodity flow problem, which serves as the basis for models developed throughout this thesis. Finally, we determine the objectives according to which this thesis dissertation has been developed.

Operations Research (OR) plays a fundamental role in contemporary decision-making processes across a wide range of fields. It provides a systematic and analytical methodology for addressing complex problems, optimizing processes, and supporting informed decisions. Its importance lies in its application to efficient resource allocation, cost minimization, risk assessment, and performance improvement, ultimately leading to greater efficiency and productivity. OR techniques are widely applied in sectors such as supply chain management, transportation, healthcare, and finance, among others. These applications have resulted in cost reductions, enhanced service quality, and optimized resource utilization. Moreover, OR equips organizations with tools to tackle real-world challenges through data-driven decision-making, enhancing their competitiveness and fostering sustainable development ([Aug, 2020]).

An optimization problem involves identifying the best feasible solution with respect to an objective function. The standard formulation of an optimization problem is as follows:

$$\begin{aligned} \text{minimize} \quad & f(x) & (\text{OP.1}) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad \forall i = 1, \dots, m, & (\text{OP.2}) \\ & h_j(x) = 0, \quad \forall j = 1, \dots, p, & (\text{OP.3}) \\ & x \in X \subseteq \mathbb{R}^n. & (\text{OP.4}) \end{aligned} \tag{1.1b}$$

Here, $f : \mathbb{R}^n \rightarrow \mathbb{R}$ represents the objective function to be minimized, and x is a vector of decision variables. The feasible set is defined by the inequality constraints g_i , the equality constraints h_j , and the domain X , which may be continuous, discrete, or a combination of both. This formulation represents one of the most general expressions of an optimization problem, as many real-world problems can be captured within this framework. Despite its generality, several structural properties and solution methods have been well established, such as the Karush-Kuhn-Tucker (KKT) conditions ([Karush, 1939]; [Kuhn and Tucker, 1951]).

Among the broad class of optimization problems, linear programming occupies a central role due to its tractability and breadth of application ([Dantzig et al., 1954]). Other notable optimization problems are Quadratic Programming ([Markowitz, 1952]), Second-Order Conic Programming, Semidefinite Programming, General Conic Programming ([Nesterov and and, 1992]; [Nesterov and Nemirovskii, 1994]; [Alizadeh, 1995]), and Fractional Programming. Most of the models developed in this work fall within the field of Linear Programming, particularly Mixed Integer Linear Programming (MILP).

A Linear Programming (LP) problem can be expressed in standard form as:

$$\text{minimize} \quad c^T x \tag{LP.1}$$

$$\text{subject to} \quad Ax = b, \tag{LP.2}$$

$$x \geq 0, \tag{LP.3}$$

where $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are given vectors, and $A \in \mathbb{R}^{m \times n}$ is a full-rank coefficient matrix. The objective is to find the vector x that minimizes the linear function $c^T x$, subject to the constraints defined by the polyhedron $Ax = b$ and $x \geq 0$.

A key property of LP problems is that, due to the linearity of the objective function and the feasible set, the optimal solution lies at a vertex of the feasible polyhedron. The Simplex algorithm, proposed by [Dantzig et al., 1954], systematically explores these vertices, iteratively improving the objective value until reaching the minimum. Each vertex corresponds to a subset of variables called basic variables, which form a basis for the system $Ax = b$. Thus, the optimal solution can be found by solving a reduced system involving only these variables.

Note that not all LP problems have a solution. A problem is said to be infeasible if the feasible set $x \in \mathbb{R}^n : Ax = b, x \geq 0$ is empty, and unbounded if the objective function can decrease indefinitely within the feasible set. Linear programming has proven to be extremely useful in modeling and solving problems related to logistics, planning, and routing. Despite the practical effectiveness of the Simplex algorithm, a polynomial-time algorithm for LP was not developed until the 1980s, due to the work of [Khachiyan, 1980].

To model complex nonlinear relationships in optimization problems, Generalized Fractional Programming (GFP) is a relevant approach. This extension of classical fractional programming allows for objective functions expressed as ratios of functions that may lack convexity or concavity, including multiple, weighted, or higher-order quotients. Algorithms for minimizing such functions have been proposed by [Barros et al., 1996, Blanco et al., 2013, Crouzeix and Ferland, 1991]. This type of

formulation is especially valuable in scenarios requiring a balance between multiple interdependent criteria, such as continuous location problems, game theory, or matrix spectral analysis [Nishioka et al., 2025].

GFP models pose significant computational challenges due to their inherently nonconvex nature, which prevents the direct application of traditional techniques. Specialized algorithms have been developed for solving them, including iterative procedures inspired by the Dinkelbach method, which ensure convergence under specific conditions [Crouzeix and Ferland, 1991]. In this thesis, this approach is used to model and solve optimization problems on complex combinatorial structures, such as directed hypergraphs, where component relationships are naturally expressed through nonlinear generalized quotients.

Branch-and-cut algorithms are widely used exact methods for solving integer programming problems. They combine two complementary approaches: branch-and-bound, which organizes the search using a divide-and-conquer strategy, and cutting-plane generation, which iteratively refines the linear relaxation of the problem.

These algorithms enable the solution of large instances with optimality guarantees, thanks to cuts derived from the problem’s polyhedral structure, such as Chvátal-Gomory cuts, knapsack cuts, and disjunctive inequalities. While cuts strengthen the relaxation, they are rarely sufficient on their own, so branching remains essential.

In practice, strategies such as generating cuts primarily at the root node (branch-and-cut approach), limiting the number of cuts per node, or storing globally valid cuts are commonly applied. Additionally, heuristics are employed to find high-quality integer solutions, as well as inequality lifting and preprocessing techniques to simplify the initial model.

The use of parallel computing further enhances performance by distributing the computational workload across multiple cores. A comprehensive analysis of these methods is provided in [C. A. Floudas, 2009].

These algorithmic strategies are especially powerful when applied to structured problems like network optimization, where nodes and connections can be leveraged to reduce complexity.

A network is a complex structure comprising two key elements: a set of nodes—often representing a population—and a set of connections or edges that define relationships or interaction patterns between the nodes. While the classical definition of nodes is universally accepted, the nature of the connections may vary considerably depending on the modeled structure and the intended objectives. These structures are used to represent large volumes of data in various domains, including biology, computer science, facility location, transportation, sociology, and economics (see Newman [2018]).

Once data is represented as a network, it becomes possible to analyze it and extract valuable insights. In this context, Operations Research (OR) and Mathematical Programming (MP) serve as essential decision-support tools for a wide range of arising problems. These techniques are commonly applied in discrete optimization problems involving networks, such as facility location (Nickel and Puerto [2009]), data centers (da Gama and Wang [2024]), logistics (Fischetti et al. [2007]), supply

chain management ([Johnson and Stevens \[2016\]](#)), and game theory ([Owen \[1968\]](#)).

Today, many real-world challenges benefit from these techniques, especially in the context of cost reduction, service quality improvement, and resource or population structure optimization, all based on large-scale data analysis. Consequently, network analysis and optimization play a critical role in data-driven decision-making by providing the mathematical and methodological foundation necessary for solving complex problems.

A key limitation of classical network models is their pairwise structure: each edge connects only two nodes. However, many complex systems exhibit interactions beyond simple pairwise connections. There is a need for more intricate tools to capture the complex, multidirectional interactions observed in many real-world systems.

Hypergraphs generalize classical networks by allowing edges (hyperedges) to connect multiple nodes simultaneously. This capability is crucial for modeling higher-order interactions in fields such as biology, process engineering, and multi-agent systems [[Andersen et al., Nagy et al., 2022](#), [Wang et al., 2024](#)]. They are also used to represent complex structures in planning models [[Michelena and Papalambros, 1997](#)].

However, this added expressiveness introduces new challenges: many problems that are tractable in graphs become NP-hard in hypergraphs, including minimum edge covering [[Repisky et al., 2007](#)], maximum matching [[Berman, 2003](#)], and shortest path computation [[Ausiello and Laura, 2017](#)]. These challenges are even more significant in directed or multiset variants, where many open questions remain. Thus, although hypergraphs provide new modeling capabilities, they also demand the development of specialized combinatorial optimization tools.

A visual comparison between classical networks and their generalization, hypergraphs, is presented in Figure 1.1.

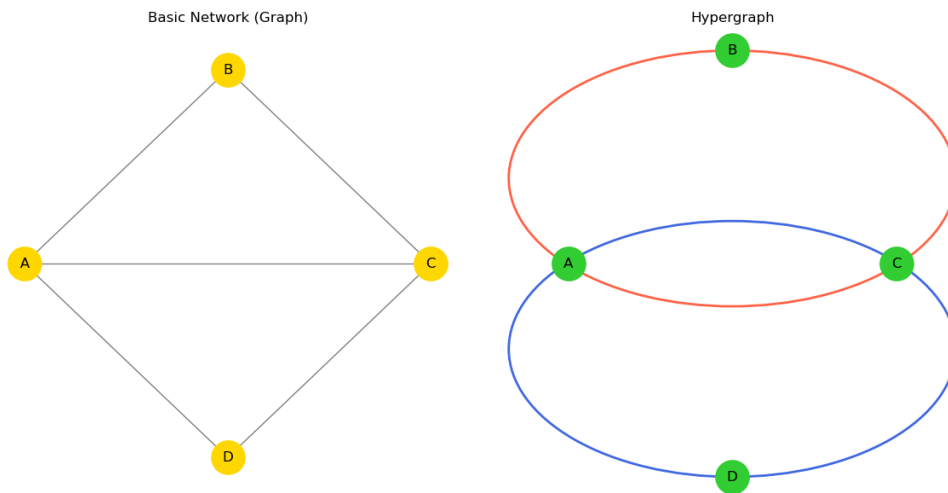


Figure 1.1: Comparison between a basic network (left) and a hypergraph (right). While the classical network contains pairwise edges, the hypergraph allows edges (hyperedges) to connect more than two nodes simultaneously.

The simplex method is one of the most efficient algorithms for solving constrained

linear optimization problems. Since network flow problems have a particular structure, it is essential to determine whether the simplex method can compete with other combinatorial methods that exploit this structure, such as shortest path algorithms. When applied without leveraging this structure, the general simplex method is not competitive in solving minimum-cost flow problems. However, if the basic concepts of the method are reinterpreted as operations on networks, it can be adapted to take advantage of this structure and result in an efficient algorithm. Thus, the simplex algorithm for networks is a specialized adaptation of the general simplex method.

The fundamental concept of the simplex algorithm for networks involves solutions in the form of spanning trees, which are obtained by setting the flow on edges not belonging to the tree to zero. Every minimum-cost flow problem has at least one optimal solution corresponding to a spanning tree. This optimal solution can be found by iteratively replacing one edge of the tree with another that does not belong to it. This procedure is called the simplex algorithm for networks because spanning trees correspond to the basic feasible solutions in linear programming, and the transition between trees is equivalent to pivot operations in the general simplex method. For a detailed analysis of flow problems in networks, see [Ahuja et al., 1993].

A minimum spanning tree (MST) is a subset of the edges of an undirected, connected, and weighted graph that connects all vertices without forming cycles and with the lowest possible total weight. Given a subset of vertices, called terminals, and a graph with non-negative edge weights, the Steiner tree problem consists of finding a minimum-weight tree that includes all terminals—possibly adding non-terminal intermediate vertices—and minimizes the total weight of the resulting tree.

This problem generalizes both the shortest path problem (when the number of terminals is two) and the minimum spanning tree problem (when all vertices are terminals). Unlike the latter, which can be solved in polynomial time, the Steiner tree problem is NP-hard, which has motivated the development of numerous exact and heuristic approaches.

In recent decades, sophisticated algorithmic frameworks have been proposed to address variants of the problem. For example, [Ljubić et al., 2006] develops an exact framework for solving the Prize-Collecting Steiner Tree Problem, a generalization that balances the connection of terminals with penalties associated with their exclusion. More recently, [Ljubić, 2021] provides a detailed review of progress in solving the classical Steiner problem, highlighting current challenges and new algorithmic perspectives. In the Euclidean context and in spaces of arbitrary dimension, [Pinto and Maculan, 2023] proposes a new heuristic for the Steiner problem in $\mathbb{R}^n$, demonstrating its effectiveness in large-scale instances.

The minimum-cost flow model is the most fundamental problem in network flow analysis. Its objective is to determine the lowest-cost route to transport a service through the network, satisfying demand at certain nodes from the supplies available at others.

Let $G = (N, A)$ be a directed network, where N is the set of nodes and A the set of m directed arcs. Each arc $(i, j) \in A$ has a cost per unit of flow c_{ij} , an upper capacity u_{ij} , and a lower bound l_{ij} . Each node $i \in N$ has a supply or demand $b(i)$. If $b(i) > 0$, the node is a supply node; if $b(i) < 0$, it is a demand node with a demand

of $-b(i)$; if $b(i) = 0$, it is a transshipment node. The decision variable is the flow x_{ij} on arc $(i, j) \in A$. The problem is formulated as:

$$\text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \quad (\text{flow 1})$$

$$\text{subject to} \quad \sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (j,i) \in A} x_{ji} = b(i) \quad \forall i \in N, \quad (\text{flow 2})$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A, \quad (\text{flow 3})$$

where $\sum_{i=1}^n b(i) = 0$. In matrix form, it is expressed as:

$$\text{Minimize} \quad cx \quad (\text{M.1})$$

$$\text{subject to} \quad \mathcal{N}x = b, \quad (\text{M.2})$$

$$l \leq x \leq u, \quad (\text{M.3})$$

where $\mathcal{N}$ is the incidence matrix of size $n \times m$. Each column $\mathcal{N}_{ij}$, associated with x_{ij} , has a $+1$ in row i , a -1 in row j , and zeros elsewhere.

The constraints (flow 2) impose flow balance: the first term represents the total output of a node, and the second the input. At supply nodes, output exceeds input; at demand nodes, the opposite is true; at transshipment nodes, both are equal. Constraints (flow 3) establish flow limits, reflecting physical or operational restrictions. Typically, $l_{ij} = 0$ is assumed unless otherwise specified.

The data are usually assumed to be integers, since rational numbers can be scaled to integers, and irrational numbers must be rationally approximated for feasibility.

This problem seeks the minimum-cost (or shortest) path from a source node s to a destination node t , considering that each edge $(i, j) \in A$ has a cost c_{ij} .

The minimum-cost flow model considers a single service. When multiple services share a network, a multi-service flow problem arises. Each service has its own origin-destination pair and its own balancing constraints, but they share the capacity of the edges. The challenge is to efficiently allocate these capacities among the services to minimize the total cost.

Since the simplex algorithm for networks generates a sequence of solutions in the form of spanning trees, its efficient implementation requires suitable data structures to represent and update these trees computationally. The tree starts from a root node (usually node 1). Each node i is associated with three indices: Predecessor index ($pred(i)$) stores the first node on the path from i to the root, with $pred(1) = 0$ by convention. Depth index ($depth(i)$) indicates the number of edges between i and the root. And branch index describes a sequence that traverses the nodes from the root down and back to the root.

The multicommodity flow problem (MCFP) is a generalization of the classic network flow problem, in which multiple types of products, services, or commodities share the same transportation or communication infrastructure. Each of these commodities has a source, a destination, and a quantity to be transported, and all compete for the limited capacity of the network's arcs.

Formally, let x_{ij}^k denote the flow of service k on arc (i, j) of a directed network $G = (N, A)$, where N is the set of nodes and A the set of arcs. Define x^k as the vector of flows of service k , and c^k as the vector of unit costs associated with that flow. The basic MCFP model can be formulated as the following linear optimization problem:

$$\text{Minimize} \quad \sum_{1 \leq k \leq K} c^k x^k \quad (\text{MCFP 1})$$

$$\text{subject to} \quad \sum_{1 \leq k \leq K} x_{ij}^k \leq u_{ij}, \quad \forall (i, j) \in A, \quad (\text{MCFP 2})$$

$$\mathcal{N} x^k = b^k, \quad k = 1, 2, \dots, K, \quad (\text{MCFP 3})$$

$$0 \leq x_{ij}^k \leq u_{ij}, \quad \forall (i, j) \in A, \forall k. \quad (\text{MCFP 4})$$

The objective function (MCFP 1) seeks to minimize the total transportation cost of all commodities. Constraints (MCFP 2) ensure that the capacity of each arc is not exceeded by the combined flow. Equations (MCFP 3) represent the flow conservation conditions at the nodes, modeled by the incidence matrix $\mathcal{N}$ and the demand vectors b^k . Finally, (MCFP 4) imposes lower and upper bounds on the individual flows.

This problem has been widely studied due to its applicability in contexts such as the design of logistics, telecommunications, and energy transportation networks. In their seminal work, [Ahuja et al., 1993] presents a comprehensive description of classical algorithms for solving network flow problems, including multicommodity extensions. The computational complexity of MCFP increases significantly compared to the single-commodity case due to the interaction between multiple flows competing for shared resources.

More recent advances have addressed sophisticated variants of the problem. [Garg and Smith, 2008] introduces models that consider general failure scenarios to ensure network resilience in the face of contingencies. [Arslan et al., 2020] applies MCFP to the design of collaborative networks in strategic alliances, highlighting the role of cost and inter-organizational coordination.

Together, these developments show how MCFP has established itself as a versatile framework for modeling and optimizing complex networks in dynamic and strategically significant environments.

The models and methods presented in this chapter form the methodological foundation of the thesis. The following chapters build upon these concepts to propose new optimization frameworks for the design and analysis of complex networks, with particular emphasis on scalability, structural complexity, and practical relevance.

1.1 Objectives

The objectives according to which this thesis dissertation has been developed can be summarized as follows:

- To propose novel mathematical optimization models for the design of complex networks, incorporating key structural and operational aspects that arise in real-world applications and extending existing approaches in the literature.
- To analyze the theoretical properties of the proposed optimization models, identifying structural characteristics that can be leveraged to improve their computational tractability and solution methodologies.
- To develop exact and approximate solution methods tailored to the complexity of the proposed models, including decomposition techniques, valid inequalities, and heuristic or matheuristic approaches.
- To conduct extensive computational experiments to evaluate the performance of the proposed models and solution methods, analyzing their efficiency, scalability, and applicability to different network design problems.
- To apply the developed methodologies to practical case studies, demonstrating their usefulness in solving real-world network design problems and highlighting their potential impact in various domains such as building infrastructures and dynamyc systems analisys.

Chapter 2

Network flow based approaches for the pipelines routing problem in naval design

In this chapter, we propose a general methodology for the optimal automatic routing of spatial pipelines, motivated by a recent collaboration with Ghenova, a leading naval engineering company. We develop a minimum-cost multicommodity network flow model that incorporates all the technical requirements necessary for feasible pipeline routing. A branch-and-cut approach is designed, and several matheuristic algorithms are developed to solve the problem efficiently. We present the results of extensive computational experiments to assess the performance of our approach, including a real-world case study provided by our industrial partner.

2.1 Introduction

The design and construction of ships is a complex process that begins with the concept design phase, during which all necessary requirements are defined to meet the client's needs. In the subsequent basic design phase, the ship's structure, piping systems, and electrical circuits are specified, and appropriate equipment is selected. One of the most challenging tasks in this phase is pipeline routing, where designers must determine the paths of various services while satisfying numerous technical constraints.

Pipeline routing requires coordination across multiple ship systems, including electrical circuits, water pipes, and fiber optics. It must also account for restrictions such as avoiding forbidden obstacles, ensuring sufficient space for machinery operation, and satisfying both compatibility and manufacturability requirements. Despite the existence of software tools to assist in this task, the problem's complexity still demands expert input to guide the design to a feasible and optimal solution.

Typically, design specifications include constraints such as the number of main lines, required branches, valves, and other components (see, e.g., [Cuervas et al. \[2017\]](#), [Yeates et al. \[2021\]](#)). Once these initial requirements are established, the pipeline routing problem must be solved in accordance with them. The main limitations of the pipeline routing problem in ship design can be categorized into three

types [Ando and Kimura \[2011\]](#), [Park \[2002\]](#), [Qian et al. \[2008\]](#):

Physical Constraints: Pipelines must avoid obstacles and connect the appropriate equipment.

Operational Constraints: Routes must allow for accessibility, ease of maintenance, and safe handling of equipment. Some areas (e.g., lower cabin spaces) may be preferred for routing due to their limited alternative uses.

Economic Constraints: Budget limitations affect both material and labor costs, incentivizing the minimization of pipe length and the number of elbows.

Some of these constraints are hard requirements for feasibility, while others can be modeled within a cost function to evaluate and compare feasible alternatives. Defining an appropriate cost function is crucial, as it must balance various considerations such as material and labor costs, elbow usage, routing through preferred areas, and traversable obstacles.

While several software solutions exist (e.g., AVEVA, FORAN, SMART3D), many fall short in handling all the technical constraints relevant to real-world ship design. Often, these tools are more suitable for industrial plant layouts and are either too rigid or too generalized to meet the specific needs of naval designers.

The objective of this chapter is to develop an efficient, mathematical programming-based methodology for the automatic routing of pipelines. This methodology should account for the specific technical requirements of naval design, while maintaining flexibility to adapt to various real-world scenarios and constructibility criteria.

Automatic pipeline routing has been widely addressed in the literature (see, e.g., [Guiradello and Swaney \[2005\]](#), [Kimura \[2011\]](#), [Min et al. \[2020\]](#), [Mostafaei et al. \[2021\]](#), [Park and Storch \[2002\]](#), [Singh and Cheng \[2021\]](#)). A critical aspect of this problem is the representation of the set of feasible routes. Although pipelines can theoretically be routed through a continuous three-dimensional space, solving the problem in this form is computationally intractable. Thus, discretization is necessary for both tractability and simplicity.

One common approach is spatial discretization via cell decomposition. For example, [Lee \[1961\]](#) proposed subdividing the space into square cells, discarding those containing obstacles. This method has been widely used in shortest-path algorithms [Hightower \[1969a\]](#), [Li et al. \[2017\]](#), [Mitsuta et al. \[1986\]](#), [Rourke \[1975\]](#), [Schmidt-Traub et al. \[1999\]](#) and adapted to pipeline routing problems [Ando and Kimura \[2011\]](#), [Asmara \[2013\]](#), [Asmara and Nienhuis \[2006a\]](#), [Kim et al. \[2013\]](#), [Zhu and Latombe \[1991\]](#). However, achieving accurate results typically requires a large number of cells. Alternatively, space can be discretized using graph structures [Guiradello and Swaney \[2005\]](#), [Newell \[1973\]](#), [Shing and Hu \[1986\]](#), which can also be combined with cell-based approaches to improve efficiency [Park and Storch \[2002\]](#).

Once the space is discretized, several heuristic algorithms have been proposed to solve the pipeline routing problem, including genetic algorithms [Ikehira et al. \[2005\]](#), [Ito \[1999\]](#), [Kimura \[2011\]](#) and ant colony optimization techniques [Xiaoning et al. \[2006, 2007\]](#).

In our work, we adopt a graph-based discretization that allows the use of combinatorial optimization tools to solve the problem effectively. The graph is constructed based on the shape of the region, location of source and destination points, pipe sizes, preferred and penetrable zones, and other relevant characteristics. Nodes include key points such as pipe endpoints, corners, holes, and intermediate grid points. Edges are created between sufficiently close nodes and assigned costs based on pipeline-specific criteria.

Based on this framework, the key contributions of our work are as follows:

1. **Graph-Based Discretization:** We discretize the continuous three-dimensional space using a graph structure that facilitates pipeline routing while accounting for obstacles and elbow usage.
2. **Optimization Model:** We propose a multicommodity network flow model that incorporates physical and operational constraints, including minimum elbow spacing and service separation. A tailored branch-and-cut algorithm is developed to solve this model efficiently.
3. **Flexible Evaluation Function:** We design a cost function capable of evaluating feasible routes based on various real-world naval design features, such as route length, proximity to ceilings or floors, elbow usage, and traversing of penetrable zones.
4. **Efficient Matheuristics:** We develop and implement several matheuristic algorithms that can solve large-scale, realistic instances within reasonable computational times.

The chapter is organized as follows. In Section 2.2, we describe the pipeline routing problem and its mathematical formulation. Section 2.3 presents our proposed optimization model tailored to naval design constraints. Subsection 2.3.1 introduces a branch-and-cut method that incrementally incorporates complex constraints. In Section 2.4, we detail two families of matheuristic algorithms designed for scalability. Section 2.5 reports the results of computational experiments and includes a case study based on real industrial data. Finally, conclusions and future research directions are discussed in Section 2.6. An appendix is provided with pseudocode describing our algorithmic procedures.

2.2 The Pipeline Routing Problem in Naval Design

The goal of this chapter is to model and solve the problem of routing multiple pipelines within the constrained space of a ship, while satisfying relevant technical requirements. In this section, we describe the main elements of the problem: the input data, the set of feasible actions, and how a particular solution is evaluated.

We assume that the region in which the pipelines must be routed is represented by a bounded polyhedron $\mathcal{P}_0 \subset \mathbb{R}^3$. In practical scenarios, this polyhedron typically takes the form of a cuboid, expressed as $\mathcal{P}_0 = [a_1, a_2] \times [b_1, b_2] \times [c_1, c_2]$, representing a ship cabin. Additionally, we are given a finite set of regions $O_1, \dots, O_o \subseteq \mathcal{P}_0$, representing o obstacles that cannot be crossed by the pipeline routes. For simplicity,

these obstacle regions are also modeled as cuboids. The available routing space is then defined as the obstacle-free region: $\mathcal{P} = \mathcal{P}_0 \setminus \bigcup_{\ell=1}^o O_\ell$.

We are also provided with a finite set of services (i.e., cylindrical pipelines), each defined by a source and destination point within $\mathcal{P}$, a radius, and a minimum required safety distance from other services. While we focus on cylindrical pipelines in this work, our approach can be readily adapted to other pipeline geometries such as parallelepipeds with rectangular cross-sections.

The main decision in this problem is to determine the three-dimensional paths within $\mathcal{P}$ that each pipeline should follow. Initially, a feasible path for a service is modeled as a continuous sequence of straight segments connecting the source and destination. For constructibility reasons, a minimum required distance must be maintained between consecutive elbows (i.e., breakpoints along the path). Moreover, after the dimensional pipeline is traced along the path, the layout must ensure that a minimum safety distance is maintained between all pairs of pipelines. In particular, pipelines must not cross or overlap.

The quality of a feasible solution—defined by the set of paths for all services—is assessed based on multiple criteria. In addition to total pipeline length (which is typically proportional to installation cost), the evaluation also considers the number of elbows, proximity to the ceiling (which may be desirable for practical reasons), and whether the paths traverse predefined preferred zones.

In the remainder of this section, we introduce a mathematical representation of the key components involved in the Pipeline Routing Problem in a Ship (PRPS), which forms the core of our analysis.

To discretize the continuous space $\mathcal{P}$, we first construct an orthogonal 3D grid over the full region $\mathcal{P}_0$, ensuring that all source and destination points are included as grid nodes and are connected to other nodes in the grid. This grid defines a base undirected graph. Then, we remove any nodes and edges that intersect with the obstacle regions O_ℓ , for $\ell = 1, \dots, o$. A simple instance of the problem is illustrated in Example 1.

Example 1. *To illustrate the procedure described in this chapter, consider a scenario with one obstacle and three services (see Figure 2.1, left). The environment is a box with dimensions $6 \times 6 \times 4$, containing a central obstacle (a cuboid with dimensions $2 \times 2 \times 4$), representing a pillar. The discretized solution space is generated using an orthogonal 3D grid, from which all cells within the obstacle have been removed (see Figure 2.1, right).*

However, when tracing a path in the original graph, it is not possible to detect or penalize the presence of elbows. Elbows in a pipeline must be explicitly identified for both constructibility considerations and optimization purposes, as solutions with fewer elbows are generally preferred. The objective is to develop a mathematical programming model that avoids non-linearities. To incorporate the number of elbows into a linear objective function, we modify the initial graph by *exploding* its nodes as follows:

1. Each physical node v in the original graph is replaced by an *exploded node*, consisting of three *virtual nodes*— v_X , v_Y , and v_Z —which share the same 3D

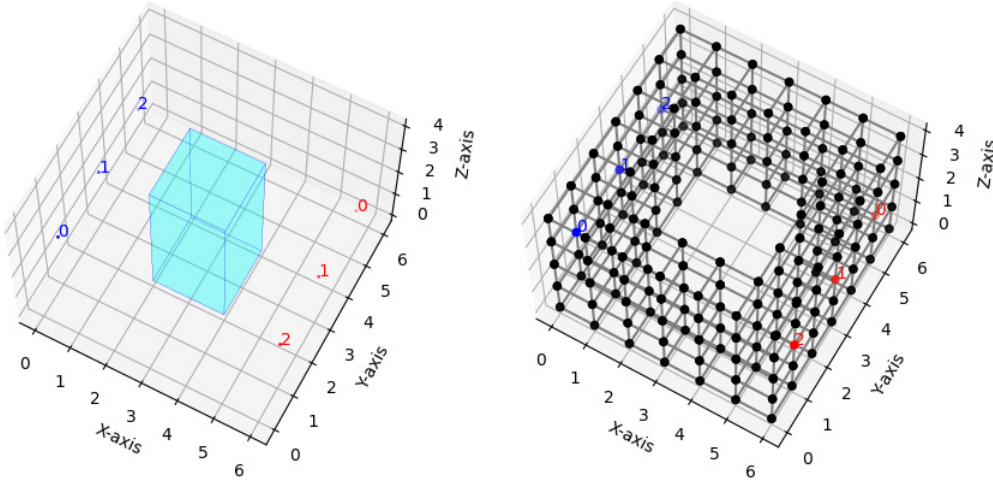


Figure 2.1: Illustration of the first discretization phase of a PRPS instance with an obstacle and three services with sources and destinations at the blue and red points, respectively. The left figure shows the origin and destination points and the obstacle. The right figure depicts the grid of the discretized space.

coordinates as v . Each virtual node corresponds to one of the three canonical directions in $\mathbb{R}^3$ (X, Y, and Z).

2. Virtual nodes associated with the same physical node are connected through *virtual edges*. These edges have zero length (as they connect nodes at the same coordinates), but are assigned a positive cost to account for the introduction of an elbow at that point in the route.
3. Each physical edge in the original graph is replaced by a corresponding edge of equal length that connects appropriate virtual nodes, instead of the original physical nodes. Specifically, an edge parallel to the X-axis connects the X-virtual nodes of its endpoints, an edge parallel to the Y-axis connects the Y-virtual nodes, and an edge parallel to the Z-axis connects the Z-virtual nodes.

Figure 2.2 illustrates this node explosion process. On the left, we show a crossing node v that is connected to six neighboring nodes in the original graph. On the right, the three virtual nodes v_X , v_Y , and v_Z —which represent the exploded form of node v —are shown. Each virtual node v_X (resp. v_Y , v_Z) is connected only to the two neighboring nodes that are linked by edges parallel to the X-axis (resp. Y-axis, Z-axis), as well as to the other two virtual nodes associated with the same physical location. The three virtual edges connecting v_X with v_Y , v_X with v_Z , and v_Y with v_Z are depicted using dashed lines in the figure.

Figure 2.3 illustrates how the modified graph is used to model different types of turns along a pipeline route. In the left image, the path from node a to node b involves no elbows, using only a single virtual node (v_Y). In contrast, the center

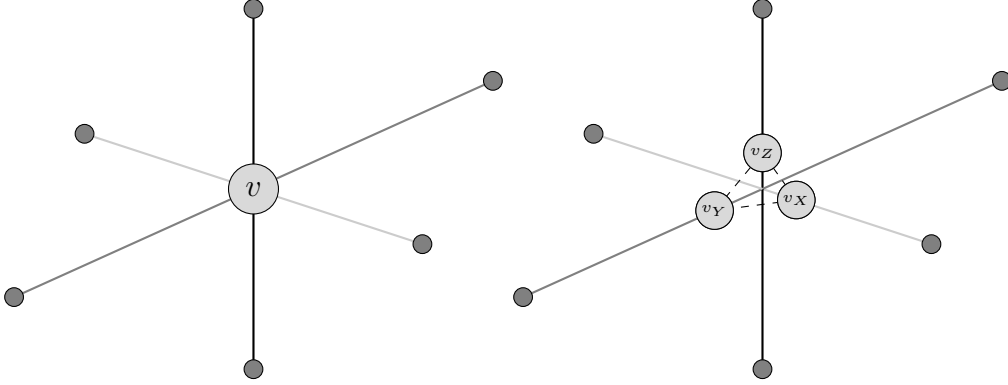
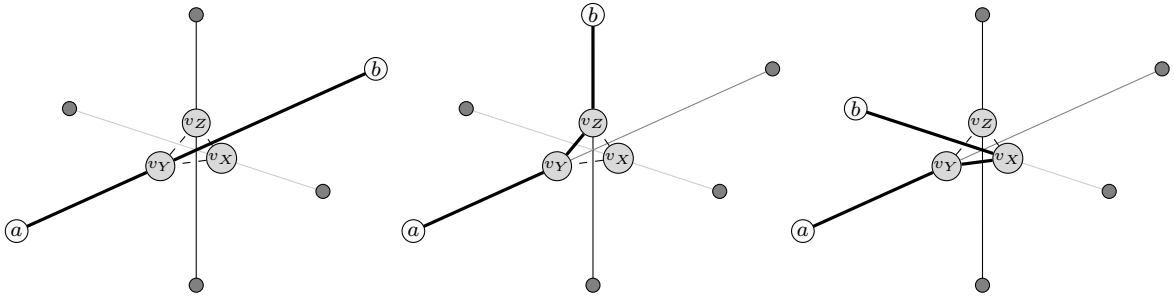


Figure 2.2: Illustration of exploited nodes and virtual edges.

Figure 2.3: Illustration of the use of exploited nodes and virtual edges to model different types of turns in a path from node a to node b .

(respectively, right) image shows a turn from an edge parallel to the Y-axis to one parallel to the Z-axis (respectively, the X-axis), which requires traversing the virtual edge (v_Y, v_Z) (respectively, (v_Y, v_X)), thereby incurring an elbow cost. The resulting graph captures the feasible space for routing pipelines, including directional transitions.

We denote the constructed graph by $\tilde{G} = (V, E)$, where V is the set of virtual nodes and E is the set of edges. The edge set E includes both the virtual edges, denoted by E^v , and the transformed physical edges from the original graph. Since $\tilde{G}$ is embedded in $\mathbb{R}^3$, each node $v \in V$ is associated with its 3D coordinates $(x_v, y_v, z_v) \in \mathbb{R}^3$. We denote by $d_{vv'} = d((x_v, y_v, z_v), (x_{v'}, y_{v'}, z_{v'}))$ the Euclidean distance between two nodes $v, v' \in V$, and by $d_{ee'}$ the Euclidean distance between two edges $e, e' \in E$. This represents either a point in $\mathbb{R}^3$ (if $e \in E^v$) or a line segment (if $e \in E \setminus E^v$).

We assume that a finite set of commodities K is given, where each commodity requires the design of a dedicated pipeline route. Each commodity $k \in K$ is defined by a pair (s^k, t^k) , where $s^k \in V$ is the source and $t^k \in V$ is the destination. For each commodity $k \in K$, a cost structure $c_e^k \geq 0$ is defined over the edges $e \in E$. These costs may depend on several factors such as edge length, elbow penalties, designer preferences, or specific routing priorities. A detailed description of the cost structure used in this study is provided in Section 3.5.3, where we present our case study.

2.3 A multicommodity network flow based model

In this section we describe the mathematical programming model that we propose for solving the PRPS imposing naval design technical requirements. In a first approximation, one can model the problem as a minimum cost multicommodity network flow problem (MCMNFP, for short). See, e.g., [Arslan et al. \[2020\]](#), [Garg and Smith \[2008\]](#) for some other applications of multicommodity flow models. Let us denote by $G = (V, A)$ (where $A = \{(i, j) \cup (j, i) : e = \{i, j\} \in E\}$ is the arc set) the directed version of the graph $\tilde{G}$ described in the previous section. Similar to the non-directed version, we assume that set A includes the set of virtual arcs, denoted by A^v , and the replicas of the physical arcs. We also denote by $d_{aa'}$ the Euclidean distance between the arcs $a, a' \in A$ (notice that $d_{aa'} = 0$ if a and a' are the arcs corresponding to the two opposite directions of the same edge). In addition, we assume that, for each commodity $k \in K$, the cost of each arc $(i, j) \in A$ is $c_{ij}^k = c_{ji}^k = c_e^k$, where $e = \{i, j\}$.

The goal of MCMNFP is to route jointly all the commodities in K from their respective sources, s^k , to their destinations, t^k , through the network G at minimum cost. This model has been widely studied in the literature (the interested reader is referred to [Salimifard and Bigharaz \[2020\]](#) and the references therein for further details on the MCMNFP).

In order to describe the mathematical programming model for the MCMNFP, we use the following set of binary decision variables:

$$x_{ij}^k = \begin{cases} 1 & \text{if arc } (i, j) \text{ is used by the route of commodity } k, \\ 0 & \text{otherwise.} \end{cases}$$

With this set of variables, the overall cost of routing the commodities through the network can be expressed as:

$$\sum_{(i,j) \in A} \sum_{k \in K} c_{ij}^k x_{ij}^k.$$

The following set of linear constraints ensures the correct representation of the problem:

$$\sum_{k \in K} \sum_{j \in V: (i,j) \in A} x_{ij}^k \leq 1, \quad \forall i \in V, \quad (\text{MCMNFP}_1)$$

$$\sum_{j \in V: (i,j) \in A} x_{ij}^k - \sum_{j \in V: (j,i) \in A} x_{ji}^k = 0, \quad \forall k \in K, i \in V (i \neq s^k, t^k), \quad (\text{MCMNFP}_2)$$

$$\sum_{j \in V: (s^k, j) \in A} x_{s^k j}^k = 1, \quad \forall k \in K, \quad (\text{MCMNFP}_3)$$

$$\sum_{j \in V: (j, t^k) \in A} x_{j t^k}^k = 1, \quad \forall k \in K, \quad (\text{MCMNFP}_4)$$

$$x_{ij}^k \in \{0, 1\}, \quad \forall (i, j) \in A, k \in K. \quad (\text{MCMNFP}_5)$$

where [\(MCMNFP₁\)](#) assures that only one commodity is routed through a node and in particular, through an arc, [\(MCMNFP₂\)](#)-([MCMNFP₄](#)) are the flow conservation constraints and [\(MCMNFP₅\)](#) is the domain of the variables.

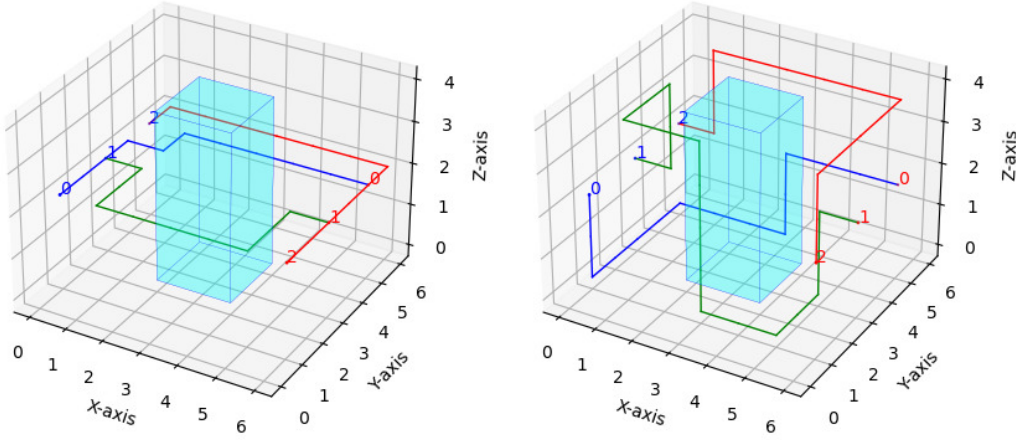


Figure 2.4: Solution of a MCMNFP in the underlined graph without additional modelling constraints (left) and the final solution after adding the additional constraints on distances between services and elbow test (right)

The solution of MCMNFP is a set of paths (one for each commodity) connecting sources with destinations that do not overlap on the graph G . The reader may observe that in the graph G each physical node has been replaced by three virtual nodes on the same coordinates. This explains that in Figure 2.4 (left) some paths occupy the same space although they do not actually overlap on G .

This problem is known to be $\mathcal{NP}$ -hard, even for the case of two commodities (see Even et al. [1975]). Additionally, in pipeline routing there are two main technical requirement that must be considered in order to construct feasible designs: a minimum distance between the pipelines and a minimum distance between consecutive elbows in a single pipeline. The first requirement allows to adequately separate the different services to be used by the pipelines (and such that, when they are enlarged to represent dimensional pipes, they fit on the space) as well as to keep space for the manipulation of machinery in the ship cabins. The second one, is a physical limitation of pipeline routing since there is not enough space for a pipe to turn twice in case consecutive elbows are positioned too close. Note that solutions consists of segments in $\mathbb{R}^3$ with no dimensionality and therefore, these considerations are not taken into account in the MCMNFP. Thus, to impose those technical requirements the following constraints must be added to the problem in order to adequately model the PRPS:

1. *Distance between Services:*

Given an (non-virtual) active edge for a service k , a minimum distance with respect to other services is required. Let R^k be the radius of the cylinder of pipeline $k \in K$ and let Δ^k be the minimum security distance from k to any other element. Then, the minimum distance allowed between services k and k' is $R^{kk'} = R^k + R^{k'} + \max\{\Delta^k, \Delta^{k'}\}$.

This requirement is assured imposing the following set of constraints:

$$\sum_{k' \in K: k' \neq k} \sum_{\substack{a' \in A \\ d_{aa'} < R^{kk'}}} x_{a'}^{k'} \leq M_a^k (1 - x_a^k), \quad \forall a \in A \setminus A^v, k \in K, \quad (\text{Dist})$$

that is, if a is an active arc in the path of service k , then, no arc in the path of any other service $k' \neq k$ can be activated at a distance smaller than the minimum required. Here, fixed $a \in A \setminus A^v$ and $k \in K$, M_a^k is a big enough constant (greater than $2(|K| - 1)$ times the number of the grid edges contained in the cuboid centered at midpoint of arc a and with edge length $\text{length}(a) + 2 \max_{k' \neq k} R^{kk'}$).

2. Elbow Test:

A feasible path for a service, $k \in K$, is required to verify a minimum distance between consecutive elbows, $D^k \geq 0$, of this single service. Since elbows are identified in the non-directed graph with virtual edges and then, with virtual arcs in the directed graph, this requirement can be incorporated to the model as follows:

$$x_a^k + x_{a'}^k \leq 1, \quad \forall a, a' \in A^v : d_{aa'} \leq D^k, a \neq a', k \in K, \quad (\text{Elbow})$$

that is, if two different virtual arcs (elbows) of the same service $k \in K$ are at a distance smaller than or equal to D^k , then only one of them is allowed to be active.

Example 2 shows the effects of considering constraints (Dist) and (Elbow).

Example 2 (Example 1 - continuation). *Let (N, E) be the grid depicted in Figure 2.1 (right). Grid (N, E) is a simplification of the actual graph $G = (V, A)$ used to solve the problem because in the final graph a directed version is considered and each of the nodes in the grid is replaced by its explosion as explained in Section 2.2.*

We have solved the Example 1 scenario with the multicommodity model with and without the additional modelling constraints (Dist) and (Elbow). Observe that the paths without additional modelling constraints do not respect the conditions about distances and thus a much shorter solution is provided (see Figure 2.4 (left)) than in the case in which the technical requirements imposed by means of constraints (Dist) and (Elbow) have to be fulfilled (see Figure 2.4 (right)).

Figure 2.5 shows the final solution with dimensional pipelines of the given pre-specified radii.

In order to tighten the LP relaxation of the problem (and thus leading to a more efficient solution method), constraints (Elbow) can be equivalently formulated as follows. Given a virtual arc $a \in A^v$, we denote by $RE(a) = \{\tilde{a} \in A^v : \tilde{a} \text{ has the same coordinates as } a\}$, namely the set of arcs linking virtual nodes associated to the same physical node as the one linked by a .

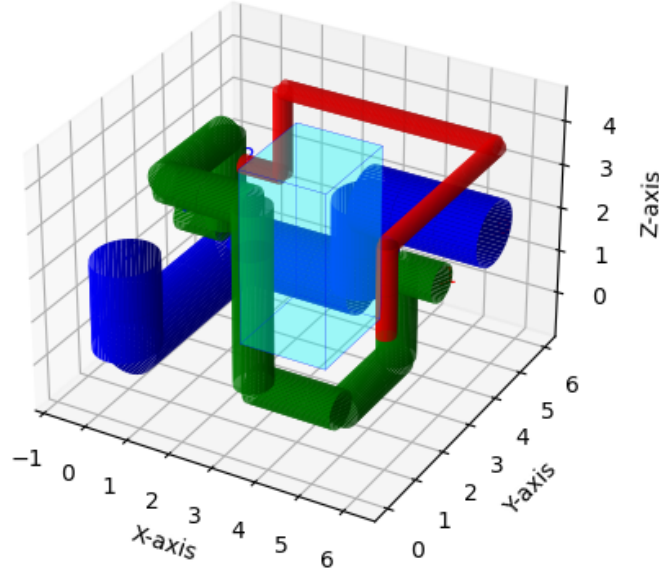


Figure 2.5: A graphical display of the solution of Example 1 with the final pipelines of the three services and their actual radii

Clearly, if $d_{aa'} \leq D^k$, all the arcs in $RE(a)$ and $RE(a')$ are incompatible and then cannot take simultaneously a value of 1. Thus,

$$\sum_{\bar{a} \in RE(a)} x_{\bar{a}}^k + \sum_{\bar{a} \in RE(a')} x_{\bar{a}}^k \leq 1, \quad \forall a, a' \in A^v : d_{aa'} \leq D^k, a \neq a', k \in K. \quad (\text{Elbow} - R)$$

With the above considerations, the PRPS can be formulated as:

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} \sum_{k \in K} c_{ij}^k x_{ij}^k \\ \text{s.t.} \quad & (\text{MCMNFP}_1) - (\text{MCMNFP}_5), \\ & (\text{Dist}), (\text{Elbow} - R). \end{aligned} \quad (\text{PRPS})$$

Problem (PRPS) is a minimum cost multicommodity network flow problem with additional constraints that enforce the fulfillment of the technical requirements for the pipelines in the solution. Although the problem is modeled as an Integer Linear Programming Problem, in real-world situations it has a large number of variables ($\mathcal{O}(|A||K|)$) and a large number of constraints. In particular, the technical constraints (Dist) and (Elbow – R) place a considerably large load leading to a model with complexity ($\mathcal{O}(|A|^2|K|)$).

2.3.1 An exact solution method for (PRPS)

The large dimensions of our model make the straightforward approach of putting it into a MILP solver not possible even for small-sized instances. We overcome this drawback using a branch-and-cut method that incorporates the constraints as they are required, instead of considering initially all of them. This strategy allows an efficient exact solution approach to the problem by means of solving an incomplete (relaxed) formulation with only some of the constraints in the model, while the remaining constraints, required to assure the feasibility of the solutions, are incorporated *on-the-fly* as needed. Although in the worst-case situation, the procedure may need all the constraints not initially included, in practice, only a small number of them are added, reducing considerably the size of the problem.

Specifically, we consider, in the beginning, the relaxed master problem (MCMNFP):

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} \sum_{k \in K} c_{ij}^k x_{ij}^k & (\text{MCMNFP}) \\ \text{s.t.} \quad & (\text{MCMNFP}_1) - (\text{MCMNFP}_5). \end{aligned}$$

The above problem is nothing but the multicommodity network flow model that does not consider the two families of technical constraints that are required for a feasible solution of our problem. Thus, when solving (MCMNFP), one may obtain solutions which do not satisfy constraints (Dist) and/or (Elbow – R). Therefore, each obtained solution must be checked for feasibility. To separate the violated constraints, we apply an enumerative procedure and those which are violated are added to the pool so that the problem is solved again (cutting off the previously obtained solution).

More specifically, for a given solution of the relaxed master problem, we check whether it violates (Dist) and/or (Elbow – R) as follows:

- To check the violation of any of the constraints in (Dist) we proceed by measuring the distance between two arcs from different commodities in the solution or the distance between an arc from a commodity in the solution and obstacles' edges. If the distance measured is smaller than the sum of the radius of both commodities plus the security distance between them, the constraint is violated. In case it is violated for a service $k \in K$ and $a \in A \setminus A^v$, the constraint is added to the constraint pool of the master problem.
- The violation of the constraints (Elbow – R) is tested by first ordering virtual arcs within each commodity in the path given by the current solution and then measuring the distance between two consecutive virtual arcs in the path. If the distance measured is smaller than the minimum required distance between elbows for this commodity then the constraint is violated. In this case, this particular constraint (Elbow – R) is introduced as a feasibility cut to the set of constraints of the problem.

For a more efficient implementation of the procedure, it is embedded in the branch-and-bound tree relaxing constraints (Dist) and (Elbow – R). This procedure, which in solvers is applied by defining those constraints as lazy cuts, results in a constraints relaxation to solve the problem by means of a row generation approach.

2.4 Two matheuristic algorithms for the PRPS

In order to solve the pipeline routing model [PRPS](#) presented in Section 2.3 one has to solve a MILP problem. However, in most cases of real-world situations, due to their large number of variables and constraints, the corresponding problem cannot be solved by a commercial solver (and in many cases it cannot be even loaded). For this reason, we provide in this section two families of matheuristic algorithms for the problem based on two different paradigms: reducing the dimension of the MILP model and decomposing the problem into simpler problems. The pseudocode of both algorithms is included in [A.](#)

2.4.1 Dimensionality Reduction

Recall that we assume that the underlined region where the pipelines are to be routed is represented by a bounded polyhedron in $\mathbb{R}^3$, and that in real-world situations, the polyhedron is usually a cuboid representing a cabin in the ship where some other smaller cuboids (obstacles) have been removed. In addition, there are zones of such a region that are rarely used by the paths in the solutions (by all or some of the services). We exploit this fact in order to reduce the number of variables of our model.

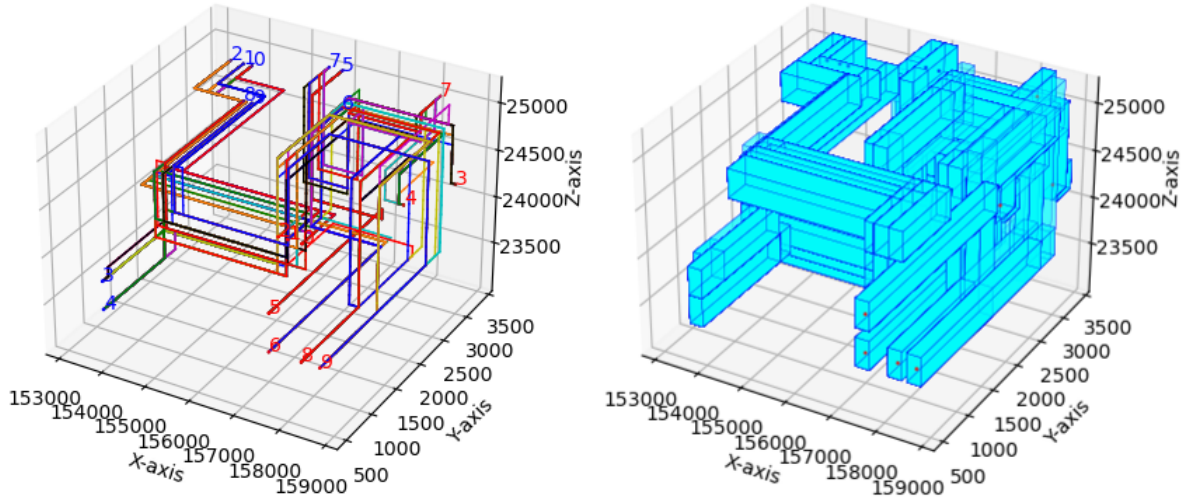


Figure 2.6: Illustration of the initial region used in the dimensionality reduction algorithm

We propose an iterative procedure that starts by considering some tentative areas as candidate zones for searching for the solution (called in the pseudocode `Init-Sol`) and then solving the multicommodity flow problem with additional (`Elbow - R`) constraints for each single commodity to determine the best solution for each service independently of the remainder services. Then, for each commodity we consider a parallelepiped of a given initial dimension around the obtained path and instead of solving ([PRPS](#)) in the whole graph, we solve the problem in the union of these parallelepipeds together with the initial candidate zones (see Figure 2.6 (right)). We do this by fixing to zero the variables indicating arcs outside the above mentioned

regions. In case the problem is feasible, it provides a solution of (PRPS). Otherwise, the dimension of the parallelepipeds is increased and the procedure is repeated until feasibility is obtained. Although in the worst case situation the algorithm requires solving the original instance of the problem (the whole graph), in practice, it allows to solve the problem with a considerably smaller number of variables. Algorithm 1 in A1. provides a pseudocode for this procedure.

2.4.2 Decomposition-based algorithms

Note that when removing the capacity constraints, (MCMNFP₁), from MCMNFP, the problem can be decomposed in $|K|$ single-commodity network flow problems with unitary demands which are equivalent to solving $|K|$ shortest path problems on the undirected graph $\tilde{G}$, and then, solvable in $\mathcal{O}(|V|^2 + |E|)$ time by the classical Dijkstra algorithm Dijkstra [1959]. Based on this idea, we propose an algorithm that exploits the use of this decomposition for solving fast and accurately the PRPS. Algorithm 2 in A2. shows the pseudocode of this algorithm.

First, the algorithm considers a sorted list of all the commodities (pipelines to route), K_0 , and a maximum number of iterations to perform, $maxit$. The iterations are grouped in three different classes: *parallel* iterations (I_par), *sequential* iterations (I_seq) and *cluster* iterations (I_cluster), such that the three sets define a partition of $\{1, \dots, maxit\}$.

The main scheme for all the iterations is similar. At each iteration $it \in \{1, \dots, maxit\}$, a shortest path is built for each commodity included in the sorted list K_{it} trying to pass the elbow test, i.e., the minimum required distance between consecutive elbows in the paths (constraints (Elbow)). This path is constructed using the procedure SPP_ETest (Algorithm 3 in A2.). In this procedure, for each commodity $k \in K_{it}$, a shortest path from the source node s^k to the destination node t^k is initially built on the undirected graph $\tilde{G}$ using Dijkstra's algorithm ($SPP(s^k \rightarrow t^k)$). In case the elbow test is passed, SPP_ETest returns that path. Otherwise, the part of the path that passes the test is kept, the cost of the first elbow that does not pass the test is increased, and Dijkstra's algorithm is run again (with the new costs) from the last elbow passing the test to the destination node (updating the source node for SPP). The process is repeated every time we find an elbow that does not pass the test. If we identify the process is cycling, we initialize it maintaining the increased costs and taking the source node of the commodity as the destination node and vice versa. This last step is repeated until the elbow test is passed or until a maximum number of iterations (of the elbow test) is reached, and then the elbow test has not been passed.

The second step consists of avoiding overlapping of paths in the solution. This phase is different for each of the types of iterations that we consider. In the sequential iterations, paths for commodities are solved one by one in the sorted order and therefore, the sorting of the commodities affects the solution. In these iterations, once the path for commodity k is solved, this path is kept, commodity k is removed from the list of sorted commodities, and overlapping is avoided by increasing for the remaining services $k' \neq k$ in the list, the cost of the edges in conflict, that is, the edges at a distance lower or equal than $R^k + \Delta^k$ from the edges used in the path of

service k . They can be tested using Algorithm 4 described in A2..

In the **parallel iterations**, the paths for the commodities are computed independently, and any overlaps are addressed after all paths have been constructed by increasing the costs of the conflicting edges. *This strategy is similar to the Touch-and-Cross method proposed in Kimura [2017].*

In the **cluster iterations**, paths for all commodities are initially computed in parallel. Commodities whose paths do not overlap with any others are removed from the set of commodities K_{it} , and their paths are retained. The remaining commodities $k \in K_{it}$ are grouped into clusters based on shared conflicting edges and are then sorted according to a predefined criterion—for example, by increasing path length. The first commodity in each sorted cluster is removed from K_{it} , and its path is preserved. For the remaining commodities in K_{it} , the costs of edges that conflict with the retained paths are increased, and new paths are computed for them in parallel. This process is repeated until $K_{it} = \emptyset$ or the maximum number of iterations is reached.

Note that, for all types of iterations, each iteration is performed with a new set of costs for the edges, trying to avoid the use of conflicting edges in the solutions for the next iteration.

In Figure 3.2 we sketch with a flowchart the decomposition approach. There, we denote by $\text{SPP_ETest}(k)$ and $\text{CovList}(k)$ the results of Algorithms 3 and 4, respectively. $\text{Cov}(k, k')$ denotes the set of edges in conflict for services k and k' , i.e. $\text{Cov}(k, k') = \text{CovList}(k) \cap \text{CovList}(k')$.

2.5 Computational experience

Next, we report on the results of some computational experiments that we have. First, we compare empirically the proposed exact and heuristic approaches in a set of 90 random instances that vary the grid density, the number of services, and a number of obstacles within the instance to test the performance, applicability, and flexibility of the proposed algorithms for solving different instances. Second, a case study on one realistic instance provided by our partner Ghenova further illustrates the applicability and performance of the proposed algorithms.

2.5.1 Random instances

The set of random instances is generated as follows. We choose $\mathcal{P}_0$ as a cube of edge length equal to 128 units. An orthogonal 3D grid of $d \times d \times d$ physical nodes, with $d \in \{17, 33\}$ is built on this cube, the distance between adjacent nodes fixed to $\frac{128}{d-1}$ units. This means a spacing of 8 and 4 units for $d = 17$ and $d = 33$, respectively. We will call d the *density* of the grid.

Five different groups of 15 obstacles each are generated with cube shape and edge length $l = 10$ units. Each coordinate of the center of an obstacle is placed in $[\varepsilon + \frac{l}{2}, 128 - \varepsilon - \frac{l}{2}]$ where ε stands for the width of an obstacles-free layer along the facets of $\mathcal{P}_0$ that ensures some feasible origin-destination paths in case too many obstacles are placed. This grid defines a baseline for the undirected graph $\tilde{G}$ described in Section 2.2 and the directed version G described in Section 2.3.

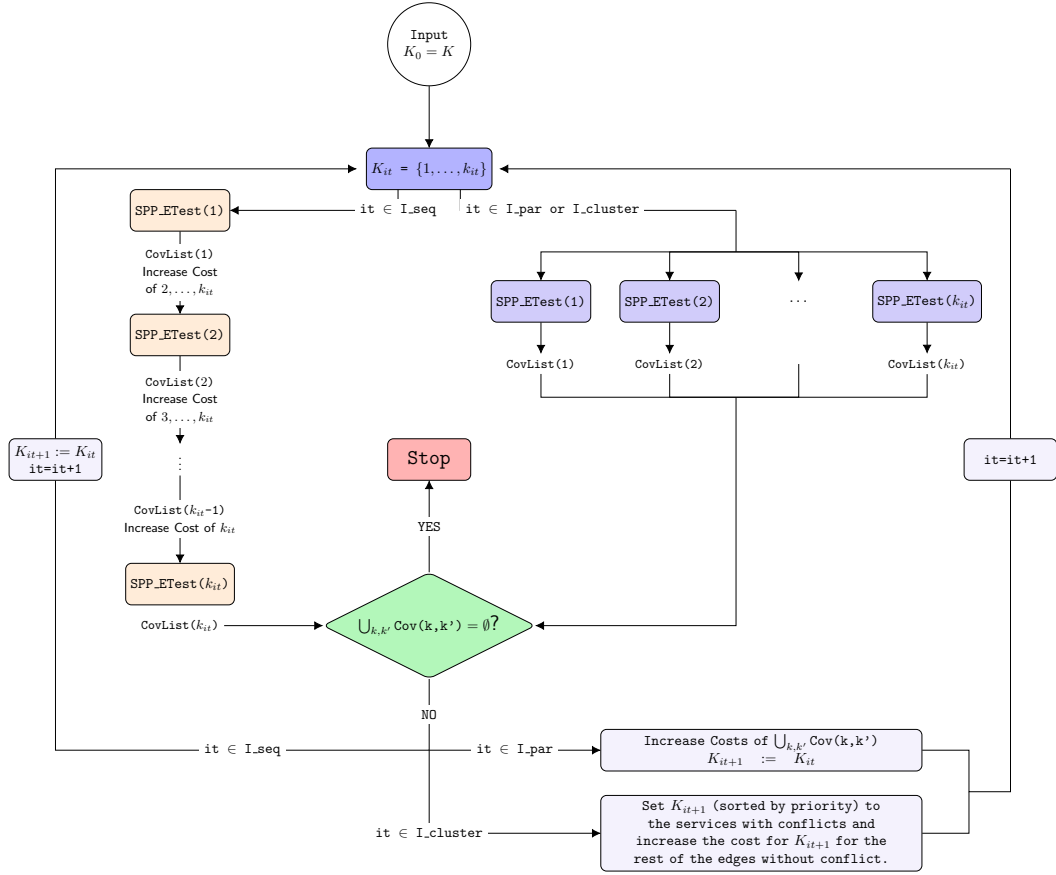


Figure 2.7: Flowchart of the decomposition-based approach

Five different groups of 12 services are generated with equal radius of 4 units and establishing a safety minimum allowed distance of 1 unit with respect to other services. For each service, a source and a destination have been randomly generated belonging to planes $y = 0$ (for the source) and $y = 128$ (for the destination) assuring that both points are nodes of the grid for $d = 17$ and therefore, for $d = 33$.

Several details on real-life cost functions for our problem are given in Section 3.5.3. However, for this preliminary study on random instances, we choose the cost function coefficients for an edge e as:

$$c_e^k = \alpha_1^k (d_e + 10El_e + 2Ch_e),$$

where:

- α_1^k takes random integer values in $[1, 9]$.
- d_e is the physical distance between the two end-nodes i and j of the edge $e = \{i, j\}$ (length of the edge).
- El_e takes value 1 if edge e represents an elbow and 0 otherwise.
- Ch_e takes value 1 if edge e is a vertical edge (in the Z -axis) and 0 otherwise.

Observe that the cost function considers the distance of the paths, the number of elbows introduced, as well as changes in height. This cost system is a simplified version of the general cost function described in Section 3.5.3 for our case study.

We denote by (d, s, o, g) the instance of density $d \in \{17, 33\}$, services $\{1, 2, \dots, s\}$ with $s \in \{5, 8, 12\}$, obstacles $\{1, 2, \dots, o\}$ with $o \in \{5, 10, 15\}$ and creating five instances, $g \in \{1, \dots, 5\}$, for each combination of services and obstacles. Therefore, 90 different benchmark instances are generated.

All instances were solved with the Gurobi 7.7 optimizer, under a Windows 10 environment in an Intel(R) Core(TM)i7 CPU 2.93 GHz processor and 16 GB RAM. Default values were initially used for all parameters of the Gurobi solver and a CPU time limit of 7200 seconds was set. We have also tested different combinations of parameters for the solver cut strategy and dimensionality reduction heuristic but, unless it is specified, the best results were obtained with the parameters of the solver set to the default values. An initial solution was given to the problem by solving the multicommodity flow problem with additional (Elbow – R) constraints for each service independently of the remainder services.

For the decomposition based heuristic, we fix the maximum number of iterations to 20, the first 10% of them of type **I_par**, the next 80% of type **I_cluster** and the last 10% of type **I_seq**. In case a feasible solution is not found with the maximum number of iterations (which only happens for some of the $d = 33$ instances), we generate the solution for the same instance by reducing the density to $d = 17$, which is indeed a feasible solution of the instance.

In Table 3.4 we report the average results of our computational experiments. The first three columns indicate the parameters identifying the instances, d (density), s (number of services) and o (number of obstacles). For each of the combinations of these parameters, we report the average results of the five generated instances. Column Vars indicates the number of variables of the (MCMNFP) problem, and column Cons the number of constraints. Column Solved reports the number of instances out of five solved to optimality by the Exact method (Ex). In the block of columns denoted by GAP, we report the gap obtained with the three proposed procedures, the Exact (Ex) model, the dimensionality reduction algorithm (H1), based on trimming down instances, and the one based on decomposing the problem (H2). The GAP reported for the exact approach indicates the MIP gap obtained at the end of the time limit in case the problem has not been optimally solved, while the gap for the heuristic procedures gives the percent deviation of the heuristic solution with respect to the best solution obtained with the exact approach. Finally, in the block of columns Time, we report the CPU time required by each one of the approaches.

A first analysis of the results shows that the exact algorithm solves all instances with $d = 17$ and any number of services and obstacles, whereas for $d = 33$ we could solve to optimality almost all the instances for the number of services $s = 5$ and 8 but not for $s = 12$ services. Actually, for $s = 12$ services none of the instances could be solved to optimality within the time limit although the final MIP gap is always very small and less than or equal to 1.48%.

On the other hand, the results reported for the heuristic algorithms are rather good. For all the instances, our two heuristic approaches (H1) and (H2) always find

feasible solutions, and the gaps with respect to the best solution found by the exact method (Ex) are less than or equal to 5.73%. Actually, heuristic (H2) reports even better gaps, being always less than or equal to 1.59%.

Concerning running times, as expected, the methods based on solving mathematical programming models, i.e., the exact method (Ex) and the matheuristic (H1), require more time to get to their solutions. Clearly, (H1) is less time consuming than (Ex) since it solves trimmed instances which results in programs with much fewer variables and constraints so that the computing time is also smaller. On the other hand, the decomposition heuristic (H2) which is based on iteratively solving shortest path problems with modified weights is much lighter and the running times are considerably smaller.

d	s	o	Vars	Cons	Solved	GAP			Time			
						Ex	H1	H2	Ex	H1	H2	
17	5	5	228,232	73,572	5	0.00	0.10	0.00	11.15	6.26	1.28	
		10	227,421	73,386	5	0.00	0.32	0.00	11.51	7.80	1.48	
		15	226,736	73,230	5	0.00	2.01	0.00	12.21	9.73	1.40	
	8	5	399,406	117,761	5	0.00	0.31	0.64	400.17	396.38	2.24	
		10	397,986	117,463	5	0.00	1.96	0.22	339.85	240.53	4.37	
		15	396,788	117,214	5	0.00	5.73	0.34	347.41	482.09	4.59	
	12	5	627,638	176,735	5	0.00	0.61	1.59	4216.11	1824.99	3.78	
		10	625,407	176,289	5	0.00	2.64	1.17	3367.94	2159.86	7.20	
		15	623,524	175,914	5	0.00	4.93	1.26	2596.96	2215.58	7.55	
	17			417,015	122,396	45	0.00	2.07	0.58	1225.45	1149.25	3.77
	33	5	5	1,694,166	537,852	5	0.00	0.10	0.00	163.20	3.31	98.08
			10	1,689,282	536,568	5	0.00	0.16	0.00	152.31	5.43	90.72
15			1,684,880	535,407	5	0.00	0.69	0.00	176.34	9.82	98.66	
8		5	2,964,791	860,609	3	0.06	0.24	0.64	5100.74	62.58	183.93	
		10	2,956,243	858,554	4	0.16	1.26	0.15	4393.18	247.27	181.27	
		15	2,948,540	856,697	4	0.16	3.07	0.32	3484.85	1546.39	184.73	
12		5	4,658,958	1,291,007	0	1.26	0.45	1.59	7200.05	236.02	315.66	
		10	4,645,524	1,287,926	0	1.48	1.32	1.17	7200.05	1214.57	311.04	
		15	4,633,420	1,285,139	0	1.38	3.68	1.26	7200.05	4278.22	308.63	
33			3,097,312	894,418	26	0.50	1.22	0.57	3896.75	844.85	196.97	
Total			1,757,163	508,407	71	0.25	1.64	0.57	2576.11	997.05	100.37	

Table 2.1: Computational results for random instances.

2.5.2 Case study

This section is devoted to presenting the application of the proposed methodology to one of the realistic instances provided by our partner Ghenova in order to show the actual applicability of our solution methods to solve scenarios that appear in naval design.

In what follows we describe the elements involved in the instance analyzed in this section, which is drawn in Figure 3.5:

- The space to design the pipelines systems consists of the three-dimensional parallelepiped with widths 5732, 2836 and 2013 units (in the X , Y and Z axis, respectively) which represent a ship cabin.
- A grid was generated for the cabin by subdividing each axis in segments of width 100, producing an initial grid of $58 \times 29 \times 21$ nodes. The graph $\tilde{G} = (V, E)$

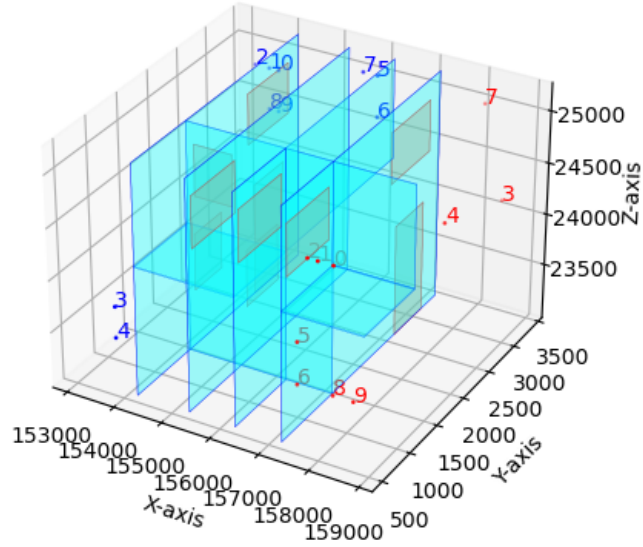


Figure 2.8: Scenario of the case study.

is generated by introducing virtual nodes and edges, with $|V| = 73407$ and $|E| = 282202$.

- Ten services are to be routed in such a cabin, each of them identified with a number from 0 to 9. The sources and destinations of the services are located as needed by the designer through the whole cabin (red and blue numbers/services in the figure, respectively).
- Five obstacles (walls) obstructing the routes are given (light blue shapes in the figure). They consist of metal slices of 10 units width and can be traversed through 8 holes/windows (light orange squares in the figure).
- The minimum distance between consecutive elbows is assumed to be 50 units, as required by the designer.
- The radii of the cylindrical pipes are 50, 75 and 100 units and the minimal security distance among two pipelines is 50 units.

As already mentioned, each edge in the generated grid for the instance has an associated cost that allows evaluating the feasible routes to be traced in the network. The cost system incorporates the preferences of the designer when routing the pipelines. We consider an additive cost structure for the objective function to model the use of edges and elbows in the route of each commodity with the following shape:

$$c_e^k = (\alpha_1^k + \alpha_5^k Pr_e) d_e + \alpha_2^k El_e + \alpha_3^k H_e + \alpha_4^k Ch_e + \alpha_6^k Pc_e + \alpha_7^k Cl_e,$$

Criteria	Description
d_e	physical distance between end-nodes i and j of edge $e = \{i, j\}$ (length of the edge).
El_e	1 if edge e represents an elbow and 0 otherwise.
$H_e = MH - h_e$	being MH the maximum height of the ship cabin and h_e the height of edge e in case it is in a plane parallel to the XY -plane and 0 otherwise.
Ch_e	1 if edge e is a vertical edge (in the Z -axis) and 0 otherwise.
Pr_e	1 if edge e belongs to a preference zone and 0 otherwise.
Pc_e	1 if edge e crosses a penetrable zone and 0 otherwise.
Cl_e	1 if edge e represents an elbow and it is close to a source or a destination point and 0 otherwise.
Parameters	Description
α_1^k	Cost per unit length.
α_2^k	Cost of an elbow.
α_3^k	Cost of moving away from the ceiling.
α_4^k	Cost of changing in z-coordinates (height). Therefore, moving to a different height is penalized.
α_5^k	Bonus per routing the pipeline in a preference zone ($\alpha_5^k < 0$; $\alpha_1^k + \alpha_5^k > 0$).
α_6^k	Cost of crossing a penetrable zone.
α_7^k	Cost of locating an elbow close to the source or destination point of a pipeline.

Table 2.2: Criteria and parameters involved in the cost function of the case study.

for each edge e and each service k . In this function, there are some α -parameters affected by the characteristics of the edge. The criteria and parameters that define the cost system are detailed in Table 3.7. This cost structure is flexible enough to reflect most of the preferences of naval designers and was determined in view of the criteria exposed by our partner company for the selection of implementable solutions.

Example 3. *In order to illustrate the above cost function, we use the part of the graph drawn in Figure 2.9.*

In this graph, Π_1 is the plane $\{(x, y, z) : z = 1\}$ while $\Pi_2 = \{(x, y, z) : z = 0\}$ and then, plane Π_1 is closer to the ceiling than plane Π_2 . We denote by v_1 and v_2 the nodes which are virtualized in the figure (for each of the hyperplanes). We assume that there is a preference zone using Π_1 but there are no preference zones using Π_2 . Furthermore, v_2 is assumed to be close to the source or destination point of some service and also that the black edge in Π_2 crosses a penetrable zone for the service. In this situation, we have:

- *The criteria d and El coincide for all the edges that belong to the planes Π_1*

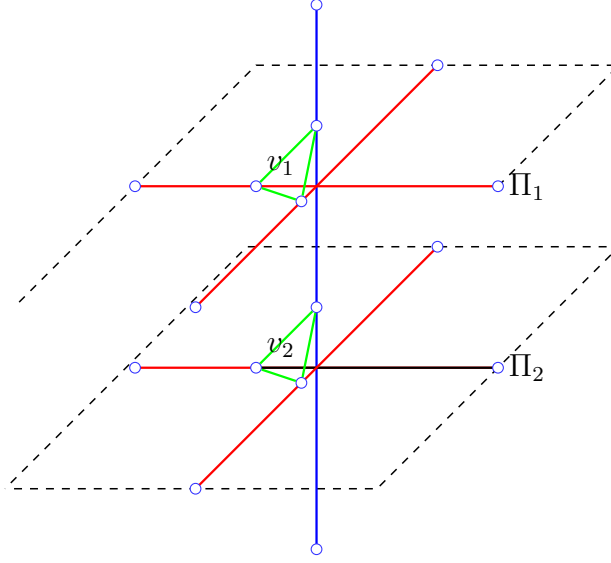


Figure 2.9: Graph used to illustrate the cost function in Example 3.

	Virtual Arcs (Elbows)	Non Virtual Arcs (Straight Pipes)
$e \in \Pi_1$	α_2^k	$(\alpha_1^k + \alpha_5^k)d_e + \alpha_3^k H_e$
$e \in \Pi_2$	$\alpha_2^k + \alpha_7^k$	$\alpha_1^k d_e + \alpha_3^k H_e$ $\alpha_1^k d_e + \alpha_3^k H_e + \alpha_6^k$ (penetrable edges)
e parallel to Z -axis		$\alpha_1^k d_e + \alpha_4^k$

Table 2.3: Illustration of the cost function.

and Π_2 , being the part of their costs affected by parameters α_1 and α_2 , the same.

- The height of the edges in Π_1 is larger than height of the edges in Π_2 , being the term $\alpha_3 H$ smaller for the edges in Π_1 .
- The criterion Pr affects the edges in Π_1 but not those in Π_2 .
- The criteria Pc and Cl affect some of the edges in Π_2 but none of those in Π_1 .
- The edges linking nodes in Π_1 with nodes in Π_2 (parallel to the Z -axis) are only affected by criteria d and Ch .

In Table 2.3 we summarize the cost function for edges in the different planes depicted in Figure 2.9.

In the actual instance considered in the case study, each edge in the graph $\tilde{G}$ has the same cost for all services, i.e., $c_e^k = c_e^{k'}$, for all $k, k' \in K$, as required by our partner company. The parameters provided for this instance were:

α_1	α_2	α_3	α_4	α_5	α_6	α_7
1	2800	700	200	-0.7	4000	3000

and the preference zones were defined as the following parallelepipeds: $[154241, 158844] \times [535, 1419] \times [24249, 25221]$ and $[152013, 154241] \times [535, 3371] \times [24249, 25221]$.

We run our model for this instance with the exact and the different heuristic approaches. A time limit of 12 hours was fixed for all the procedures. The exact branch-and-cut approach was not able to find a feasible solution to the problem within the time limit. Note that the number of variables for this problem is 2819480 and the (initial) number of constraints is 31014280 (constraints (Dist) and (Elbow – R) that are added as lazy constraints in the branch-and-cut approach are not counted here). The number of explored nodes after 12 hours was 1804, which gives an idea of the high computational load required to solve the problem at each node of the branch-and-bound tree.

For the dimensionality reduction matheuristic (H1), we first solve, independently, the multicommodity flow problem with additional (Elbow – R) constraints for each single service, and restrict the search region of our problem to the region induced by these paths enlarged by a parallelepiped of dimension $\delta_k = \max\{R^{k'} + \Delta^{k'} : k' \in K\}$ around the paths. The value of δ_k is sequentially enlarged by 0.1 units until a feasible solution is obtained. This procedure obtains a solution in 132.37 seconds, and the last problem explored 18041 nodes in the branch-and-bound tree. The number of variables of this problem was 473292 (16.78% of the variables of the exact approach) and the initial number of constraints was 807657 (2.6% of the number of constraints required in the exact approach). The number of lazy constraints added in the branch-and-cut approach was 40800 (12 of them of type (Elbow – R)). The objective value of the solution was 448690.78.

For the decomposition based heuristic (H2) we fix to 10 the maximum number of iterations with the first 10% of them of type I_par, the next 20% of type I_cluster and the last 70% of type I_seq. This configuration was adequately tuned using a simplified instance for the problem as a training sample. This heuristic computed a solution in 94.54 seconds after 6 iterations and the obtained objective value was 433387.86. Observe that the deviation between the solutions obtained with the two heuristics was 3.41%.

We draw in Figure 3.6 the obtained solution (with heuristic H1) as paths in the graph. In the picture, we represent the obtained paths without (left) and with obstacles and holes (right). The obtained solution as actual pipelines is drawn in Figure 2.11. One can observe from the picture that real-world instances require routing intricate pipelines, avoiding obstacles and respecting the space between the pipes, which really makes it difficult to locate them in the cabin. The heuristic approaches produce, in view of the results obtained with the synthetic instances, good quality solutions in reasonable CPU times, providing the naval designer with a powerful decision-aid tool for this task.

2.6 Conclusions

This chapter develops a general methodology for the optimal automatic network design of different commodities incorporating real-world constructability conditions. This methodology was originated by an actual problem consisting of the design of

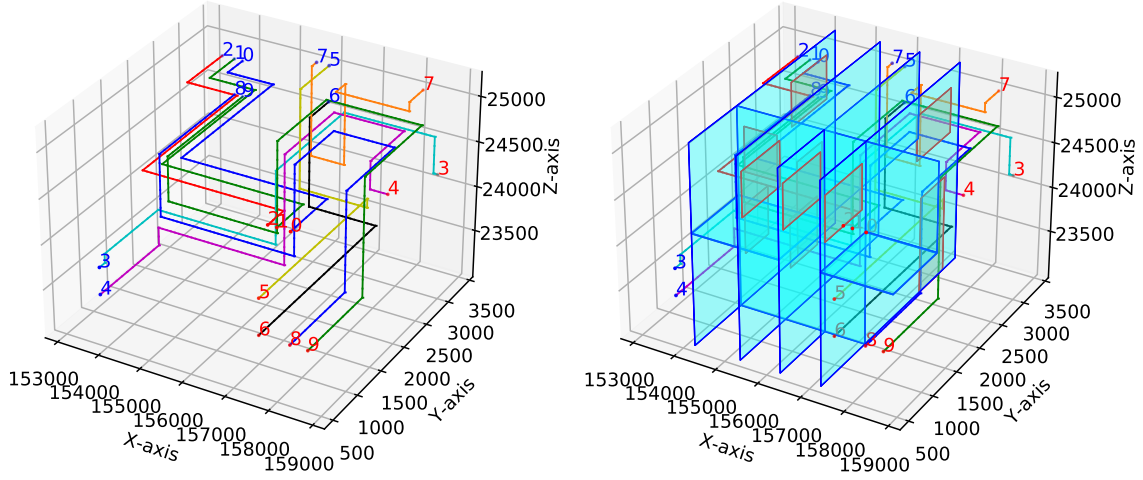


Figure 2.10: Graphical display of the solution for the case study with 10 services. In the left it is shown the solution without showing the obstacles. The figure on the right shows the same solution integrated within the set of obstacles in the scenario.

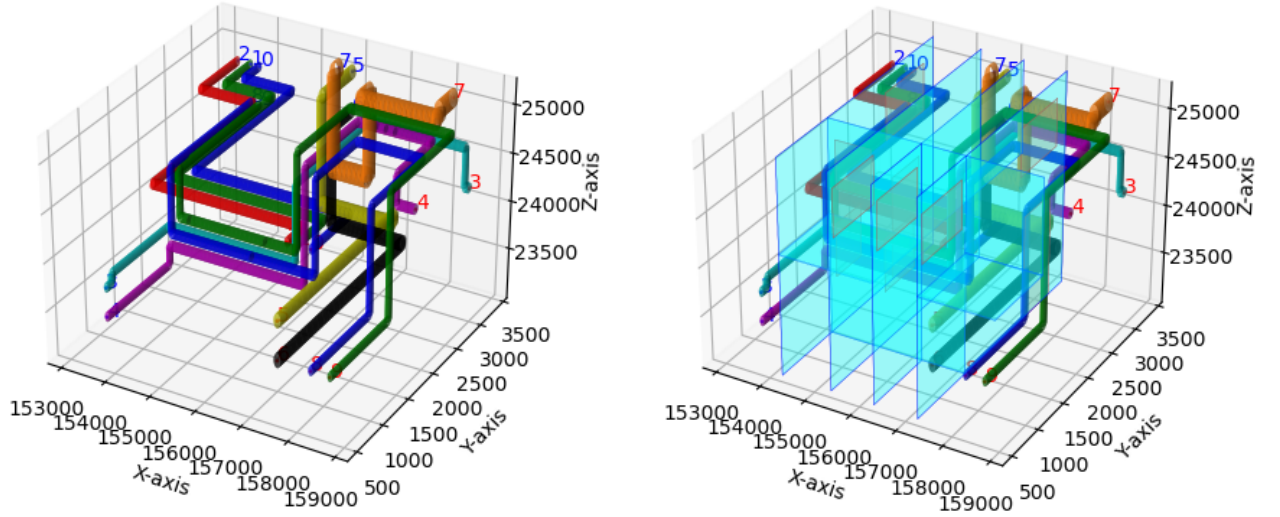


Figure 2.11: Graphical display of the solution for the case study with 10 services actual pipelines. In the left it is shown the set of pipelines without showing the obstacles. The figure on the right shows the same solution integrated within the set of obstacles in the scenario.

spatial pipelines with ships cabins motivated by a recent collaboration with the company Ghenova, a leading naval engineering company. Our proposal is to adapt an ‘*ad hoc*’ minimum cost multicommodity network flow model to the problem that includes all the technical requirements of feasible pipelines routing. The large number of variables and constraints of real-sized instances makes it inappropriate (impossible in real-world instances) to load into the solver the complete model so that we implemented a branch-and-cut algorithm, where, initially, only the standard multicommodity flow model is considered and additional technical constraints are separated within the branch-and-bound tree as lazy constraints. On top of that, we also develop two heuristic algorithms that provide rather good feasible solutions. The first one is a matheuristic based on solving trimmed instances and the second one is a decomposition heuristic that iteratively solves shortest path problems with modified edge weights. Our computational results on randomly generated instances are rather promising, showing that we can solve to optimality medium-sized instances. We also report a case study based on a realistic naval instance of a ship cabin provided by our partner company to our research group to test the methodology.

Future work on this topic includes the consideration of more general underline networks including non-orthogonal designs, which are conceptually easy to handle once the admissible solutions graphs are generated, whereas the generation of these graphs is still a challenging question that will be the focus of future work.

We will also explore the use of Machine Learning tools to determine, in advance, the parameters of the cost function, that are assumed in this work to be given by the designer. The study of the subregions of α -parameters inducing the same routes will allow us to provide the designers different options of reasonable parameters to explore in the decision-making process.

A. Pseudocode of the algorithms described in Section 2.4

We gather in this section the pseudocode of the different algorithms described in Section 2.4 devoted to our matheuristic algorithms.

A1. Pseudocode for the dimensionality reduction algorithm in Section 2.4.1

Next, we include the pseudocode of the dimensionality reduction algorithm.

Algorithm 1: Dimensionality reduction algorithm.

input : Initial set of candidate zones provided by user $\{\text{Init-sol}_k\}_{k \in K}$.
 $\{\delta_k = R^k + \Delta^k\}_{k \in K}$ initial dimension of the parallelepiped around
the best path for commodity k .
it=1. stop = 0.

1 **for** $k \in K$ **do**
2 Solve the Multicommodity Flow Problem with additional (Elbow – R)
constraints for the single commodity k in the whole graph G :
 $\{\text{path}_k\}_{k \in K}$

3 **while** stop = 0 **do**
4 Solve (PRPS)(it), where all variables x_{ij}^k for a given service k , that are
outside Init – sol_k union with the parallelepiped of dimension δ_k
around path_k , are fixed to zero.
5 **if** (PRPS)(it) *is feasible* **then**
6 stop = 1
7 **else**
8 Increase δ_k for all k in K
9 it=it+1

output: Solution of (PRPS)(it)

A2. Pseudocode for the decomposition-based algorithm described in Section 2.4.2

In the following, we describe the different modules of the decomposition-based matheuristic.

Algorithm 2: Decomposition-based Algorithm.

```

input :  $stop := 0$  (termination criterion);  $it=0$  (iterations counter);
          $\{1, \dots, maxit\} = I\_par \cup I\_seq \cup I\_cluster$  (Index set for
         iterations);  $K_0$  (sorted set of all services in  $K$ ).
1 while  $stop = 0$  and  $it \leq maxit$  do
2   if  $it \in I\_seq$  then
3      $K_{it} = K_0$  and  $\bar{K} = K_0$ .
4   for  $k \in K_{it}$  do
5     Apply the SPP_ETest to  $k$ . (Algorithm 3)
6     Update  $CovList(k)$ . (Algorithm 4)
7     if  $it \in I\_seq$  then
8        $\bar{K} = \bar{K} \setminus \{k\}$ . For all  $k' \in \bar{K}$ , slightly increase the cost of edges
       linking vertices in  $CovList(k)$ .
9   for  $k \in K_{it}$  and  $k' \in K$  do
10    Compute  $Cov(k, k') := CovList(k) \cap CovList(k')$ .
11  if  $Cov(k, k') \neq \emptyset$  for some pair  $(k, k')$ ,  $k \in K_{it}$ ,  $k' \in K$  then
12     $stop = 0$ .
13    if  $it \in I\_par$  then
14      For all service  $k \in K_{it}$ , increase the cost (to the most costly
      service) of edges linking vertices in  $\bigcup_{k' \in K} Cov(k, k')$ .
15    if  $it \in I\_cluster$  then
16       $K_{it+1} = \emptyset$ ;
17      Sort the services,  $k_1 \succ k_2 \succ \dots \succ k_\ell$  (by priority);
18      if  $Cov(k_i, k_j) \neq \emptyset$  and  $k_i \succ k_j$  then
19         $K_{it+1} = K_{it+1} \cup \{k_j\}$ .
20      For all service  $k \in K_{it+1}$ , increase the cost (to the most costly
      service) of edges linking vertices in  $\bigcup_{k' \in K \setminus K_{it+1}} Cov(k, k')$ .
21     $it = it + 1$ 
22  else
23     $stop = 1$ 

```

Algorithm 3: SPP_ETest(k).

input : $N^0 := s^k$ (source node for service k); $N^f := t^k$ (destination node for service k); $P_k = \emptyset$ (initial path for service k); $test := 1$ (initial infeasibility test).

- 1 **while** ($test = 1$) **do**
- 2 $P_k := P_k \cup \text{SPP}(N^0 \rightarrow N^f)$;
- 3 **if** (Elbow) is verified by P_k **then**
- 4 $test := 0$ and **return** P_k
- 5 **else**
- 6 Identify previous node that has passed the elbow test: i_0 ;
- 7 $N^0 := i_0$;
- 8 $P_k := P_k \setminus \text{SPP}(i_0 \rightarrow N^f)$;
- 9 Increase the cost of the elbow not verifying the elbow test;

Algorithm 4: CovList(k).

input : P_k = path connecting source node s^k and destination node t^k ;
 $L_k = P_k$ and $\delta_k = R^k + \Delta^k$ (covering of the pipeline).

- 1 **for** $v_0 \in P_k$ **do**
- 2 **for** $v \in L_k$ **do**
- 3 **while** there exists a vertex v' adjacent to v such that $d_{v'v_0} < \delta_k$ and $v' \notin L_k$ **do**
- 4 $L_k = L_k \cup \{v'\}$;
- 5 **go to line 2.**

output: L_k .

Chapter 3

The pipelines and cable trays location problem in naval design

This chapter deals with the determination of optimal locations for pipelines and cable trays in naval design. The problem consists of finding the number and types of cable tray routes to be created between various devices in order to minimize a user-defined cost function. We reduce the problem to an *ad hoc* min-cost multicommodity flow problem with additional constraints imposed by technical requirements. This problem is solved for small-sized instances by using off-the-shelf optimization solvers. We also develop an exact relax-and-cut strategy that allows us to handle medium-sized instances. For larger instances, we propose a family of heuristic algorithms consisting of the combination of two phases: I) Construction of initial cable tray paths; and II) Transformation to feasible cable trays verifying the technical requirements. For each of them, we also propose different strategies which give rise to several algorithms. These algorithms are compared in a computational experience using two types of instances: the first one based on random instances of different sizes and the second one based on instances with well-defined corridors to assess the availability of our methodology to enforce the creation of cable trays. Finally, we also analyze a real-size case study provided by our industrial partner, Ghenova, a leading Naval Engineering company, validating our proposal to find solutions for this problem.

3.1 Introduction

3.1.1 Motivation

Routing pipelines or cables *optimally* is a recurrent problem in naval design and the electrical industry since it is the most important activity during the detail-design phase. Actually, it takes over 50% of the total detail-design person-hours and all other activities of detail design depend on it [Park and Storch, 2002]. Designers face the demanding task of strategically placing multiple elements while ensuring obstacle avoidance, providing adequate machinery space, and meeting technical requirements for compatibility and manufacturability. These objectives must be accomplished within a confined space, leaving little room for maneuvering.

Certain elements, such as water pipes and electrical circuits, require separation, while others, like power, phone, or optical fiber cables, can be consolidated using cable trays. These trays effectively reduce the spatial footprint by grouping elements of the same type together. Although software tools exist to aid engineers in visualizing pipe placement and identifying overlaps or interferences, achieving an optimal pipe and cable layout still heavily relies on experienced designers, who often command higher salaries, leading to increased expenses. By automating the design process for pipelines and cable trays, costs are reduced as the need for a specialized designer dedicated to this task is eliminated. This automation holds the potential for achieving a better, or even optimal, design in a more cost-effective manner. Additionally, it would significantly decrease the time required to complete this task and ultimately result in safer designs.

The **P**ipelines and **C**able **T**rays **L**ocation **P**roblem (PCTLTP) consists of finding the overall number and types of pipeline and cable tray paths to be created between various devices and equipment in a reduced space (building, ship, etc.) under constructability constraints in order to minimize a user defined cost function. This function usually depends on the total length and on the number of changes of direction (*elbows*) in the path followed by the pipes and cable trays. Although initially these paths must be traced in a continuous three-dimensional region, a discretization of the entire space is desirable in order to mathematically address the problem. One of the possible options is to discretize the space by means of a graph structure. Thus, in case no further additional constructability constraints are required and a single pipeline or cable tray is to be routed, this problem is equivalent to finding an optimal path between two nodes in a graph, and solutions can be found in polynomial time using Dijkstra's algorithm [Dijkstra, 1959]. However, in these types of problems many pipelines and cable trays must be routed together, and specific constructability constraints are also required. For instance, the length of the cable that can be accommodated when laying lines (wiring tray) should be upper bounded, a minimum distance between consecutive elbows of the same cable tray, and between different cable trays and pipelines, must be assured, obstacles must be avoided, some zones are preferable, etc., making the problem extremely hard to solve. Therefore, the PCTLTP poses a significant mathematical and computational challenge.

In this chapter, we propose a suitable mathematical optimization-based tool for the PCTLTP in naval design, considering not only the joint arrangement of several elements of the same type through cable trays but also the technical feasibility requirements of the design.

3.1.2 Literature Review

The pipeline location problem in naval design has been widely considered in the literature since it is a fundamental and challenging problem in the design of large ships. The main contribution of most proposals for this problem is the development of metaheuristic algorithms for determining the *optimal* routes of multiple pipelines in an intricate and obstructed three-dimensional (3D) space. There are two elements that are shared among the different works on this topic. Firstly, most of the works agree that the most efficient way to handle this problem in a continuous region (the

3D ship layout) is by appropriately discretizing the solution space. Thus, the first phase is to derive a suitable discrete space to represent the feasible solutions for the problem. Secondly, most of the approaches are based on decomposing the multiple pipeline routing problem into single pipeline problems and applying a shortest-path-like algorithm to solve them.

Concerning the discretization of the space, different strategies have been proposed. In Lee [1961], the author develops a method based on partitioning the region into square cells and removing cells containing obstacles. Then, paths are constructed in the remaining continuous regions. This technique has been applied to the pipeline routing problem for several authors [see, e.g., Ando and Kimura, 2011, Asmara, 2013, Asmara and Nienhuis, 2006b, Kim et al., 2013]. On this domain, one can apply several strategies to construct pipeline paths, as a maze algorithm [Lee, 1961, Rourke, 1975], or an escape algorithm [Hightower, 1969b]. On the other hand, Guirardello and Swaney [2005] proposes a network-based discretization scheme to reduce the dimension of the solution space, where each vertex in the network models a junction of a feasible piperack structure. In this space, routing a single pipeline reduces to finding the shortest path between its source and its target. In Park and Storch [2002], the authors propose a mixture of the above two methods, where a network-based cell-generation method is developed.

Jiang et al. [2015] proposes an ant colony optimization heuristic for the pipeline routing problem. In Dong et al. [2022], a multiobjective framework for the pipe route design is considered. Also, last chapter addresses the problem of routing multiple pipelines using an adapted min-cost flow formulation and proposes different matheuristic algorithms.

In most of Chapter 2, the pipelines are routed while avoiding overlapping. However, in realistic situations, it is allowed to jointly route similar elements through cable trays. These devices help reduce the total cost and required space. As a result, they have been recognized as a common and advisable practice, not only in naval design [Park et al., 2020] but also in building constructions [Pogorelskiy and Kocsis, 2022]. Nevertheless, the literature on the PCTLTP is scarce. Once the clusters of cables/pipelines to be routed through the same cable tray are determined, one can apply any of the available approaches for the pipeline routing problem. Castorani et al. [2018] propose a two-phase approach for this problem. In the first phase, the arrangement of the cables within the trays is decided, and in the second phase, the routing of the cable trays is determined using a genetic algorithm. This strategy may provide suboptimal solutions since the determination of the routes and the decisions on the contents of the trays are computed separately.

In this chapter, we analyze the PCTLTP in a single mathematical optimization model for the first time and provide efficient strategies to obtain high-quality solutions. This model extends the one proposed in Chapter 2 by incorporating cable trays. The inclusion of cable trays in the design of pipeline routes significantly increases the complexity of the problem. This complexity arises from the need to determine not only the routes of the pipelines but also the placement of the trays and the allocation of pipelines or cables to each tray, considering the technical requirements of both the pipelines and the trays. All these decisions are optimally made by minimizing the overall space occupied by the pipelines and the cable trays.

3.1.3 Contributions

The main contributions of this work are:

- A mathematical programming formulation for the PCTLTP as a generalized Minimum Cost Multicommodity Network Flow Problem incorporating several technical requirements.
- The design of an exact *ad hoc* relax-and-cut procedure for solving the problem by relaxing some of the difficult technical constraints.
- A novel family of math-heuristic approaches for the problem based on two phases: search of potential pipeline and cable trays routes and transformation of these initial routes into feasible pipeline and cable trays routes verifying the technical requirements. Different strategies are proposed for each of the phases that can be adequately combined to derive different heuristic methods suitable to obtain good quality feasible solutions in reasonable CPU time.
- Extensive computational experiments in synthetic instances in order to compare the different solution approaches.
- The validation of our proposals on a real instance provided by our industrial partner.

3.1.4 Organization

The chapter is organized as follows: [Section 2](#) describes the main elements involved in the PCTLTP and their mathematical representation. [Section 3](#) presents the mathematical programming model for the PCTLTP in naval design, as well as the relax-and-cut method proposed for solving the problem optimally. Several matheuristic algorithms are provided in [Section 4](#), which can be combined to solve large-sized instances. In [Section 5](#), we report the results of computational experiments. Additionally, a case study built from a real scenario provided by our industrial partner is included. Finally, in [Section 6](#), we present the conclusions drawn from this work and outline future research directions.

3.2 Main elements of the PCTLTP

In the PCTLTP, the objective is to route multiple pipelines and cable trays (services) optimally through a complex and obstructed space (solution space), while minimizing certain design costs. In this section, we will describe the key elements of the PCTLTP and establish the notation that will be used throughout the rest of the document.

3.2.1 Services

In naval design, PCTLTP involves connecting several devices or equipment by means of pipelines and cables, giving rise to a finite set of services of different nature (gas

pipes, water pipes, drainage pipes, optical fiber cables, power cables, air conditioning pipes, etc.) that are to be located. Each of these services is classified into a type from a finite set $\mathcal{T}$, so that, for a given $t \in \mathcal{T}$, services of type t are allowed to be jointly routed through the same tray. On the other hand, services of different types must ensure certain distance-based requirements. For instance, services such as power, phone, or fiber optic cables are considered to belong to the same type t , and it is reasonable to assume that they can be traced jointly in the same tray. In contrast, services such as water pipes and power cables clearly belong to different types of services and must be conveniently separated in the design.

We assume that each service of type $t \in \mathcal{T}$ is characterized by a finite set of sources, each of them with a given integer amount of out-flow, and a finite set of sinks, each of them with a given integer amount of in-flow, such that the overall out-flow coincides with the overall in-flow. For each type $t \in \mathcal{T}$, this flow represents the number of services of type t to be routed.

We will denote by $\{s_1^t, \dots, s_{p_t}^t\}$ the set of sources, by $\{d_1^t, \dots, d_{q_t}^t\}$ the set of sinks, by $g_{s_i^t}$ for $i = 1, \dots, p_t$, the number of services of type t sent from origin s_i^t (out-flow at source s_i^t) and by $g_{d_j^t}$, for $j = 1, \dots, q_t$, the number of services of type t sent to destination d_j^t (in-flow at sink d_j^t) assuming that, $\sum_{i=1}^{p_t} g_{s_i^t} = \sum_{j=1}^{q_t} g_{d_j^t} \forall t \in \mathcal{T}$.

3.2.2 Solution space

As introduced in Section 2.2, we model the routing space as a discretized 3D grid derived from an obstacle-free polyhedral region. Each grid node is exploded into a set of virtual nodes to account for directional transitions and associated elbow costs. In this section, we formalize the resulting graph structure and define the elements involved in path computation.

Let $\tilde{G} = (V, E)$ be the routing graph, where each node $v \in V$ represents a specific location and direction in the 3D grid, and each edge $e \in E$ corresponds to a possible movement between two such states.

To model the routing cost accurately, we associate with each edge $e \in E$:

- A routing element $RE(e)$ capturing the movement type (e.g., straight, bend).
- A cost function $d_{ee'}$ that penalizes transitions between adjacent edges e and e' based on their geometric relationship—particularly emphasizing the cost of direction changes or elbows.

This formalization provides a precise and compact structure over which path-finding algorithms can operate, enabling efficient evaluation of routing constraints and objectives.

3.2.3 Cost structure

The cost structure used in the design of the pipeline and tray routing is defined in detail in the case study presented in Section 3.5.3 of the previous chapter. It incorporates various criteria such as pipeline length, elbow usage, height preferences, and traversal of specific zones, reflecting the preferences of naval designers.

In particular, each edge in the graph $\tilde{G} = (V, E)$ is associated with a cost function that accounts for these factors, and a unified set of cost parameters is used across all services in the instance considered. Virtual edges are used to model direction changes (elbows), while physical edges represent actual pipeline segments with length costs. The complete description of the cost components, parameters, and their role in the objective function is detailed in Table 3.7 and illustrated through Example 3 in the case study section.

3.2.4 The Pipelines and Cable Trays Location Problem

The PCTLP in a ship that we analyze here, consists of routing the different types of services, by determining the location of the possible trays concentrating the flow of the services of the same type in part of the route, and the different connections between the pipelines/cables and the trays by minimizing the overall design cost, as well as verifying the technical constructability requirement. Specifically, a minimum distance between consecutive elbows of a pipeline/tray and a minimum security distance between services of different types are required.

We denote by D^t the minimum allowed distance between consecutive elbows on a pipeline/cable tray of services of type $t \in \mathcal{T}$, by R^t the width of the pipeline/cable of a single service of type t , by Δ^t the minimum distance between services of type t and services of any other type, and by $\Delta^{tt'} := \max\{\Delta^t, \Delta^{t'}\}$ the minimum distance between services of type t and t' .

In Figure 3.1, we show a feasible PCTLP solution for a small example. There, we assume that services of two types are to be routed (services of color **blue** and **green**). Services of type **blue** have four different origins (each of them with out-flow 1) and two sinks (each of them with in-flow 2) available to route flows, while services of type **green** have two origins and two sinks (two sources with out-flow of 2 and 1 each, and two sinks with in-flow 1 and 2 separately). In the left picture, the origins and sinks are depicted as squares ($\square$) and crosses ($\times$), respectively. As can be observed in the picture, the services of the same type are jointly routed in a tray in part of their paths from the origins to the sinks. This solution is drawn (left) as a union of edges (a path) in the graph, but they represent dimensional pipelines or trays in the cabin of the ship. This might cause incompatibilities between the different types represented in each edge if the minimum safety distance between services of different types is not met. In Figure 3.1 (right), we show the 3D representation of the solution once we have adapted the volume of the pipelines/trays to their actual size.

3.3 A mathematical programming model for the PCTLP

In this section, we describe the mathematical programming model that we propose for the PCTLP described in the previous section. We model the problem as an adapted **Minimum Cost Multicommodity Network Flow Problem** (MCMNFP) with additional constraints, that include, among other features, the pipelines/cable

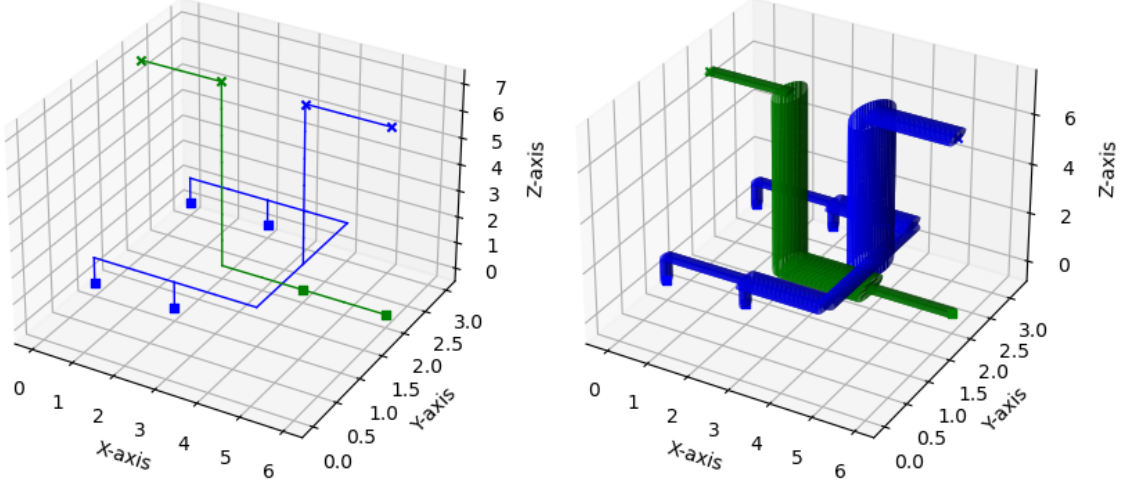


Figure 3.1: Feasible solution for a small instance of the PCTLTP: as a union of edges in the graph (left) and its 3D actual representation (right).

trays layout. The goal of MCMNFP is to simultaneously route all the types of services from sources to sinks, through a network at minimum cost. This model has been widely studied in the literature and has been applied to different industrial problems [Garg and Smith, 2008, Guimarães et al., 2022]. The interested reader is referred to Salimifard and Bigharaz [2020] and the references therein for further details on the MCMNFP.

In our model, (integer) flow variables, f_{ij}^t are defined for each type $t \in \mathcal{T}$ and each arc $(i, j) \in A$. We assume that a single unit of flow is to be routed for each service of type $t \in \mathcal{T}$ and thus, these variables indicate the number of services of type $t \in \mathcal{T}$ routed through the arc $(i, j) \in A$.

We denote by F^t the maximum flow of type $t \in \mathcal{T}$ allowed to be routed along any edge of the graph, i.e., the maximum number of services of type $t \in \mathcal{T}$ that can be jointly routed in the same cable tray.

Flow variables together with the following binary variables, also used in our model, allow us to detect the routes of each of the pipelines/cable trays in the solution.

$$x_e^t = \begin{cases} 1 & \text{if edge } e \text{ is used by the route of services of type } t, \\ 0 & \text{otherwise.} \end{cases} \quad \forall e \in E, \forall t \in \mathcal{T}.$$

Let $V' \subset V$ be the subset of vertices in the graph which are neither sources nor sinks and let $D^{tt'} > 2R^t F^t + 2R^{t'} F^{t'} + \Delta^{tt'}$ be an upper bound on the width occupied by two pipelines/cable trays, one of them of type t and the other one of type t' .

Table 4.1 summarizes the main elements and variables used in our model. Once these elements are set up, we can formulate the PCTLTP as the following **Mixed Integer Linear Programming (MILP)**:

Parameters	
V	set of virtual nodes.
E	set of total edges.
E^v	set of virtual edges.
$RE(e)$	$\{\tilde{e} \in E^v : \tilde{e} \text{ has the same coordinates as } e \in E^v\}$.
A	arc set induced by E .
$\mathcal{T}$	set of possibles service types.
$\{s_1^t, \dots, s_{p_t}^t\}$	set of sources for services of type $t \in \mathcal{T}$
$\{d_1^t, \dots, d_{q_t}^t\}$	set of sinks for services of type $t \in \mathcal{T}$
$V' \subset V$	subset of vertices which are neither sources nor sinks.
$g_{s_i^t}$	number of services of type t sent from source s_i^t (out-flow at s_i^t).
$g_{d_j^t}$	number of services of type t sent to sink d_j^t (in-flow at d_j^t).
c_e^t	cost of using edge e by a service of type t .
R^t	width of the pipeline/cable of a single service of type t .
$D^{tt'}$	upper bound on the width occupied by two pipelines of type t and t' .
D^t	minimum allowed distance between consecutive elbows on a pipeline of type t .
Δ^t	minimum allowed distance between services of type t and services of any other type.
$\Delta^{tt'}$	minimum allowed distance between services of type t and t' .
$d_{ee'}$	minimum Euclidean distance between the edges $e, e' \in E$
F^t	maximum allowed number of services of type t jointly routed in the same cable tray.
Variables	
x_e^t	$\begin{cases} 1 & \text{if edge } e \text{ is used by the route of services of type } t, \\ 0 & \text{otherwise.} \end{cases}$
f_{ij}^t	amount of flow of pipeline of type t routed through arc (i, j) .

Table 3.1: Parameters and variables used for formulating PCTLTP.

$$\min \sum_{e \in E} \sum_{t \in \mathcal{T}} c_e^t x_e^t \quad (3.1)$$

$$\text{s.t.} \quad \sum_{\substack{j \in V': \\ (i,j) \in A}} f_{ij}^t - \sum_{\substack{j \in V': \\ (i,j) \in A}} f_{ji}^t = 0, \quad \forall t \in \mathcal{T}, \forall i \in V', \quad (3.2)$$

$$\sum_{\substack{j \in V: \\ (s_i^t, j) \in A}} f_{s_i^t j}^t - \sum_{\substack{j \in V: \\ (j, s_i^t) \in A}} f_{j s_i^t}^t = g_{s_i^t}, \quad \forall t \in \mathcal{T}, i = 1, \dots, p_t, \quad (3.3)$$

$$\sum_{\substack{i \in V: \\ (i, d_j^t) \in A}} f_{i d_j^t}^t - \sum_{\substack{i \in V: \\ (d_j^t, i) \in A}} f_{d_j^t i}^t = g_{d_j^t}, \quad \forall t \in \mathcal{T}, j = \dots, q_t, \quad (3.4)$$

$$(f_{ij}^t + f_{ji}^t)R^t + (f_{i'j'}^{t'} + f_{j'i'}^{t'})R^{t'} + \Delta^{tt'} \leq d_{ee'} + D^{tt'}(2 - x_e^t - x_{e'}^{t'}), \quad \forall e = \{i, j\}, e' = \{i', j'\} \in E, \forall t, t' \in \mathcal{T}, \quad (3.5)$$

$$\sum_{\tilde{e} \in RE(e)} x_{\tilde{e}}^t + \sum_{\tilde{e} \in RE(e')} x_{\tilde{e}}^{t'} \leq 1, \quad \forall e, e' \in E^v : 0 < d_{ee'} \leq D^t, e \neq e', \forall t \in \mathcal{T}, \quad (3.6)$$

$$x_e^t \leq f_{ij}^t + f_{ji}^t \leq F^t x_e^t, \quad \forall t \in \mathcal{T}, \forall e = \{i, j\} \in E, \quad (3.7)$$

$$x_e^t \in \{0, 1\}, \quad \forall e \in E, \forall t \in \mathcal{T}. \quad (3.8)$$

Objective function (2) minimizes the overall cost of the routes. Note that this objective function benefits the creation of cable trays since the cost of using an edge

does not depend on the number of services using this edge (edge flow) but only if it is used or not. Constraints (3), (4), and (5) are the flow conservation constraints ensuring the adequate construction of paths for the services. Constraints (8) ensure that in a feasible solution to the problem, the minimum allowed distances between services of a different type are kept considering the pipelines/cable tray width, which depends on the flow. The expression $R^t(f_{ij}^t + f_{ji}^t) + R^{t'}(f_{i'j'}^{t'} + f_{j'i'}^{t'}) + \Delta^{tt'}$ for a pair of edges $e = \{i, j\}$ and $e' = \{i', j'\}$ and types $t, t' \in \mathcal{T}$, models the width of the space occupied by the two pipelines/cable trays (plus the safety distance $\Delta^{tt'}$). If both e and e' are used to route services of type t and t' , respectively, this amount cannot exceed the minimum Euclidean distance between the edges, since otherwise, the edges are not far enough to fit the two pipelines/cable trays.

Constraints (11) assure that in case two elbows of the same pipelines/cable trays are closer than the minimum allowed distance (D^t), only one of them can be activated in the solution (*elbow test*). Clearly, given two edges $e, e' \in E^v$, if $d_{ee'} \leq D^t$, all the edges in $RE(e)$ and $RE(e')$ are incompatible and then cannot take simultaneously a value of 1. Finally, constraints (3.7) (together with constraints (8)) impose that, if edge $e = \{i, j\}$ is used by the route of a service of a given type $t \in \mathcal{T}$ then, only services of type $t \in \mathcal{T}$ can be routed through the arcs (i, j) and (j, i) . They also force to activate edge $e = \{i, j\}$ for routing services of type t if there exists a service of type t routed through the edge and limit the number of services of type t routed through the edge to be at most F^t .

The solution of the PCTLP is a set of paths connecting sources with destinations where services of different types do not overlap on the graph G (nor in the 3D representation of the pipelines/cable trays). Ideally, in this solution, several services of the same type should join and go together through various edges to eventually separate and finish on their sinks.

3.3.1 An exact relax-and-cut procedure for solving PCTLP

Note that the number of constraints in families (8) and (11) is too large to make it possible to include them all into a MILP solver for medium/large-sized instances. To overcome this issue, we start by solving a relaxed formulation without these constraints and, to avoid the infeasibility of the solutions, we proceed by incorporating them as needed in a relax-and-cut procedure. More precisely, the solution of the relaxed problem may not meet the requirements of constraints (8) and/or (11). To make sure each solution is valid, we check for feasibility by using an enumerative procedure. If a constraint is found to be violated, it is separated and added to the constraint pool. Then, we solve again the relaxed formulation of the problem obtained after adding the constraints in the pool. The procedure terminates when a solution is found verifying all the constraints.

To check if constraints (8) are met, we measure the minimum distance between edges belonging to different cable trays. If this distance is smaller than the sum of the width of the involved cable trays and the required security distance, we consider the constraint to be violated and add it to the constraint pool.

To verify the validity of constraints (11), we first arrange the virtual edges in the path of a pipeline/cable tray and then measure the minimum distance between

consecutive virtual edges. If this distance is smaller than or equal to the minimum required distance between elbows for the pipelines/cable tray type, the constraint is considered violated and it is introduced as a feasibility cut to the set of constraints pool.

Note that the size of the *activated* edges (by x) and flows (f) in a relaxed solution of the problem for a given type t is much smaller than the number of edges and arcs in the original graph, and thus, the separation process can be efficiently solved in practice. Furthermore, we embed this procedure into a branch-and-cut scheme using callbacks, which are available in most of the commercial off-the-shelf software, such as Gurobi, CPLEX, or FICO.

3.4 A general heuristic approach

Although the relax-and-cut procedure mentioned in the previous section alleviates the problem-solving caused by the difficulty of the technical constraints, the number of variables and constraints of the model is still huge even for medium-sized instances. In this section, we develop a family of heuristic approaches designed to provide feasible solutions in lower computation times than the exact relax-and-cut procedure.

The different heuristic approaches that we propose are based on the following two main phases:

Phase I: Construction of initial paths and cable trays. In this first step, the goal is to construct promising paths in the graph not necessarily feasible connecting the sources and sinks of the services of each type and encouraging services of the same type to share edges using the overall less costly edges.

Phase II: Transformation to feasible paths and cable trays. Given a partial solution obtained in Phase I, the aim of this stage is to check for feasibility (and to provide feasible solutions if necessary) of the PCTLTP by correcting the technical requirements, i.e., the minimum distance between services and the elbow test.

In what follows, we describe the different strategies that we propose to deal with the different phases above.

3.4.1 Strategies for Phase I

For this phase, we propose two different approaches to enforcing or encouraging the construction of cable trays.

In the first strategy, the so-called *Constructive Algorithm* (H1) locates certain points (called *supernodes*) that will define the beginning- and end-nodes of the pipelines/cable trays. Then, both supernodes of each cable tray are connected by a path, constructed by an adapted shortest path algorithm. Finally, connections (not necessarily verifying the technical requirements) between sources, sinks, and trays are established.

The second strategy, the so-called *Decomposition Algorithm* (H2), is based on solving separately for each type of service the **Minimum Cost Multicommodity Network Flow Problems** (MCMNFP) which is solvable in polynomial time by a primal network simplex algorithm [see, e.g., [Orlin, 1997](#)]. The different types of services are independently routed, and the construction of cable trays is encouraged by increasing the costs of certain edges in the underlying graph.

The two approaches are independently performed for each type $t \in \mathcal{T}$ and are detailed as follows:

Constructive Algorithm (H1).

For each type $t \in \mathcal{T}$:

- **H1.1 (Supernodes location).** In order to find the supernodes (among the set of sinks and sources) which are the end-nodes of the q cable trays that are constructed, we solve a mathematical programming model for each service type based on a p -median location problem. We use the following set of binary variables:

$$y_j^l = \begin{cases} 1 & \text{if node } j \text{ is selected as the } l\text{-th supernode,} \\ 0 & \text{otherwise,} \end{cases}$$

where $l \in L = \{1, \dots, 2q\}$, $q = \lceil \phi^t / F^t \rceil$ and ϕ^t is the total flow that type t services transport through them (number of services of type t). Also, we consider the following set of binary variables:

$$z_{ij}^l = \begin{cases} 1 & \text{if source/sink } i \text{ is assigned to the } l\text{-th supernode located at node } j, \\ 0 & \text{otherwise.} \end{cases}$$

Parameters	
h_i	flow of the source/sink i .
d_{ij}	minimum distance between edges i and j .
ϕ^t	number of services of type t .
F^t	maximum number of services of type $t \in \mathcal{T}$ that can be jointly routed in the same cable tray.
q	$\lceil \phi^t / F^t \rceil$.
L	$\{1, \dots, 2q\}$.
V	vertices in the graph.
ST	total set of sources and sinks for service type t .
Variables	
z_{ij}^l	$\begin{cases} 1 & \text{if source/sink } i \text{ is assigned to the } l\text{-th supernode located at node } j, \\ 0 & \text{otherwise.} \end{cases}$
y_j^l	$\begin{cases} 1 & \text{if node } j \text{ is selected as the } l\text{-th supernode,} \\ 0 & \text{otherwise.} \end{cases}$

Table 3.2: Parameters and variables of formulation (3.9)-(3.18).

Let $ST = \{s_1^t, \dots, s_{p_t}^t\} \cup \{d_1^t, \dots, d_{q_t}^t\}$ be the total set of sources and sinks for service type t . The model is formulated as:

$$\min \sum_{l \in L} \sum_{j \in V} \sum_{i \in ST} |h_i| d_{ij} z_{ij}^l \quad (3.9)$$

$$\text{s.t. } \sum_{l \in L} \sum_{j \in V} z_{ij}^l = 1, \quad \forall i \in ST, \quad (3.10)$$

$$z_{ij}^l \leq y_j^l, \quad \forall l \in L, \forall i \in ST, \forall j \in V, \quad (3.11)$$

$$\sum_{l \in L} y_j^l \leq 1, \quad \forall j \in V, \quad (3.12)$$

$$\sum_{j \in V} y_j^l = 1, \quad \forall l \in L, \quad (3.13)$$

$$\sum_{l \in L} \sum_{j \in V} y_j^l = 2q, \quad (3.14)$$

$$\sum_{l \in L} \sum_{i \in ST} h_i z_{ij}^l \leq F^t y_j^l, \quad \forall j \in V, \quad (3.15)$$

$$\sum_{i \in ST} \sum_{j \in V} h_i z_{ij}^l = - \sum_{i \in ST} \sum_{j \in V} h_i z_{ij}^{l+1}, \quad \forall l \in L : l \text{ is odd}, \quad (3.16)$$

$$y_j^l \in \{0, 1\}, \quad \forall l \in L, \forall j \in V, \quad (3.17)$$

$$z_{ij}^l \in \{0, 1\}, \quad \forall l \in L, \forall i \in ST, \forall j \in V. \quad (3.18)$$

We show in Table 3.2 a summary of the parameters and variables used in our formulation. The objective function (3.9) minimizes the flow per path length between sources/sinks and supernodes, where h_i is the flow of the source/sink i and d_{ij} is the shortest path distance from source/sink i to supernode j . Constraints (3.10) force each source/sink to be assigned to one supernode. Constraints (3.11) ensure sources/sinks can only be assigned to an activated supernode. Constraints (3.12) guarantee there cannot be more than one supernode with the same label. Constraints (3.13) assign each label $l \in L$ to a single supernode. Constraint (3.14) ensures there is a total of $2q$ supernodes.

Constraints (3.15) guarantee that the total flow that reaches a supernode must be less than its capacity F^t . Constraints (3.16) match supernodes so they have the same flow of opposite sign. If F^t is large enough, then only one tray of type t is formed. In this case, $L = \{1\}$, $q = 1$ and constraints (3.11), (3.12), (3.14), (3.15) are ignored.

- **H1.2 (Supernodes connection).** To connect supernodes of each service type through feasible cable trays we use an adapted shortest path-based method similar to the one proposed in Blanco et al. [2022]. This approach ensures that the pipelines/cable trays verify all the technical requirement imposed in our problem (elbow-test and minimum distance between different types).
- **H1.3 (Linking sources/sinks to trays).** Every source/sink is connected to its corresponding tray using the shortest path, which is also considered as part of the cable tray and can be used to connect other sources/sinks to it. This process is repeated until all the sources/sinks are connected to a tray. We denote by E_t^{best} the list of edges obtained at the end of the procedure. Note that these connections may not verify the technical requirement imposed in the PCTLP.

Decomposition-based algorithm (H2).

For each type $t \in \mathcal{T}$:

- **H2.1.** A MCMNFP is solved only for type t . We denote by E_t the list of edges induced by the solution (arcs with a positive flow). Observe that since the remainder types are not considered in the problem, the technical constraints are not guaranteed.
- **H2.2.** For each connected component of E_t , we increase the cost of all the edges that do not belong to the selected connected component and we solve again the MCMNFP for this type of service t . This boosts all the services to use the same set of edges (those in the selected connected component). Once it is repeated for each connected component, the best solution (based on the objective value of the PCTLP), E_t^{best} , is kept. This strategy allows to concentrate the flow in the overall less costly set of edges.

3.4.2 Strategies for Phase II

For this phase, we propose two different iterative approaches in order to construct feasible PCTLP solutions, verifying all the technical requirements. Both approaches use as starting solution the one obtained in Phase I, i.e., E_t^{best} . The first algorithm, the so-called *Graph trimming*, is based on solving the PCTLP on a trimmed graph of the original one, and so with a smaller number of variables and constraints. The second approach, the so-called *Increasing Cost*, consists of increasing the costs of the edges not verifying the technical constraints to avoid their use, and repeating the process until they are fulfilled.

Graph Trimming (GT). Once the initial heuristic solution is obtained from any of the strategies described in Phase I, constraints (8) and (11) are evaluated. If they are verified, the initial heuristic solution is feasible for the PCTLTP. Otherwise, we propose an iterative procedure that begins with the initial heuristic solution. Then, for each service type, a cylindrical region of a given initial width is considered around each edge of the union of edges that take part of the initial solution. Instead of solving the entire PCTLTP in the graph, we solve it in a reduced space, the union of these regions by fixing the variables indicating arcs outside of these regions to zero. If the problem is feasible, it provides a feasible solution, otherwise, the width of the regions is increased and the process is repeated until feasibility is achieved. Note that as the number of iterations increases the dimension of the trimmed graph becomes larger, and then, the problem requires more computational time to be solved. Although in the worst-case scenario, the algorithm requires solving the original instance of the problem (the whole graph), in practice, this strategy significantly reduces the number of variables needed to solve the problem.

Increasing Cost (IC). This strategy consists of guiding the solutions obtained by the MCMNFP to verify all the requirements in the PCTLTP by increasing the cost of the non-desired edges in the design (to the maximum cost value). The algorithm considers a maximum number of iterations to be performed, $MaxIt$, and for each iteration $it \in \{1, \dots, MaxIt\}$ a sorted list of all the commodities (types), $T_{\text{sort}}(it)$. A multistart approach is then applied by testing different sorted lists and choosing the best of them. Some of the sorted lists are based on the decision maker's preferences and others are obtained randomly. This approach, for iteration $it \in \{1, \dots, MaxIt\}$ and for type $t \in T_{\text{sort}}(it)$, consists of the following iterative procedures:

- **Elbow-Test.** The elbow constraints (11) are checked for the obtained solution. If any of them is violated one of the two elbows not verifying the test is chosen and its cost is increased to avoid its use in later elbow-test iterations. The MCMNFP is again solved with the new costs and E_t^{best} is updated accordingly. This step is repeated until the elbow test is verified or a maximum number of elbow-test iterations is reached (in whose case the solution in iteration it is not valid).
- **Minimum Distance.** The distance requirements (8) are tested by checking overlapping edges with respect to each of the previously processed types $t' \in \{1, \dots, t-1\}$, if any. If edge e used in type t with flow f_e^t overlaps edge e' used in any type $t' \in \{1, \dots, t-1\}$ with flow $f_{e'}^{t'}$, this stage is addressed by increasing, for type t , the cost of the edges in conflict, that is, the edges at a distance lower than or equal to $f_e^t R^t + f_{e'}^{t'} R^{t'} + \Delta^{tt'}$ from the edges used in $E_{t'}^{\text{best}}$ and then, go to Phase I to construct a new list of edges for type t , E_t^{best} (updated). This procedure is repeated until there are no conflicts or a maximum number of overlapping iterations ($MaxOverIt$) is reached (in whose case the solution, in iteration it , is not valid). Once no conflicts appear, to prevent overlapping for the next

types in the sorted list, the cost for types $t + 1, \dots, T$ is increased for the edges in E_t^{best} .

Once all iterations are solved we keep the solution of the best iteration (based on the objective value of the PCTLP).

Note that the strategies described above for the two different phases of the heuristic can be combined in different ways. In Figure 3.2, we illustrate a possible choice in which H2 is applied for Phase I and Increasing Cost for Phase II. Note that since in Phase II, the types of services are sorted, the sequence in which the types are processed in Phase I follows the same order.

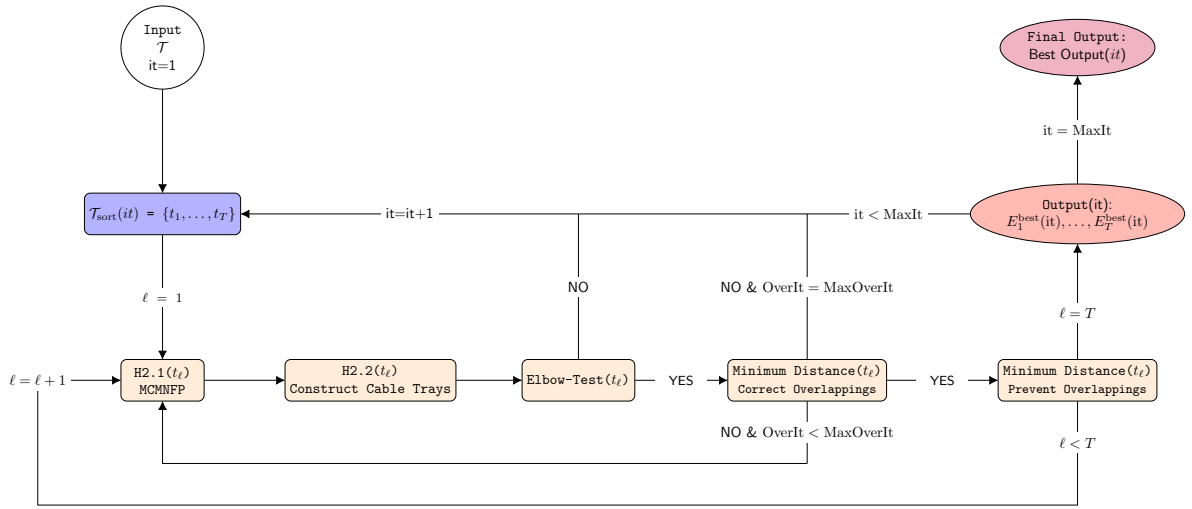


Figure 3.2: Flowchart of the heuristic combination of H2 for Phase I and Increasing Cost for Phase II.

3.5 Computational experience

In this section, we report the results of our computational study aimed at evaluating the effectiveness of our proposed methods for solving the PCTLP. Our computational experience is organized into three parts. First, we examine the computational performance of our algorithms on a battery of randomly generated instances. Second, we assess the performance of the algorithms on instances that we have specifically designed for the PCTLP, the so-called “corridor instances”. These instances are publicly available at <https://github.com/anticlclon/Dataset-Trays-Location-Problem>. Finally, we apply our methodology to a real-world case study provided by our industrial partner.

We use Gurobi v7.7 optimizer to solve all instances on a Windows 10 platform, powered by an Intel(R) Core(TM)i7 CPU 2.93 GHz processor and 16 GB RAM. We use the default settings for all Gurobi solver parameters and set a CPU time limit of 4 hours.

3.5.1 Random instances

In our first study, we consider a set of random instances, that have been already used in the literature, to validate our proposal. Each instance consists of a cube of length 128 units containing a number of obstacles and different services that must be routed between a set of sources and sinks (see Table 3.3). More specifically, for each instance, an orthogonal 3D grid is built on such a cube with a density $d \in \{17, 33\}$, with fixed distances between nodes, i.e., each axis is divided into 16 or 32 equally sized parts, and the grid is constructed. Five groups of $o \in \{5, 10, 15\}$ 10-units cubic obstacles are randomly generated. Five groups of $s \in \{5, 8, 12\}$ services are also generated, each of them classified into one of two types of service (A or B), such that type A will have services enumerated with an odd number whereas B has those services enumerated with an even number. The generated grid is constructed such that it has enough capacity to gather all services of the same type. In scenarios with 5 services, the width of a service in both types is $R^t = 0.33$. Analogously, in scenarios with 8 or 12 services, widths are $R^t = 0.25$ and $R^t = 0.17$, respectively.

We adopt the same cost function structure as introduced in Section 2.5.1, where edge costs take into account physical distance, the presence of elbows, and vertical transitions. For completeness, the cost of an edge ee of type tt is defined as:

$$c_e^t = \alpha_1^t(d_e + 10El_e + 2Ch_e),$$

where the meaning of each term and coefficient is detailed in that earlier section.

In total, 90 benchmark instances are generated based on the combination of distances, services, and obstacles. Table 3.3 summarizes the types of instances used in this study.

Density (d)	#Services (s)/Radii (R)	Obstacles (o)
17, 33	5/0.33	5, 10, 15
	8/0.25	
	12/0.17	

Table 3.3: Summary of the types of random instances.

The size of each instance can be described by the number of edges in the resulting graph. For a given density d , this graph has $3(d-1)d^2$ physical and $3d^3$ virtual edges.

Finally, we recall that for each case, the obstacles generated within the grid remove some edges. Since there is one binary variable and two flow variables for each edge and each type, the final average number of edges can also be obtained by dividing column **#Vars** by six.

The proposed general heuristic approach described in Section 3.4 is very flexible and can be parameterized in many different ways. Among all possible combinations of strategies for Phases I and II, we have run two of them and, additionally, a mixing strategy to compare their performance against the exact relax-and-cut (EX) namely:

H2+IC: We test the decomposition-based algorithm (H2) improving the results with the increasing cost method in order to guarantee the elbow-test and minimum distances between services.

H12+GT: We trim the graph around the solutions provided by H1 and H2 and solve the PCTLTP in the trimmed graph with a time limit of 900 seconds. This time limit is extended if required until a feasible solution is found.

GT+IS: In order to improve previous solutions, we trim the graph again 32 units around the H2+IC and H12+GT solutions and solve the PCTLTP with a time limit of $14400 - t_1 - t_2$ seconds, where t_1 and t_2 stand for the time consumed by H2+IC and H12+GT, respectively. Here we adopt as initial solution (IS) the best result from both H2+IC and H12+GT methods.

A time limit of 4 hours (14400 seconds) was fixed for solving each instance with the exact approach (EX).

In Table 3.4, we report the average results of this first battery of computational experiments. The first three columns indicate the parameters identifying the instances, d (density), s (number of services), and o (number of obstacles). For each of the combinations of these parameters five instances were generated and we report the average results of the five instances. Column **#Vars** indicates the number of variables of the (MCMNFP) problem and column **#Ctrs** the number of constraints. Column **MIPGAP** indicates the relative MIP gap obtained at the end of the time limit for the exact method (EX). In the block of columns denoted by **Dev_UB**, we report, for each one of the procedures, the relative percent deviation of the obtained solution with that procedure (upper bound), z_{ub} , with respect the best solution among the two approaches, exact and GT+IS, z_{best} , that is:

$$\text{Dev_UB} = \frac{z_{ub} - z_{best}}{z_{best}} \times 100.$$

Finally, in the block of columns **CPU Time**, we report the CPU time, in seconds, required by each one of the approaches. The flag **TL** indicates that the time limit was reached in all the instances averaged in that row.

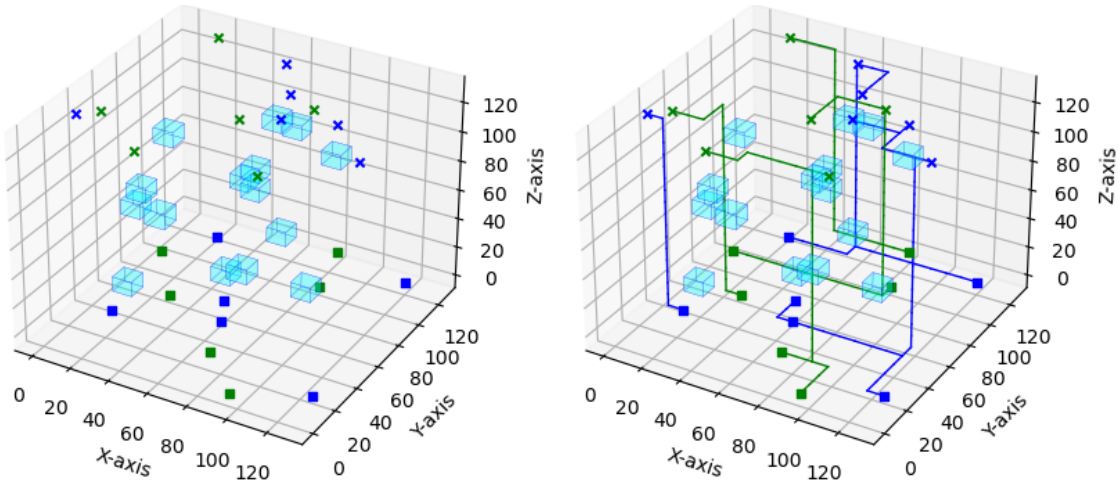
A first analysis of the results shows that the exact approach was not able to solve, up to optimality, any of the instances within the time limit, being the **MIPGAP** significantly large. In addition, the average number of explored nodes in the branch-and-bound tree was below 10 for $d = 17$ and equal to 1 for $d = 33$ which implies that the **MIPGAP** is close to the gap with respect to the linear relaxation for those instances. This behavior was expected given the already known big size of the formulations in terms of variables and constraints. Regarding columns **Dev_UB** for EX and GT+IS, we can compare the results obtained by the exact method with the results obtained by matheuristic GT+IS in the same computational time (14400 seconds). We observe that the best solution (deviation from the best solution = 0.0) is obtained by GT+IS approach in most of the tested instances, especially for $d = 33$. Therefore, we can assert that matheuristic GT+IS outperforms the exact method.

On the other hand, the results on the deviations obtained by the heuristic approaches H2+IC and H12+GT show that H2+IC obtains better solutions in less computational time, being in many instances better than the solution obtained by the exact procedure, especially for $d = 33$. H12+GT also provides good results in small running times, which also benefits significantly the results obtained with GT+IS.

d	s	o	#Vars	#Ctrs	MIPGAP	Dev_UB				CPU Time			
					EX	EX	H2+IC	H12+GT	GT+IS	EX	H2+IC	H12+GT	GT+IS
17	5	5	171174	143535	25.0	0.2	1.5	3.7	0.0	TL	42.6	949.1	TL
		10	170565	143055	27.3	0.0	1.6	7.2	0.0	TL	47.7	947.6	TL
		15	170052	142650	26.8	0.0	1.6	10.2	0.4	TL	42.8	946.8	TL
	8	5	171174	143535	26.7	0.7	6.5	15.2	0.9	TL	65.5	970.3	TL
		10	170565	143055	27.0	2.6	5.8	12.9	0.3	TL	64.0	970.0	TL
		15	170052	142650	27.0	0.7	5.3	15.1	1.5	TL	63.5	971.4	TL
	12	5	171174	143535	27.1	1.7	6.9	15.8	0.0	TL	94.2	930.2	TL
		10	170565	143055	26.9	2.7	7.0	11.7	0.2	TL	105.3	1018.3	TL
		15	170052	142650	27.0	4.0	7.7	10.0	0.0	TL	92.7	1013.9	TL
	17			170597	143080	26.8	1.4	4.9	11.3	0.4	TL	68.7	968.6
33	5	5	1270624	1062214	37.7	6.4	0.5	15.2	0.0	TL	1956.0	1617.0	TL
		10	1266961	1059258	36.9	4.9	0.4	10.4	0.0	TL	1068.6	1614.9	TL
		15	1263660	1056593	36.9	4.9	0.7	12.6	0.0	TL	3030.2	1618.6	TL
	8	5	1270624	1062214	36.9	14.2	2.2	18.0	0.0	TL	1312.4	1947.2	TL
		10	1266961	1059258	36.7	13.4	2.2	20.7	0.0	TL	1073.6	1963.1	TL
		15	1263660	1056593	36.6	11.5	1.0	17.7	0.0	TL	1206.4	2041.3	TL
	12	5	1270624	1062214	36.6	8.8	2.1	24.2	0.0	TL	1620.8	2636.0	TL
		10	1266961	1059258	36.7	10.6	3.7	22.4	0.0	TL	1565.2	2556.4	TL
		15	1263660	1056593	36.6	7.0	3.0	20.6	0.4	TL	1864.5	2748.0	TL
	33			1267082	1059355	36.9	9.1	1.8	18.0	0.0	TL	1633.1	2082.5
Total			718839	601217	31.8	5.2	3.3	14.6	0.2	TL	850.9	1525.6	TL

Table 3.4: Computational results for random instances.

Figure 3.3 shows an example of a random instance $(d, s, o) = (33, 12, 15)$ together with the best obtained solution obtained with the math-heuristic GT+IS. There, the squares ($\square$) represent the sources, the crosses ($\times$) the sinks, the obstacles are the light blue cubes, and the services are highlighted in blue (type A) and in green (type B). As can be observed, the solution shows that there are seven trays that combine four green and three blue services, simplifying the design of twelve one-by-one single pipelines.

Figure 3.3: Graphical display of a random instance $(d, s, o) = (33, 12, 15)$ and the best provided solution.

3.5.2 Corridor instances

In our second battery of experiments, we generate new instances encouraging the formation of cable trays in real situations. The set of random instances used in Section 3.5.1 are cubes containing random obstacles that may not favor the trays' formation. In order to force services to be grouped in trays, we have designed a specific set of instances with elongated shapes and a specific obstacles layout. This collection of corridor instances is generated as follows.

The domain of this new set of instances is the 3D box $[0, 128] \times [0, 128] \times [0, 1024]$. An orthogonal 3D grid of $d_1 \times d_2 \times d_3$ physical nodes, with $(d_1, d_2, d_3) \in \{(9, 9, 65), (17, 17, 129)\}$ is built on this box, being the distance between adjacent nodes fixed to $\frac{128}{d_1-1}$ in X -axis, $\frac{128}{d_2-1}$ in Y -axis and $\frac{1024}{d_3-1}$ in Z -axis. This means a spacing of 16 and 8 units for $d = (d_1, d_2, d_3) = (9, 9, 65)$ and $d = (d_1, d_2, d_3) = (17, 17, 129)$, respectively. Here, d represents, again, the *density* of the grid. The graph for a corridor instance with density $d = (d_1, d_2, d_3)$ has $d_1 d_2 (d_3 - 1) + d_1 (d_2 - 1) d_3 + (d_1 - 1) d_2 d_3$ physical edges and $3d_1 d_2 d_3$ virtual edges.

We consider a total of thirteen services partitioned into two types of services. There are eight services of type A that have to be sent from four sources, each one with an outgoing flow of two to four sinks, two of them with an in-flow of three, and another two with an in-flow of one. The remaining five services are of type B and depart from two sources with an outgoing flow of two and three, respectively, to five sinks, each one with an in-flow of one. Two different scenarios, S_1 and S_2 , are generated with these characteristics by randomly distributing sources and sinks of the two types of services on the walls of the constructed 3D grid.

Two different groups of obstacles are also generated. The first group has two obstacles and the second one has four obstacles. Each obstacle is a 3D box that is given by their opposite vertices. In Table 3.5, we specify the vertices of obstacles as well as the different parameters that affect the generation of this new set of instances. In total, eight (2 densities $\times 2$ scenarios $\times 2$ groups of obstacles) corridor instances have been generated. Finally, we consider that each tray has enough capacity to gather all services of the same type and an established radius of $R^t = 2$ for all $e \in E$, $t \in \mathcal{T}$.

We consider the same cost structure as in the random instances in the previous section.

Density	Service Types	Obstacles	Radii
(9,9,65)	Type A (8 services): 4 sources: (out-flows: 2, 2, 2, 2) 4 sinks: (in-flows: 3, 3, 1, 1)	2 obstacles: $O_{11} = \{(16; 16; 288); (48; 128; 736)\}$ $O_{12} = \{(96; 16; 288); (128; 128; 736)\}$	2
(17,17,129)	Type B (5 services): 2 sources: (out-flows: 2, 3) 5 sinks: (int-flows: 1, 1, 1, 1, 1)	4 obstacles: $O_{21} = \{(0; 0; 288); (48; 48; 736)\}$ $O_{22} = \{(80; 0; 288); (128; 48; 736)\}$ $O_{23} = \{(0; 80; 288); (48; 128; 736)\}$ $O_{24} = \{(80; 80; 288); (128; 128; 736)\}$	

Table 3.5: Summary of the corridor instances.

Figure 3.4 shows two examples of these types of instances and the type of solution obtained. In these two examples, the squares ($\square$) are the sources and the crosses ($\times$) the sinks, in color **blue** for service type A and in **green** for service type B. In

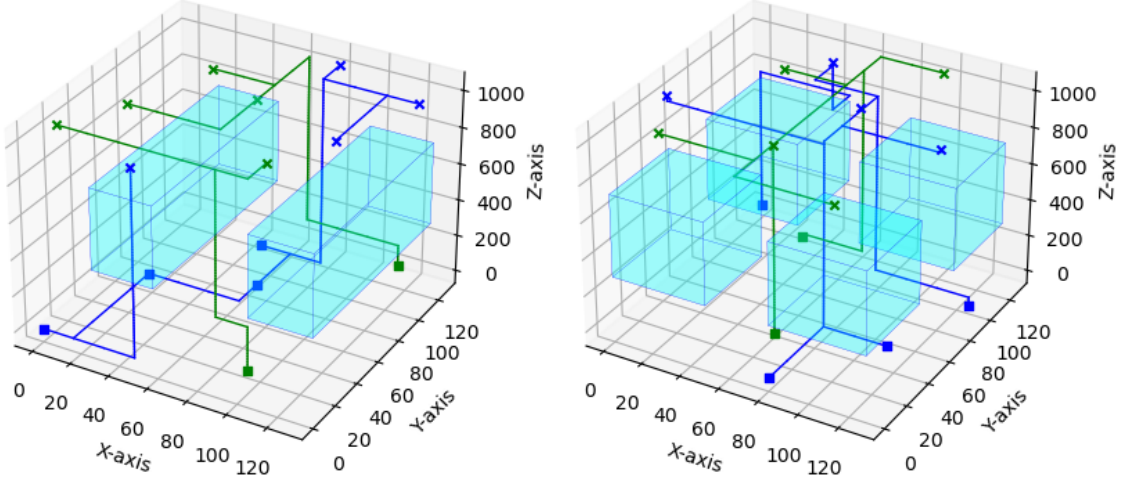


Figure 3.4: Graphical display of the solution for two corridor instances.

the left figure, two obstacles are considered and the sources and sinks are placed under scenario S_1 . In the right figure, the instance consists of four obstacles, and the positions of the sinks and sources are generated under scenario S_2 . The density in both figures is $d = (d_1, d_2, d_3) = (9, 9, 65)$. The reader should observe that Figure 3.4 shows how these corridor instances favor the aggregation of services into trays routed by the feasible corridors defined by the obstacles.

In Table 3.6, we report the results obtained for these instances, with the same structure that Table 3.4 where column S indicates the scenario type. We observe that, again, the exact approach did not solve to optimality any of the instances, and that the MIP gaps at the end of the time limit are large. If we compare the exact method EX with GT+IS approach, we observe that the best solution is obtained by matheuristic GT+IS in all the instances. In addition, regarding columns MIPGAP and Dev_UB for EX and $d = (17, 17, 129)$, we can assert that this solution is near to the optimal solution.

On the other hand, again the results on the deviations obtained by the heuristic approach H2+IC show that this approach is able to obtain a very good solution, which is much better than the solution obtained by the exact procedure for $d = (17, 17, 129)$, in considerably less computational time.

d	o	S	#Vars	#Ctrs	MIPGAP	Dev_UB				CPU Time			
					EX	EX	H2+IC	H12+GT	GT+IS	EX	H2+IC	H12+GT	GT+IS
(9,9,65)	2	S_1	130242	110066	16.6	0.5	4.4	82.0	0.0	TL	52.7	962.4	TL
		S_2	130242	110066	24.2	1.9	6.4	49.7	0.0	TL	72.9	970.4	TL
	4	S_1	114834	97010	16.2	0.0	4.8	63.4	0.0	TL	2077.9	1997.4	TL
		S_2	114834	97010	13.5	1.0	4.1	50.9	0.0	TL	1087.9	2155.3	TL
(17,17,129)	2	S_1	1000236	839210	60.2	78.4	3.3	78.4	0.0	TL	873.9	4138.4	TL
		S_2	1000236	839210	56.1	52.7	4.5	140.8	0.0	TL	950.0	2144.8	TL
	4	S_1	910698	763786	56.2	64.2	4.7	67.8	0.0	TL	655.5	1792.3	TL
		S_2	910698	763786	64.4	85.6	5.4	122.2	0.0	TL	655.5	1792.3	TL
Total			910698	763786	38.4	35.5	4.7	81.9	0.0	TL	655.5	1792.3	TL

Table 3.6: Computational results for corridor instances.

3.5.3 Case study

This section presents an application of the proposed methodology to a real-world-based instance provided by our partner Ghenova, a leading Naval Engineering company. This is an instance with a number of services of different types and a series of obstacles and corridors that compromise the available space for the design (see Figure 3.5). The structure of the instance is as follows.

- The designated space for pipelines and cable trays system design is a three-dimensional rectangular prism representing a ship cabin. Its dimensions are 5732 units (X -axis), 2836 units (Y -axis), and 2013 units (Z -axis).
- A grid was created for the cabin by dividing each axis into segments of 100 units, resulting in an initial grid of $58 \times 29 \times 21$ nodes. Virtual nodes and edges were introduced to generate the graph $\tilde{G} = (V, E)$, consisting of 73,407 virtual nodes and 141,101 edges (virtual and physical).
- The cabin houses five types of services identified by numbers 0 to 4. The distribution of services includes three services of the first type, two services of the second and third types, one service of the fourth type, and six services of the fifth type. Each service type can be accommodated within a tray, and their sources and destinations are positioned throughout the cabin as per the designer's requirements.
- The routes for the trays face five obstacles (walls) depicted as light blue shapes in the plot. These obstacles represent metal slices with a width of 10 units and contain eight holes/windows represented by light orange squares, allowing passage.
- The designer specifies a minimum distance of 50 units between consecutive elbows, which is a typical requirement for this scenario.
- Each service type has assigned a different width: $R^1 = 50$, $R^2 = 25$, $R^3 = 75$, $R^4 = 150$, and $R^5 = 25$ units.

Once the grid is generated, we assign a cost to each of its edges, facilitating the evaluation of feasible routes within the network. The cost system incorporates the designer's preferences when routing the pipelines. It employs the following additive cost structure,

$$c_e^k = (\alpha_1^t + \alpha_5^t Pr_e) d_e + \alpha_2^t El_e + \alpha_3^k H_e + \alpha_4^t Ch_e + \alpha_6^t Pc_e + \alpha_7^t Cl_e,$$

considering the use of edges and elbows in each service's route. The cost calculation involves various parameters $(\alpha_1, \dots, \alpha_7)$ influenced by the edge characteristics. Detailed criteria and parameters defining the cost system are provided in Table 3.7. This flexible cost structure aligns with the preferences of naval designers and has been determined based on the selection criteria expressed by our partner company.

The cost associated with each edge in the graph $\tilde{G}$ is the same for all service types. In other words, $c_e^t = c_e^{t'}$ for any t and t' belonging to the set $\mathcal{T}$. The chosen parameters for this instance are:

Criteria	Description
d_e	physical distance between nodes i and j with $e = \{i, j\}$ (length of the edge).
E_e	1 if e represents an elbow and 0 otherwise.
$H_e = H - h_e$	being MH the maximum height of the ship cabin and h_e the height of e in case it is in a plane parallel to the XY -plane and 0 otherwise.
Ch_e	1 if e is a vertical edge (in the Z -axis) and 0 otherwise.
Pr_e	1 if e belongs to a preference zone and 0 otherwise.
Pc_e	1 if e crosses a penetrable zone and 0 otherwise.
Cl_e	1 if e represents an elbow and it is close to a source or a destination point and 0 otherwise.
Parameters	Description
α_1^k	Cost per unit length.
α_2^k	Cost of an elbow.
α_3^k	Cost of moving away from the ceiling.
α_4^k	Cost of changing in z-coordinates (height). Therefore, moving to a different height is penalized.
α_5^k	Bonus per routing the pipeline in a preference zone ($\alpha_5^k < 0$; $\alpha_1^k + \alpha_5^k > 0$).
α_6^k	Cost of crossing a penetrable zone.
α_7^k	Cost of locating an elbow close to the source or destination points of a pipeline.

Table 3.7: Criteria and parameters involving the cost function of the case study.

α_1	α_2	α_3	α_4	α_5	α_6	α_7
1	2800	700	200	-0.7	4000	3000

Furthermore, two preference zones are defined as parallelepipeds with the following dimensions: $[154241, 158844] \times [535, 1419] \times [24249, 25221]$ and $[152013, 154241] \times [535, 3371] \times [24249, 25221]$.

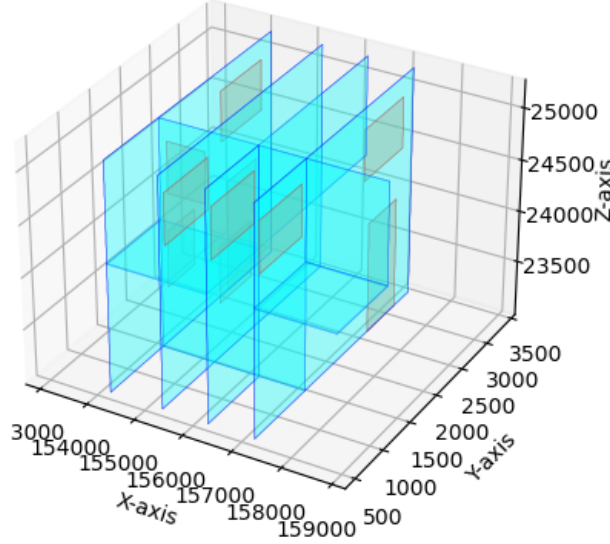


Figure 3.5: Scenario of the case study.

We replicate in this case study the solution methods applied in the corridor and random instances described in sections 3.5.1 and 3.5.2. The reader may observe that the exact approach (EX) was not able to return a feasible solution after 4 hours of computing time. This is shown in Table 3.8 with the flag **NF**. The best obtained objective function value is 323766.5, which is given by the GT+IS approach in four hours. We draw in Figure 3.6 the obtained solution in the graph (left) and the 3D representation of the obtained solution as actual pipelines and trays (right). Each type of service is highlighted with a different color to ease the identification of the different types of services. We would like to highlight that for all types of services, the model favors the integration of several services (of the same type) in single trays. Additionally, our approaches assure that the obtained solutions satisfy the requirements on the minimum distance between consecutive elbows, the minimum distance between different types of services, and that the design is routed throughout feasible space avoiding forbidden regions defined by obstacles.

Matheuristic GT+IS produces, in view of the result obtained, a good quality solution in reasonable CPU time, providing the naval designer with a powerful decision-aid tool for this task. The results on the deviations and CPU times for H2+IC and H12+GT approaches appear in Table 3.8. These two approaches are less time-consuming and they are also able to obtain a very good solution.

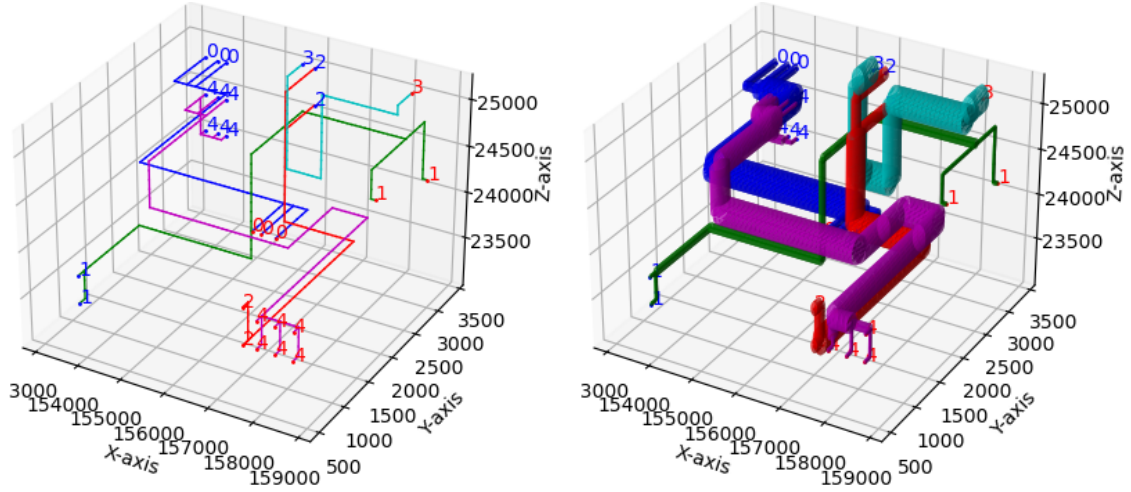


Figure 3.6: Graphical display of the solution for the case study with 14 services. In the left it is shown the solution as a collection of edges. The figure on the right shows the same solution in 3-dimensions.

Dev_UB				CPU Time			
EX	H2+IC	H12+GT	GT+IS	EX	H2+IC	H12+GT	GT+IS
NF	3.0	0.6	0.0	14400.0	175.0	1689.0	14400.0

Table 3.8: Computational results for the case study.

3.6 Conclusions

In this chapter, we provide different mathematical optimization-based tools to solve the Pipelines and Cable Trays Location Problem (PCTLTP), a challenging problem of determining the routes of different pipelines and cable trays in an intricate obstacle-ridden space of a ship. Apart from the constructability conditions required for the routes (such as the separation of services, or the distance between elbows), some services are desired to be jointly routed through the same *cable trays*. We propose an exact relax-and-cut approach for solving the problem based on a min-cost multicommodity flow formulation. The large dimension of the formulation for medium-sized instances makes it advisable to consider some heuristic algorithms that provide good-quality solutions in smaller computing times. Thus, we have developed two families of heuristic algorithms and tested all our methods (exact and heuristic) in two batteries of computational experiments, namely randomly generated and of corridor type. In all cases, the exact method is only able to prove optimality for small- to medium-sized instances. However, for large-sized instances, it fails to prove optimality and, in some cases, cannot find any feasible solution within the time limit. This limitation underscores the model's challenge in handling larger problem sizes. The heuristic approaches, on the other hand, provide satisfactory solutions within a reasonable CPU time. However, they still face difficulties in achieving good results for large-sized instances within an acceptable time frame. To enhance solutions for these instances, it would be beneficial to explore other heuristic approaches to

combine with our math-heuristic family to derive more effective solution algorithms. However, such an exploration exceeds the scope of this work. Furthermore, we conducted a real-world case study provided by our industrial partner to validate our proposal.

The methodology presented in the chapter is rather general and admits the incorporation of new elements required in the design as additional constraints in the formulation and heuristics, as well as the integration of the PCTLTP with other problems that appear in Naval Design. An important challenge is determining precise connection locations between pipelines or cables and the ship's power supplies, valves, and filters, following a predefined labeling scheme. The approval of a layout proposal depends on effectively addressing this connection problem and adhering to the specified scheme. Key considerations include imposing topological restrictions on branch connections and strategically placing valves within the branches. After proposing a layout for the pipes and cables, the methodology involves identifying specific points where branches must connect to ship devices. An optimal routing problem is then solved to determine the final branch layout. Integrating the pipeline and cable tray layout problem with the connection problem enables the discovery of new solutions that would not arise from sequential approaches. This integration has the potential to reduce overall costs. Furthermore, another possible extension is integrating the pipeline and cable tray problem with ship cabin layout design [Wang et al. \[2023\]](#). Although discrete optimization tools have been used to analyze these problems separately, a comprehensive joint study has not been conducted yet.

Chapter 4

Fixed Topology Minimum-Length Trees with Neighborhoods: A Steiner tree based approach

In this chapter, we introduce the Fixed Topology Minimum-Length Tree with Neighborhood Problem (FTTNP), which aims to embed a rooted tree-shaped graph into a d -dimensional metric space while minimizing its total length, subject to node placement within restricted regions. This problem has significant applications in efficiently routing cables or pipelines in engineering design. We propose novel mathematical optimization-based approaches to solve different variants of the problem based on the embedding domain. For continuous spaces, we present several Mixed Integer Nonlinear Programming formulations; for network-based embeddings, we develop a Mixed Integer Linear Programming formulation and a dimensionality reduction technique to solve larger instances efficiently. We also introduce a specialized extension for cable routing in industrial layouts, known as the Wiring Diagram Problem, which incorporates practical engineering constraints into the FTTNP framework. Finally, we report computational results that validate the effectiveness of our methodologies.

4.1 Introduction

Designing the routes for cables or pipelines is one of the most challenging phases of engineering projects, such as constructing ships, buildings, or bridges. This process requires designers to strategically trace cables through complex spaces while avoiding incompatible interactions, maintaining safety distances, and meeting intricate technical requirements. Typically, these designs are accompanied by a reference diagram that serves as a visual guide, illustrating connections between various elements linked by the cables. This *schematic planning* explicitly depicts the relationships between system components and provides guidelines for their placement. Following these connections ensures the proper functioning of the system, representing the topological structure that cables must adhere to in connecting elements such as valves and outlets. While these diagrams specify connectivity, they do not define the precise spatial positions of the elements, which must be placed within designated

areas. Consequently, designers face the challenge of selecting among infinitely many possible cable routes within the workspace.

In this chapter, we analyze the combinatorial optimization problem underlying such designs. Specifically, we introduce the *Fixed Topology Minimum-Length Tree with Neighborhood Problem* (FTTNP, for short). This problem involves a set of regions and a tree-shaped hierarchical structure that relates the elements to be placed. These elements correspond to the nodes of the tree, and the objective is to embed the tree into the designated space by placing the nodes within specified regions to minimize the total length of the design. In Figure 4.1 we show an instance of the problem (tree structure and regions) and an optimal solution.

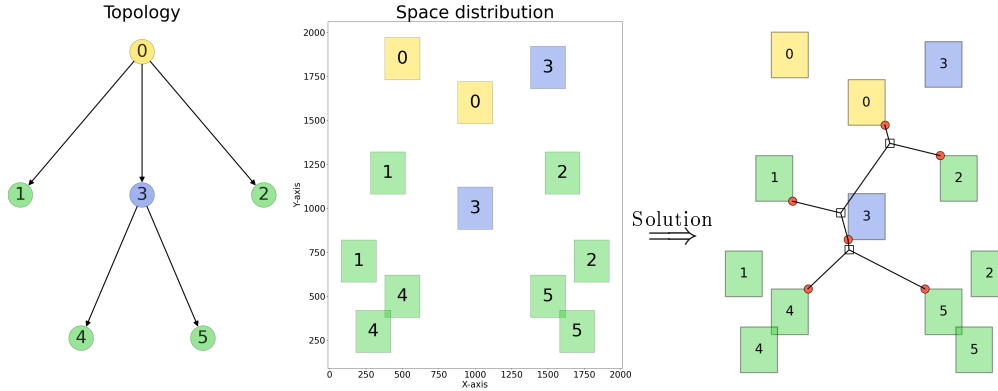


Figure 4.1: Representation of the graph T and the neighborhoods assigned to each node (shown in the two left images) and a solution to the problem (shown on the right).

The FTTNP shares characteristics with various Combinatorial Optimization problems with Neighborhoods studied in the literature (see e.g., Blanco et al. [2017], Blanco and Puerto [2022], Puerto and Valverde [2024], Disser et al. [2014], Marcucci et al. [2024], Espejo et al. [2022]). In such problems, a combinatorial structure (e.g., a path, spanning tree, hub network, or matching) is embedded into a d -dimensional space, ensuring that nodes lie within predefined regions (or *neighborhoods*) and optimizing a specific metric. These neighborhoods account for uncertainties in node positions or provide flexibility for placing elements in the embedding space.

For instance, the Minimum Spanning Tree with Neighborhood Problem (MSTNP), involves constructing a minimum-distance spanning tree with nodes placed within given neighborhoods Blanco et al. [2017], Yang et al. [2007], Dorigiv et al. [2015], Disser et al. [2014]. It is well-established that neighborhood versions of combinatorial optimization problems, even those solvable in polynomial time in discrete settings, become NP-hard when incorporating continuous location requirements within neighborhoods.

Although the FTTNP resembles a neighborhood version of a combinatorial optimization problem -embedding a tree in a given space- its objective is not to minimize the sum of individual edge lengths. Instead, the goal is to minimize the overall cable length routed through the structure, allowing for additional connection points if they reduce the total length. This problem is closely related to a neighborhood

version of the Steiner Tree Problem (STP).

The (continuous) STP involves finding a minimum-length (for a given metric) spanning tree in a graph, including at least a specified set of nodes (*terminals*) and possibly additional nodes (*Steiner points*). The STP is an NP-hard problem, even in the Euclidean case [Fampa et al., 2016, Garey et al., 1977], and is of great interest for both theoretical and practical reasons. In continuous spaces, the STP seeks the shortest network connecting a set of points according to a specified distance metric. The STP generalizes the Weber problem, which minimizes the sum of distances to a given set of points Drezner et al. [2002], Fekete et al. [2005]. The STP (both in continuous and discrete domains) has been extensively studied, and several exact and heuristic approaches have been proposed Ljubić et al. [2006], Ljubić [2021], Pinto and Maculan [2023].

In this chapter, we propose a mathematical optimization-based approach for the FTTNP, which we derive by showing its equivalence to a specific neighborhood version of the STP. Thus, the FTTNP also shares features with the Fixed Topology Steiner Tree (FTST) problem Sankoff and Rousseau [1975], Kiefner [2016], Lusheng and Jia [1999]. In the FTST, a tree structure is given, along with a partial embedding of nodes in a metric space. The task is to embed the remaining nodes to minimize the overall tree length. In contrast, the FTTNP provides a single set of nodes with a tree-structured relationship, specifying regions for node placement rather than fixed points. Moreover, the FTTNP permits additional points in the metric space to minimize the total embedded tree length. Notably, the FTST problem has primarily been analyzed using rectilinear distances, whereas we address general ℓ_τ distances in d -dimensional spaces. Additionally, the previous approaches are based on combinatorial constructions and analyze the theoretical complexity of the problems. Our approach leverages mixed-integer linear and nonlinear optimization models, enabling the use of standard optimization software to serve as decision-support tools for practitioners.

The FTTNP addresses real-world challenges in cable and pipeline routing design. Applications include naval architecture, wind farm layout, and circuit design (see, e.g. Blanco et al. [2022, 2023], Fischetti and Pisinger [2018], Liu et al. [2024]). Thus, our approach has a direct impact on challenging industrial problems that arise in many different fields. To tackle these challenges, we provide two frameworks for the FTTNP:

- **Continuous-domain framework:** Here, the tree embedding is allowed in the entire d -dimensional space $\mathbb{R}^d$, with nodes placed within specified neighborhoods. These neighborhoods are assumed to be unions of second-order cone representable sets, allowing flexibility in design.
- **Discrete-domain framework:** In this approach, an embedded graph in $\mathbb{R}^d$ is provided, and the tree is embedded within this graph. Nodes of the tree are mapped to graph nodes, and edges correspond to shortest paths in the graph. This framework accommodates additional technical requirements, such as obstacles or less preferred links, which are challenging to address in the continuous-domain framework.

Building on the discrete-domain framework, we extend our approach to address

the *Wiring Diagram Problem* (WDP), a complex industrial layout task introduced by Ghenova, a leading naval engineering firm. The WDP involves routing cables or pipelines in a 3D space, modeled as a forest of directed trees rooted at main pipelines and extending to terminal devices. This problem incorporates practical constraints such as safety distances between branches, and obstacle avoidance, making it highly relevant for industrial applications like ship design and motor control centers. We propose an integer linear programming model for the WDP, leveraging a graph-based discretization of the 3D space to ensure computational tractability while meeting stringent technical requirements.

In summary, the main contributions of this chapter are as follows:

- We propose a novel optimization-based approach to address the FTTNP across both continuous and discrete domains, which are relevant for a variety of practical applications.
- For the continuous domain, we develop a general Mixed-Integer Second-Order Cone Programming (MISOCP) model, along with specialized formulations tailored to Euclidean and rectilinear metrics. For the discrete domain, we introduce a Mixed-Integer Linear Programming (MILP) model, along with enhancements that reduce the problem size and improve computational performance.
- We extend the discrete-domain framework to tackle the Wiring Diagram Problem (WDP), introducing an MILP model that incorporates realistic constraints such as safety distances, obstacle avoidance, and minimum spacing in 3D cable and pipeline routing.
- We present results from an extensive set of computational experiments that demonstrate the effectiveness and robustness of our approach to solving the FTTNP.

The remainder of the chapter is structured as follows. Section 4.2 introduces the FTTNP. In Section 4.3, we show its equivalence to a particular variant of the Fixed Topology Minimum-Length Steiner Tree with Neighborhoods Problem. We then develop a general model for the continuous case, capable of addressing arbitrary metrics in d -dimensional space. Specialized formulations for the Euclidean and rectilinear cases are also presented to further strengthen the model. Section 4.4 focuses on the discrete case, introducing a tailored mathematical model for this framework. In Section 4.5, we extend the discrete-domain approach to the Wiring Diagram Problem, addressing complex 3D routing scenarios under realistic industrial constraints. Section 4.6 presents a detailed analysis of our computational experiments, comparing algorithmic performance and solution characteristics across different FTTNP frameworks. Finally, Section 4.7 summarizes our key findings and outlines potential directions for future research.

4.2 Fixed Topology Minimum-Length Trees with Neighborhoods

In this section, we introduce the problem under analysis and establish the notation used throughout the chapter.

We are given a rooted, tree-shaped graph (arborescence) $T = (V, A)$, where $V = \{v_0, \dots, v_n\}$ (with v_0 identified as the root of the tree), representing the set of nodes, and A denotes the set of arcs. The index set for the nodes is denoted as $N = \{0, \dots, n\}$. The hierarchical (parent-child) relationships in the tree are defined by the mapping $P : V \setminus v_0 \rightarrow V$, where $P(v)$ denotes the unique parent of node $v \in V \setminus \{v_0\}$. For convenience, we use C to denote the inverse mapping, which associates each non-leaf node with its set of children. Specifically, $C : V \rightarrow \mathcal{L}^{2^V}$ is defined such that $C(v) = \{w \in V : P(w) = v\}$. This topology specifies only the connections between nodes; it does not determine the nodes' positions or the lengths of the arcs.

Each node $v_i \in V$ is associated with a neighborhood $\mathcal{R}_i \subset \mathbb{R}^d$, indicating the set of possible positions in $\mathbb{R}^d$ where v_i can be embedded. Additionally, a distance metric D in $\mathbb{R}^d$ is provided to measure the lengths of the arcs in the embedding. The goal of the FTTNP is to find an embedding $\rho : V \rightarrow \mathbb{R}^d$ such that $\rho(v_i) \in \mathcal{R}_i$, it satisfies the hierarchy defined by T and minimizes the total length of the embedded tree.

Figure 4.1 provides an illustrative instance of the FTTNP. The left plot shows the rooted tree T , specifying the required topology. The embedding is constrained to account only for the lengths of paths connecting children to their parents in the tree. The right plot depicts the neighborhoods for each node. For example, the root node v_0 can be positioned only within the regions labeled "0", node v_1 within the "1"-labeled regions, and so on.

As discussed in the introduction, a key practical application of this problem arises in cable routing. In building or ship design, once primary pipelines or cable trunks are laid out in 3D space according to constructibility criteria (see Blanco et al. [2022, 2023] for further details), the challenge shifts to optimally routing cables from these primary pipelines to terminals while minimizing cable length. Guided by a schematic plan, valves are positioned in predefined regions to meet technical requirements. This challenge aligns with the FTTNP, where the schematic layout of each pipeline forms a rooted tree, and valves act as intermediate points optimizing cable paths.

It is important to distinguish the FTTNP from the Fixed Topology Minimum-Length Spanning Tree Problem with Neighborhoods (MSTN).

The MSTN is formulated as follows:

$$\begin{aligned} \min \quad & \sum_{(v_j, v_k) \in A} D(\mathbf{x}_j, \mathbf{x}_k) \\ \text{s.t.} \quad & \mathbf{x}_j \in \mathcal{R}_j, \forall j \in N. \end{aligned} \tag{MSTN}$$

Here, each node v_j is embedded within its neighborhood (at coordinates $\mathbf{x}_j \in \mathcal{R}_j$) to minimize the sum of the arc lengths. Unlike the FTTNP, the MSTN does not allow additional intermediate points to minimize the overall length.

The MSTN can be efficiently solved (in polynomial time) by most off-the-shelf software if the distance measure is an ℓ_τ -norm or a polyhedral norm. This is because in those cases, the problem can be reformulated as a second-order cone or linear optimization problem (see e.g. [Blanco et al. \[2014\]](#), [Blanco and Martínez-Antón \[2024\]](#)) and solved using interior point methods. However, this approach may not yield desirable solutions for practical applications. Specifically, since the distances considered in the objective function of (MSTN) are calculated edge-by-edge, there is no encouragement for cases where multiple edges share part of the route from the root node to the leaves, which could result in less costly solutions. Moreover, solving (MSTN) may lead to solutions where different geometric edges overlap or cross, resulting in unrealistic outcomes for practical applications, such as those encountered in engineering designs where the edges may represent cables. The aim of this chapter is to develop an alternative methodology to obtain realistic solutions for the problem, taking into account the various scenarios that should be promoted or avoided when constructing solutions to be implemented in real-world situations.

Figure 4.2 compares the solution of MSTN and FTTNP. The two left plots represent a toy instance for the problem (the tree topology and the neighborhoods, respectively). The third picture shows the solution of the MSTN. That is, the nodes are embedded in their neighborhoods such that the sum of the lengths of the two embedded arcs is minimized. In contrast, the far-right picture shows a solution of the FTTNP, which, instead of separately minimizing each of the lengths, minimizes the design of the overall trace of the system.

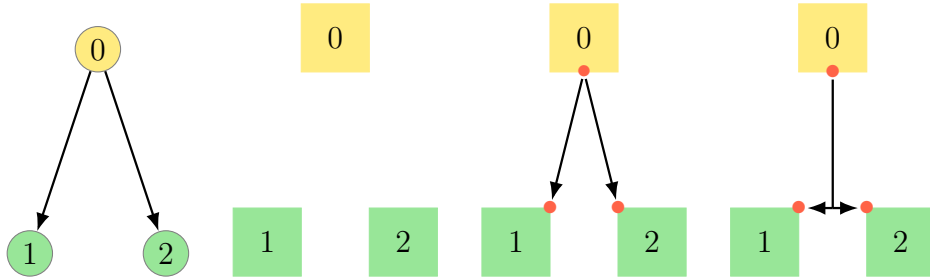


Figure 4.2: A toy instance to show the distance between the MSTN and FTTNP.

This chapter proposes alternative methodologies for embedding trees in d -dimensional space while minimizing the space occupied by the tree arcs. Specifically, we propose methodologies based on constructing minimum-length Steiner Trees with specifications provided by the input tree. Along these lines, we develop two approaches, each with its own advantages and disadvantages, based on whether or not the positions of the embedding are restricted to a finite set of positions in the neighborhoods. Both approaches differ in the domain of the embedding, and can be useful in different situations. Furthermore, we provide a data-driven methodology to derive approximate solutions for the continuous case by using the discrete approach on a certain graph obtained from the original topology of the tree and neighborhoods.

4.3 The continuous-domain FTTNP

In this section, we extend (MSTN) to allow for the sharing of paths from the root node to the leaves among certain edges, aiming to reduce the overall length of the tree. We achieve this by formulating the FTTNP. Our methodology has two main components: (1) transforming the input tree, T , by adding an additional set of nodes and edges that incorporate the overall design length into the problem; and (2) determining the tree structure and the geometric positions of the nodes in $\mathbb{R}^d$, with the possibility of discarding some of the newly introduced sets of nodes, using a Steiner tree-like model.

In the following subsections, we detail these steps.

4.3.1 Graph Transformation

The first step of our approach involves constructing a new graph from the original input tree that allows us to adequately model the FTTNP as a suitable mathematical optimization problem. The key idea of this transformation is to enable the paths resulting from embedding the arcs in the input graph to be shared among different arcs. Thus, we replace parent-children arcs by two-leg paths with auxiliary nodes which are all connected between them. In the following, we detail this construction.

Given the input rooted tree $T = (V, A)$, we construct a new graph $\tilde{T} = (\tilde{V}, \tilde{A})$ as follows:

- We initialize $\tilde{T} = T$.
- For each $v \in V$ with $|C(v)| \geq 2$ (indicating that node v has at least two children in the tree), we remove all arcs between v and its children, i.e.,

$$\tilde{A} \leftarrow \tilde{A} \setminus \{(v, w) : w \in C(v)\}, \text{ for all } v \in V.$$

- For each node $v \in \tilde{V}$, $|C(v)| - 1$ new nodes, $w_1^v, \dots, w_{n_v-1}^v$, are added to $\tilde{V}$, as well as arcs with source v and target to each of these new nodes, and arcs from each of these new nodes to each of the children:

$$\tilde{A} \leftarrow \tilde{A} \cup \{(v, w_j^v)\} \cup \{(w_j^v, w) : w \in C(v)\},$$

for all $j = 1, \dots, n_v - 1$. Furthermore, all the auxiliary nodes for node v are completely linked in the new graph, i.e.,

$$\tilde{A} \leftarrow \tilde{A} \cup \{(w_j^v, w_k^v) : j, k = 1, \dots, |C(v)| - 1, j \neq k\}.$$

Note that with this construction the graph $\tilde{T}$ is no longer a tree (all the auxiliary nodes that are children of the same parent are fully connected), but it is still connected.

In Figure 4.3, we illustrate this transformation. On the left side, a simple tree topology T is shown. The right picture depicts the transformed graph $\tilde{T}$ for this

tree. Note that the spatial positions of these nodes are not representative of the embedding we aim to construct in our approach; only the topology of the tree. The new graph includes all the nodes of T , as well as the new nodes 6, 7, and 8. Nodes 6 and 7 are incorporated because node 0 has three children (nodes 1, 2, and 3). Thus, arcs $(0, 6)$ and $(0, 7)$ are added, as well as arcs linking these nodes with the children nodes. Similarly, node 8 appears because node 3 has two children, but only the arcs linking the parent to it, and it to the children, are added.

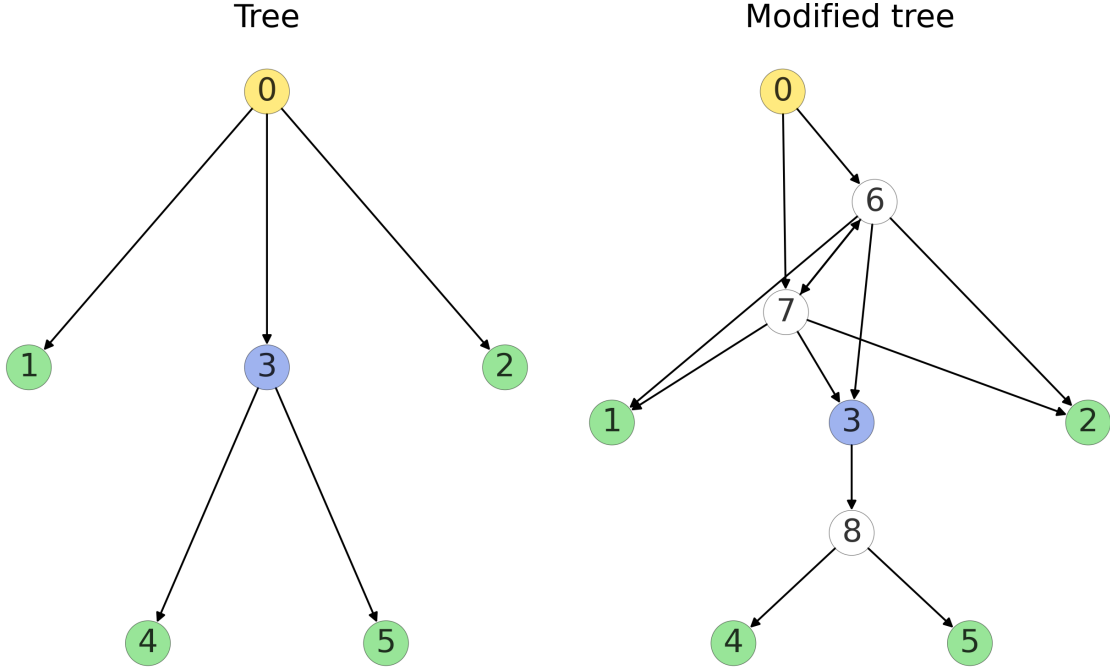


Figure 4.3: Representation of tree graph T and the transformation $\tilde{T}$.

Furthermore, each node $v \in V$ in the tree T is endowed with a neighborhood $\mathcal{R}_v \subseteq \mathbb{R}^d$. In the transformed graph $\tilde{T} = (\tilde{V}, \tilde{A})$, the nodes in V retain their original neighborhoods. However, for nodes $w \in \tilde{V} \setminus V$, the neighborhoods are defined as $\mathcal{R}_w = \mathbb{R}^d$, indicating that any position in space is feasible for these nodes.

4.3.2 The Steiner tree problem on $\tilde{T}$

In this phase, we equivalently model the FTTNP as a restricted Steiner tree problem over the transformed graph $\tilde{T}$.

Given a (di)graph and a given set of terminal nodes from the graph's node set, the goal of the Steiner tree problem is to construct a minimum length tree in the graph that at least uses the terminal nodes, while also being allowed to use some of the other nodes from the graph.

A Minimum Length Spanning Tree connects the nodes for which there is a direct arc linking them directly, which might not always be the most cost-effective way to lay cables if intermediate nodes (additional points where cables can be laid) can help to reduce the total cable length. In contrast, Minimum Length Steiner Trees

allow for the use of additional nodes that can significantly reduce the total length of the resulting network. Furthermore, Steiner trees provide more flexibility in the placement of cables, as they are not constrained to a fixed number of nodes. This can be particularly useful in physical spaces where laying cable directly between all original points is not feasible or too costly. In any case, if no new nodes are needed, the Steiner tree problem will decide not to incorporate them, and its solution will coincide with the minimum spanning tree.

Denoting by $\tilde{V} = \{v_0, \dots, v_{n+1}, v_{n+2}, \dots, v_{|\tilde{V}|}\}$ the labels for the nodes of the transformed graph, and by $N' = N \cup \{n+1, \dots, |\tilde{V}|\}$ its index set, the Steiner tree problem that we analyze here can be formally formulated as follows:

$$\min_{\substack{S \subset \tilde{V} \\ B \subset \tilde{A} \\ (v_j, v_k) \in B}} \sum D(\mathbf{x}_j, \mathbf{x}_k) \quad (4.1)$$

$$\text{s.t. } v_j \in S, \quad (4.2)$$

$$(S, B) \text{ is a tree,} \quad (4.3)$$

$$\mathbf{x}_j \in \mathcal{R}_j, j \in N, \quad (4.4)$$

$$\mathbf{x}_j \in \mathbb{R}^d, j \in N' \setminus N, \quad (4.5)$$

where the coordinates $\mathbf{x}_j$ indicate the embedding of the nodes in $\mathbb{R}^d$.

The objective function minimizes the total length of the tree. Constraint 4.2 ensures that the *terminals* (original nodes of the tree) are in the resulting subgraph (which is required to be a tree by constraint (4.3)). Constraints (4.4) and (4.5) define the domains for the embedding. Here, abusing notation we identify $\mathcal{R}_{v_j}$ with $\mathcal{R}_j$. For terminal nodes, the embedding is constrained to their neighborhoods, and for the Steiner points, it is allowed in the whole space.

Observe that the problem above is not exactly a Steiner tree problem on a Euclidean space since some hierarchical conditions are imposed on the problem to respect the required topology.

4.3.3 Mathematical Optimization Model

To construct a model for the FTTNP, we formulate it as a Mixed Integer Nonlinear Optimization problem through a flow-based formulation. Apart from the $\mathbf{x}$ -variables already described, we use the following sets of variables:

$$y_{jk} = \begin{cases} 1, & \text{if arc } (v_j, v_k) \text{ is part of the resulting tree,} \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{for all } (v_j, v_k) \in \tilde{A}.$$

$$f_{jk}^\ell = \begin{cases} 1, & \text{if arc } (v_j, v_k) \text{ is used in the path from the root node to the leaf node } v_\ell, \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{for all } (v_j, v_k) \in \tilde{A}, \text{ and } v_\ell \in \tilde{\mathcal{L}} \text{ (here } \tilde{\mathcal{L}} \text{ denotes the set of leaves in the original tree).}$$

$$d_{jk} \in \mathbb{R}_+: \text{ Length of arc linking } \mathbf{x}_j \text{ and } \mathbf{x}_k, \text{ for all } (v_j, v_k) \in \tilde{A}.$$

The model, that will be referred as (C), can be stated as follows:

$$\min \sum_{(j,k) \in \tilde{A}} d_{jk} y_{jk} \quad (\text{C}_1)$$

$$\text{s.t.} \quad \sum_{\substack{k \in N: \\ (v_0, v_k) \in \tilde{A}}} f_{0k}^\ell = 1, \quad \forall \ell \in \mathcal{L}, \quad (\text{C}_2)$$

$$\sum_{\substack{k \in N: \\ (v_j, v_k) \in \tilde{A}}} f_{jk}^\ell - \sum_{\substack{k \in N: \\ (v_k, v_j) \in \tilde{A}}} f_{kj}^\ell = 0, \quad \forall \ell \in \mathcal{L}, j \in N' \setminus N, \quad (\text{C}_3)$$

$$\sum_{\substack{j \in N: \\ (v_j, v_\ell) \in \tilde{A}}} f_{j\ell}^\ell = 1, \quad \forall \ell \in \mathcal{L}, \quad (\text{C}_4)$$

$$f_{jk}^\ell \leq y_{jk}, \quad \forall \ell \in \mathcal{L}, j, k \in N', (v_j, v_k) \in \tilde{A}, \quad (\text{C}_5)$$

$$d_{jk} \geq D(\mathbf{x}_j, \mathbf{x}_k), \quad \forall j, k \in N' : (v_j, v_k) \in \tilde{A}, \quad (\text{C}_6)$$

$$\mathbf{x}_j \in \mathcal{R}_j, \quad \forall j \in N, \quad (\text{C}_7)$$

$$\mathbf{x}_j \in \mathbb{R}^d, \quad \forall j \in N' \setminus N, \quad (\text{C}_8)$$

$$f_{jk}^\ell \geq 0, \quad \forall j, k \in N' : (v_j, v_k) \in \tilde{A}, \ell \in \mathcal{L}, \quad (\text{C}_9)$$

$$y_{jk} \in \{0, 1\}, \quad \forall j, k \in N'. \quad (\text{C}_{10})$$

In the above formulation, the objective function (C₁) aims to minimize the total length of the activated arcs of the tree. Constraints (C₂) ensure that exactly an arc is activated in the solution, departing from the root. Constraints (C₃) represent the flow conservation constraint for each leaf node. Constraints (C₄) guarantee that each leaf node is reached by exactly a single arc. Constraints (C₅) prevent the activation of flows on arcs that are not utilized. Constraints (C₆) ensure the proper definition of the lengths of the arcs in the tree. Finally, Constraints (C₇) and (C₈) are the domains for the embedding (the original nodes in the neighborhoods and the auxiliary nodes freely in $\mathbb{R}^d$). Note that the domain of the f -variables can be relaxed to the nonnegative orthant.

Note that the above formulation is a Mixed Integer Nonlinear Optimization problem due to the objective function and constraints (C₅) and (C₆). However, the objective function can be linearized exactly by applying McCormick envelopes, and ℓ_p or polyhedral norm-based distances are efficiently represented through Second Order Cone Constraints (see e.g., Blanco et al. [2014], Blanco and Martínez-Antón [2024]) or linear inequalities. Additionally, if the neighborhoods are second-order cone representable or a union of second-order cone representable sets, one can still keep a Mixed Integer Second Order Cone Optimization model, that can be solved, at least for medium-sized instances, using off-the-shelf software such as Gurobi, CPLEX, or FICO.

Special Cases: ℓ_1 and ℓ_2 norm based distances

The Steiner Tree problem with particular distance choices for D has been studied in the literature in order to exploit the geometric properties to be incorporated into the mathematical optimization models (see e.g. Brazil et al. [2009, 2024], Fampa et al.

[2016], Du et al. [1993], Garey and Johnson [1977], Ljubić et al. [2006], Ljubić [2021], Hanan [1966], Maculan et al. [2000], Ouzia and Maculan [2022], Pinto et al. [2023], among many others). Although our problem is not strictly a Steiner tree problem on a continuous space (because of the requirements imposed by the topology of the tree), some of the properties of Steiner trees still apply to our problem.

Specifically, one can project out the flow variables in our model by replacing Constraints (C₂), (C₃), (C₄), and (C₅) in Subsection 4.3.3 by the following constraints that only involve the y -variables:

$$\sum_{\substack{k \in N: \\ (v_j, v_k) \in \tilde{A}}} y_{jk} = 1, \quad \forall j \in N' \setminus \mathcal{L}, \quad (7)$$

$$y_{jk} + y_{kj} \leq 1, \quad \forall j, k \in N' : (v_j, v_k), (v_k, v_j) \in \tilde{A}, \quad (8)$$

$$\sum_{\substack{k \in N': \\ (v_k, v_j) \in \tilde{A}}} y_{kj} = 1, \quad \forall j \in N \setminus \{0\}. \quad (9)$$

The first set of constraints specifies that from an original node of T , one can only proceed to a new *Steiner* node; the second set of constraints ensures that one is allowed to use an arc in a single direction; the third set of constraints requires that only one arc arrives at a node of the original tree, T .

Specifically, for ℓ_p norms, one can derive conditions on the degrees of the auxiliary nodes in the Steiner tree, as stated in the following result.

Lemma 4 (Du et al. [1993]). *Let D be a distance induced by a norm $\|\cdot\|$, i.e., $D(a, b) = \|a - b\|$, for $a, b \in \mathbb{R}^d$. If $\|\cdot\|$ is differentiable and strictly convex, then the degree of v_j for $j \in N' \setminus N$ is exactly 3.*

Note that the above result applies to any ℓ_p norm with $1 < p < \infty$, particularly to the Euclidean norm. For more information on the approximability of the Steiner tree problem in ℓ_p -metrics, see Fleischmann et al. [2024]. For the ℓ_1 -norm, we have the following result.

Lemma 5 (Hanan [1966]). *Let D be a distance induced by the ℓ_1 norm, i.e., $D(a, b) = \sum_{k=1}^d |a_k - b_k|$, for $a, b \in \mathbb{R}^d$. Then, the degree of v_j for $j \in N' \setminus N$ is either 3 or 4.*

The result above is based on the construction of the Hanan grid, which is obtained by drawing the hyperplanes parallel to the axes crossing each of the terminal nodes of a classical Steiner tree.

With the above lemmas, we incorporate these conditions into the model as linear inequalities involving the y -variables:

- If $D(a, b) = \|a - b\|_p$, for $1 < p < \infty$:

$$\sum_{\substack{j \in \tilde{V} \\ (i, j) \in \tilde{A}}} y_{ij} + \sum_{\substack{j \in \tilde{V} \\ (j, i) \in \tilde{A}}} y_{ji} = 3, \quad i \in P. \quad (10)$$

- If $D(a, b) = \|a - b\|_1$:

$$3 \leq \sum_{\substack{j \in \tilde{V} \\ (i, j) \in \tilde{A}}} y_{ij} + \sum_{\substack{j \in \tilde{V} \\ (j, i) \in \tilde{A}}} y_{ji} \leq 4, \quad i \in P. \quad (11)$$

Observe that this representation not only reduces the number of constraints in the model, but also drastically reduces the number of variables, since the flow variables f are no longer needed.

In Figure 4.4 we show the solutions of the same instance that we use to illustrate the problem above, for the ℓ_1 norm (left) and the ℓ_2 norm (right).

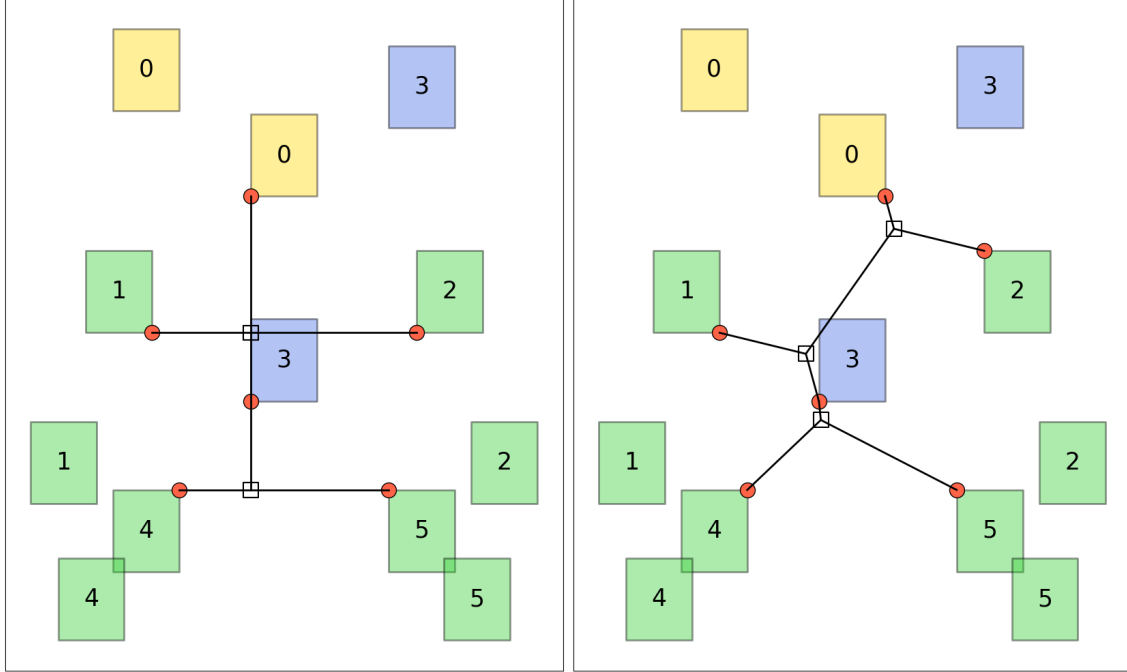


Figure 4.4: Graphical representation of the scenario solution. 4.1 for the norm ℓ_1 on the left and ℓ_2 on the right.

4.4 The discrete-domain FTTNP

In the discrete-domain case, we consider an undirected, connected graph $G = (U, E)$, where $U = \{u_1, \dots, u_{|U|}\} \subset \mathbb{R}^d$. Each node of the tree T , denoted v , is mapped to a corresponding node $\rho(v) \in U$ in the graph. The length of an embedded arc $(\rho(v), \rho(w))$ is defined as the shortest path in G , calculated using any of the available directions in the graph, with $\rho(v)$ as the source and $\rho(w)$ as the target.

Foundational insights into discrete-based approaches for solving Steiner problems have been provided by Maculan [1987], Aneja [1980], Diané [1993], Milos [2022], demonstrating the effectiveness and relevance of discrete methods in these optimization tasks.

One of the most commonly used graphs G is a grid in $\mathbb{R}^d$ with a specified density and width. When the grid density and width are sufficiently large, this discrete problem can approximate the solution to the continuous-domain case discussed in the previous section.

This discrete approach offers several practical advantages over the continuous-domain approach:

1. *Formulation as a Mixed-Integer Linear Optimization Problem:* The problem can be expressed as a mixed-integer linear optimization problem, which is generally more computationally efficient to solve than the nonlinear models derived for the continuous domain.
2. *Flexibility for Cable Routing Considerations:*
 - *Obstacle Handling:* Obstacles can be accounted for by removing the nodes and edges in G that are not permissible for cable routing.
 - *Preference Zones:* Preferences can be incorporated by assigning weights to graph edges, which are then used when calculating shortest path lengths.
 - *Direction Change Penalties:* The number of direction changes in cable routes can be minimized or penalized, as demonstrated in [Blanco et al. \[2022, 2023\]](#).

Another significant advantage of the discrete approach is its utility in approximating solutions to continuous-domain problems. Continuous-domain problems often require extensive computational resources, whereas the discrete approach can provide high-quality approximations within reasonable computation times. For example, the rectilinear Steiner tree problem in $\mathbb{R}^d$ is known to have a finite dominating set for Steiner points, constructible via the Hanan grid [Hanan \[1966\]](#). Although our version of the problem does not have a direct equivalent, a similar construction can yield effective approximations. The quality of these approximations will be evaluated empirically in Section 4.6.

Despite these advantages, discrete models also present certain limitations:

- **Discretization Constraints:** The optimal solution of the continuous model cannot be obtained if it lies outside the chosen discretization.
- **Impact of Inadequate Discretization:** Poorly designed discretizations can result to significant deviations from the true optimal solution. Similarly, overly dense discretizations may introduce a prohibitive number of variables and constraints, leading to computational costs that far exceed those of the continuous model.
- **Metric Approximation Limitations:** Not all metrics that can be utilized in the continuous model are accurately approximable in the discrete model.

In the subsequent subsections, we elaborate on this procedure in detail.

4.4.1 Mathematical Optimization Model

The main difference, in terms of the mathematical optimization model, between the continuous and discrete-domain frameworks, is that nonlinearities can be avoided by computing, in preprocessing, the lengths of the arcs involved in the construction, since the nodes are embedded into a finite set of potential points in $\mathbb{R}^d$ (this also helps in defining more sophisticated distances). In contrast, the input data for the

discrete problem can be very large (the number of nodes and edges in the graph G) which implies a large number of variables and constraints in the model.

Let us start introducing the mathematical model that we propose by describing the variables that will appear in the formulation. For the ease of presentation, we introduce the following sets:

- $N_G = \{1, \dots, |U|\}$: index set for the nodes in G .
- $\mathcal{R}_j^G = \{i \in N_G : u_i \in \mathcal{R}_j\}$: the nodes of G in the neighborhood of node $j \in N'$ (for those nodes in $N' \setminus N$, $\mathcal{R}_j^G = N_G$).

And the following decision variables:

$$\begin{aligned}
 f_{i\ell}^{jk} &= \begin{cases} 1, & \text{if the path from } v_j \text{ to } v_k \text{ is routed through edge } \{u_i, u_\ell\} \in E, \\ 0, & \text{otherwise,} \end{cases} \\
 &\quad \forall j, k \in N : (v_j, v_k) \in V, i, \ell \in N_G : \{u_i, u_\ell\} \in E. \\
 t_{ji} &= \begin{cases} 1, & \text{if node } v_j \in V \text{ is embedded into node } u_i \in U, \\ 0, & \text{otherwise,} \end{cases} \\
 &\quad \forall j \in N, i \in N_G. \\
 q_{i\ell} &= \begin{cases} 1, & \text{if edge } \{u_i, u_\ell\} \in E \text{ is part of the constructed tree,} \\ 0, & \text{otherwise,} \end{cases} \\
 &\quad \forall i, \ell \in N_G.
 \end{aligned}$$

Note that we do not use a special set of variables to decide which nodes in V are activated as Steiner points or not, but all nodes will be embedded into the new graph. Since the potential Steiner points can be freely embedded in the whole graph, in case a particular node is not used in the final tree, it will coincide with another node, not being accounted for in the objective function and not affecting the length of the tree.

With these variables we propose the following model to solve the discrete-domain

FTTNP:

$$\min \sum_{\substack{i, \ell \in N_G: \\ \{u_i, u_\ell\} \in E}} d_{i\ell} q_{i\ell} \quad (\text{D}_1)$$

$$\text{s. t. } \sum_{i \in \mathcal{R}_j^G} t_{ji} = 1, \quad \forall j \in N, \quad (\text{D}_2)$$

$$\sum_{\substack{\ell \in N_G: \\ \{u_\ell, u_i\} \in U}} f_{i\ell}^{jk} - \sum_{\substack{\ell \in N_G: \\ \{u_\ell, u_i\} \in U}} f_{\ell i}^{jk} = t_{ji}, \quad \forall j, k \in N : (v_j, v_k) \in A, i \in N_G, \quad (\text{D}_3)$$

$$\sum_{\substack{\ell \in N_G: \\ \{u_\ell, u_i\} \in U}} f_{\ell i}^{jk} - \sum_{\substack{\ell \in N_G: \\ \{u_\ell, u_i\} \in U}} f_{i\ell}^{jk} = t_{ki}, \quad \forall j, k \in N : (v_j, v_k) \in A, i \in N_G, \quad (\text{D}_4)$$

$$\sum_{\substack{i, \ell \in N_G: \\ \{u_i, u_\ell\} \in E}} (f_{i\ell}^{jk} + f_{\ell i}^{jk}) \leq q_{i\ell}, \quad \forall j, k \in N : (v_j, v_k) \in A, \quad (\text{D}_5)$$

$$q_{i\ell} \geq f_{i\ell}^{jk}, \quad \begin{aligned} &\forall i, \ell \in N_G : \{u_i, u_\ell\} \in E, \\ &j, k \in N' : (v_j, v_k) \in A, \end{aligned} \quad (\text{D}_6)$$

$$f_{i\ell}^{jk} \geq 0, \quad \forall j, k \in N' : (v_j, v_k) \in A, i, \ell \in N_G, \quad (\text{D}_7)$$

$$q_{i\ell} \in \{0, 1\}, \quad \forall i, \ell \in N_G. \quad (\text{D}_8)$$

The entire optimization problem is referred to as (D).

The objective function accounts for the total length of the tree. Constraints (D₂) enforce that all nodes in V are assigned to exactly one node in U (the embedding). Constraints (D₃) and (D₄) are the path-based constraints that ensure that a unit of flow is sent between the embeddings of nodes j and k . Constraints (D₅) avoid the use of both direction of the edges in the solution and that different origin-destination pairs share the same edges in the solution. Constraints (D₆) correctly define the variable q .

In Figure 4.5, we observe two distinct discretizations of the example scenario 4.1, along with the corresponding solutions obtained by solving the model (D₁-D₈). On the left, the solution space has been discretized using a square grid, while on the right, a triangular grid is utilized. Additionally, the positions of the final nodes are highlighted in orange, and the Steiner tree, representing the solution to the problem, is depicted in black.

4.4.2 A variable fixing strategy for geometric graphs

The graph G , representing the solution space in the discrete-domain case, can be dense, particularly when it is generated to discretize a continuous space. In this context, the graph G is typically a geometric graph inducing a grid in $\mathbb{R}^d$. In Figure 4.5 we showed dense grids (square and triangular) used as graph G . In such graphs, the special hierarchical shape of the problem may help to fix some of the f and q variables to zero, reducing the problem size and enabling the solution of larger instances more quickly. This reduction is more efficiently applied by removing parts of the graph that the paths between nodes will never use.

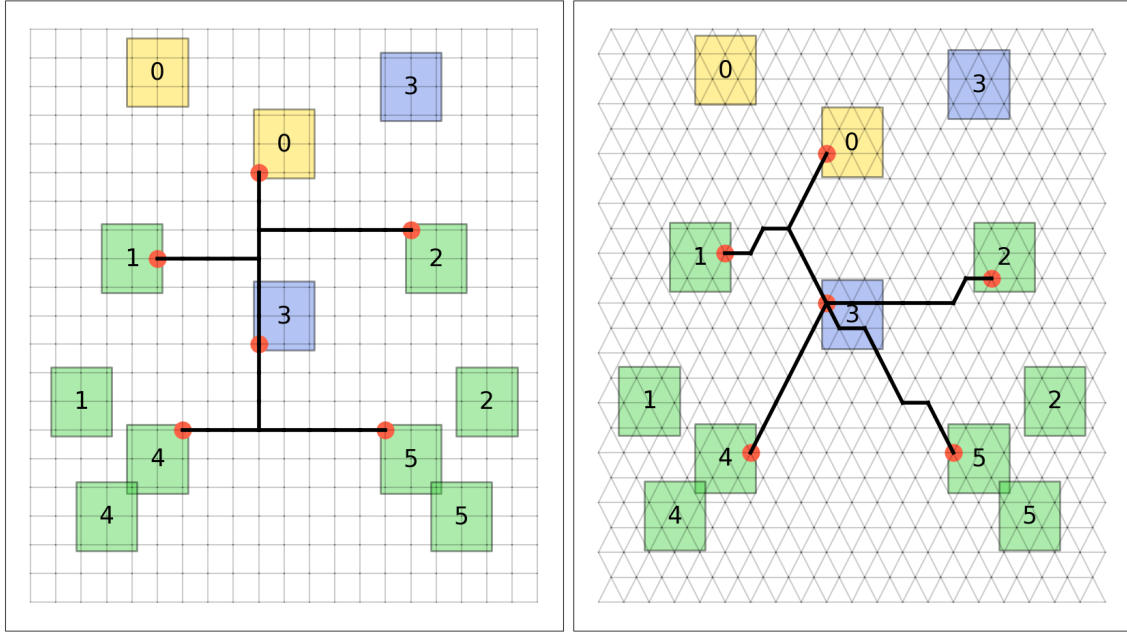


Figure 4.5: Graphical representation of the solution for scenario 4.1 using the formulation (D₁-D₈) for two different graphs G .

Let $v \in V'$ and $C(v)$ be its set of children. The paths in G linking v with its children start at some point in $\mathcal{R}_v$ and end at points in $\bigcup_{w \in C(v)} \mathcal{R}_w$. Let $\mathcal{C}_v$ be the convex (for ℓ_p -norms with $1 < p < \infty$) or rectangular (for $p = 1, \infty$) hull of these two sets. Thus, the paths linking v with its children in the optimal solution, since the lengths of the edges are induced by norm-based distances, will never use edges outside $\mathcal{C}_v \cap E$. Consequently, the variables f_{il}^{jk} and q_{il} can be fixed to zero for any $\{i, \ell\} \in E$ that is not in $\mathcal{C}_v$ (j is the index identified with v and k the indices of each of its children).

In Figure 4.6, we illustrate this construction for a toy instance. In the two left plots, we show the topology of the tree T and the distribution of the neighborhoods in the grid graph, respectively. In the third picture, we highlight with a yellow box the space in the graph where the paths from 0 to 1, 2, and 3 will be traced, and in blue the space where the paths from 3 to 4 and 5 will be routed in an optimal solution. Thus, all the f variables for nodes/edges outside the yellow rectangle can be fixed to 0 for those links and assignments of the first depth of the tree, and analogously for the blue box. In the right plot, we show an optimal solution for this instance.

4.5 Wiring Diagram Problem

Building on the discrete-domain FTTNP approach from Section 4.4, we extend the model to address the **Wiring Diagram Problem** (WDP), a complex industrial layout task involving cable and pipeline routing. This problem was introduced by Ghenova, a leading naval engineering firm.

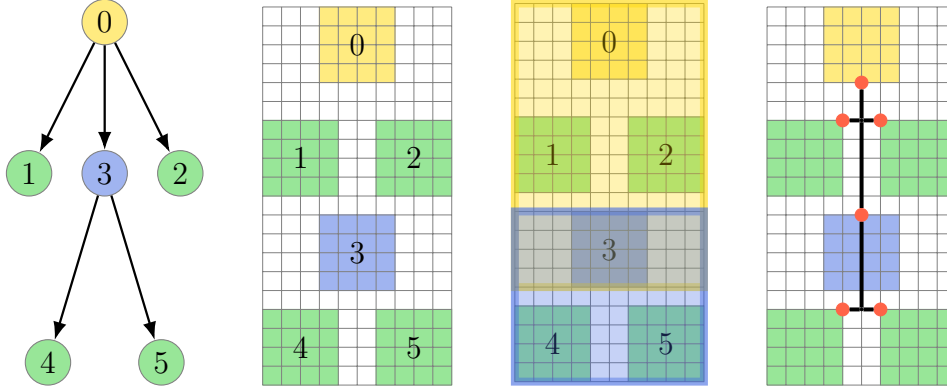


Figure 4.6: Illustration of the variable fixing strategy for geometric graphs.

Wiring diagrams are modeled as directed trees (or a forest), rooted in existing pipelines and extending toward terminal devices. Each node represents a physical component: root nodes are on pipelines, leaf nodes are fixed demand points, and intermediate nodes are within designated spatial regions.

A wiring diagram serves as a hierarchical reference plan, visually mapping cable connections in industrial systems. For instance, Figure 4.7 shows a motor control center where a main cable supplies power to various endpoints via intermediary systems.

We formalize the diagram as a finite forest $\mathcal{F}$ of rooted, directed trees, each associated with a main pipeline from the set $\mathcal{C}$. In each tree $T \in \mathcal{F}$, nodes $\mathcal{N}(T)$ and arcs $\mathcal{A}(T)$ form a connected, acyclic structure. The root node $r(T)$ is fixed on a pipeline, leaf nodes $\mathcal{L}(T)$ are at known final positions, and intermediate nodes $\mathcal{I}(T)$ lie within 3D regions defined by constructibility constraints.

Figure 4.8 illustrates an example forest with four trees, each with 10 nodes, anchored to pipelines c_0 and c_1 . Figure 4.9 shows the 3D spatial regions assigned to each node type: root nodes on pipelines, intermediate nodes in colored hyperboxes, and leaf nodes at final points.

Technical constraints include:

- All nodes must lie within prescribed 3D regions.
- Safety distances must be maintained between branches and from unrelated manifolds to allow maintenance and avoid interference.
- Obstacles in the space (e.g., restricted zones or structural elements) must be avoided.
- Minimum spacing must be observed between consecutive bends to ensure structural integrity.

The goal is to locate the forest in 3D space satisfying all constraints while minimizing total cable length (i.e., forest edge lengths).

Figure 4.10 presents a feasible solution to the instance shown in Figures 4.8 and 4.9, illustrating both the forest layout and node region assignments.

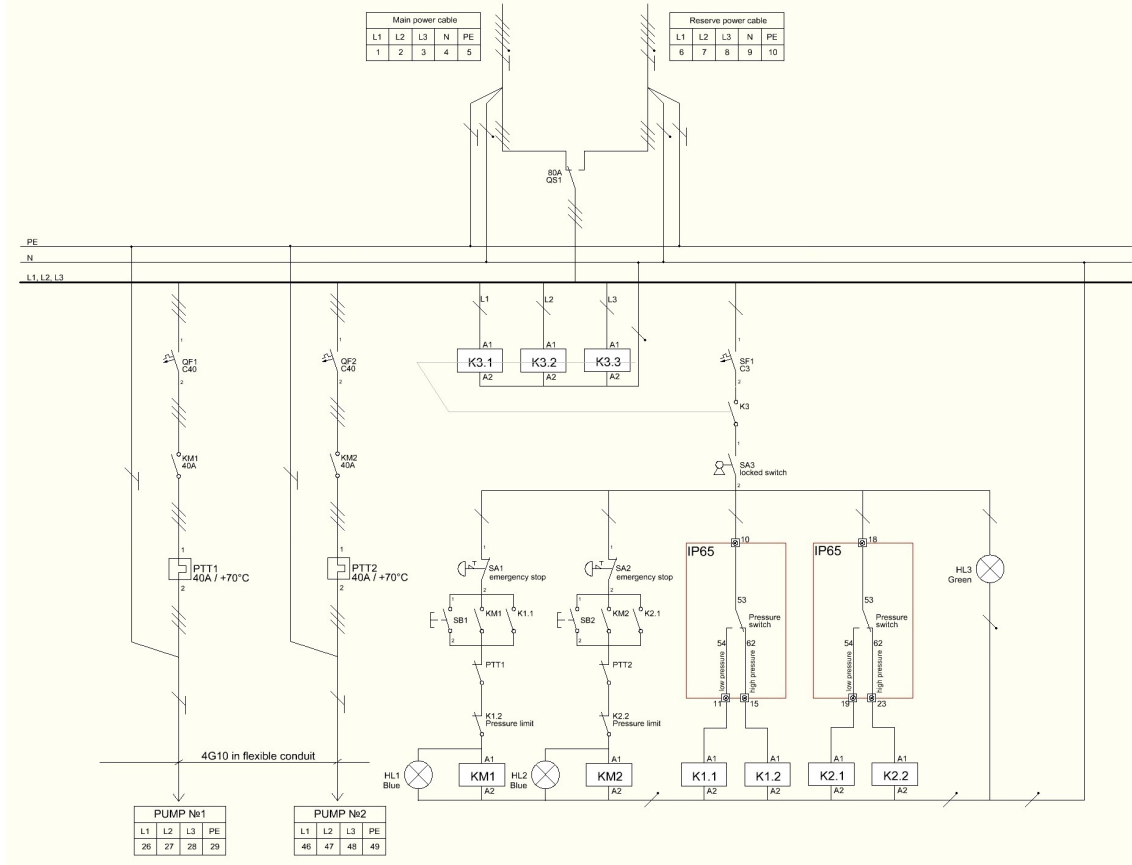


Figure 4.7: Example of wiring diagram (source: https://commons.wikimedia.org/wiki/Motor_control_centre#Wiring_diagrams).

4.5.1 A graph-based discretization of the space

The continuous 3D region $\mathcal{R} \subset \mathbb{R}^3$, where cables are routed, is discretized using an ad-hoc graph-based method to apply a network-flow approach. For each main pipeline $c \in \mathcal{C}$ and each tree T (with $\phi(T) \in \mathcal{C}$), an orthogonal 3D grid is constructed over $\mathcal{R}$. This ensures that pipeline endpoints, intermediate node regions, and terminal points align with grid nodes, which are connected to form a baseline undirected graph. Obstacles within $\mathcal{R}$ result in the removal of intersecting edges and nodes. This graph is denoted $\mathcal{G}(T) = (\tilde{N}(T), \tilde{E}(T))$, and the global graph is $\mathcal{G} = \bigcup_{T \in \mathcal{F}} \mathcal{G}(T)$. Each edge $e \in \tilde{E}(T)$ has a weight d_e , and each intermediate node s is associated with a region $\mathcal{R}_s$.

Due to the graph's density, finding feasible solutions is difficult. However, the hierarchical structure allows for a simplified graph with fewer variables. For a node $s \in \tilde{N}(T)$ and its children $C(s)$, paths from $\mathcal{R}_s$ to $\bigcup_{t \in C(s)} \mathcal{R}_t$ use the 3D Hanan grid $\mathcal{H}_s$ generated by the vertices of those regions. Given the use of the ℓ_1 -norm, optimal paths use only edges in $\mathcal{H}_s$.

Figure 4.11 illustrates the process: a subtree (left), its node regions (center), and the corresponding discretization (right), where dashed red lines show flow variables from source 1 to sink 2, and solid blue lines from source 2 to sinks 3 and 4.

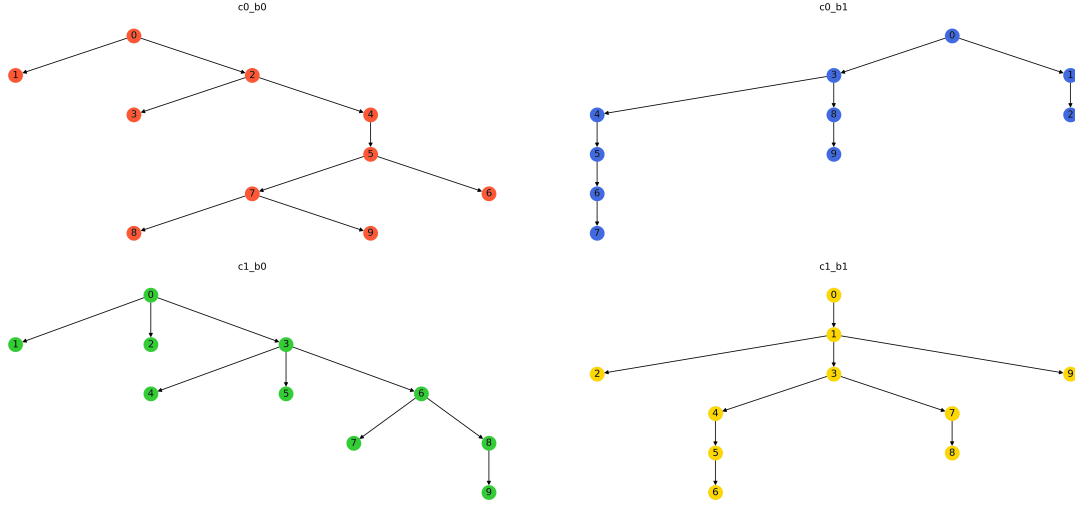


Figure 4.8: A forest with four trees, each of them with 10 nodes.

If the parent is a root node, the manifold's endpoints and direction-change points replace the valve space (Figure 4.12). If a child node is a leaf, a single point substitutes the valve space (Figure 4.13).

This discretization is applied to all trees in each branch, resulting in a solution space partitioned by grids, each corresponding to a branch.

4.5.2 Mathematical Optimization Model

We propose an integer linear programming model to solve the problem over a discretization graph, using the ℓ_1 -norm metric suitable for a finite set of points in $\mathbb{R}^3$.

We begin by introducing the notation used in the formulation. The key distance parameters are defined as follows:

- $\delta_{aa'}$: ℓ_1 -distance between arcs a and a' in $\mathbb{R}^3$, $a, a' \in \mathcal{A}$.
- $\Delta_a^c = \min_{a'} \delta_{aa'}$: minimum distance from arc a to arcs c in the pipeline.
- Δ : minimum safety distance between arcs.

The model employs the following binary decision variables:

$$\begin{aligned}
 f_a^{st} &= \begin{cases} 1 & \text{if path } s \rightarrow t \text{ uses arc } a, \\ 0 & \text{otherwise} \end{cases} \quad \forall a \in \tilde{\mathcal{A}}(T), s, t \in \tilde{\mathcal{N}}(T), T \in \mathcal{F}. \\
 y_v^s &= \begin{cases} 1 & \text{if node } s \text{ is at vertex } v, \\ 0 & \text{otherwise} \end{cases} \quad \forall v \in \mathcal{R}_s, s \in \mathcal{I}(T), T \in \mathcal{F}. \\
 x_a &= \begin{cases} 1 & \text{if arc } a \text{ is selected,} \\ 0 & \text{otherwise} \end{cases} \quad \forall a \in \tilde{\mathcal{A}}(T), T \in \mathcal{F}.
 \end{aligned}$$

Next, we describe the constraints that ensure the feasibility of the solution and enforce the technical requirements of the problem:

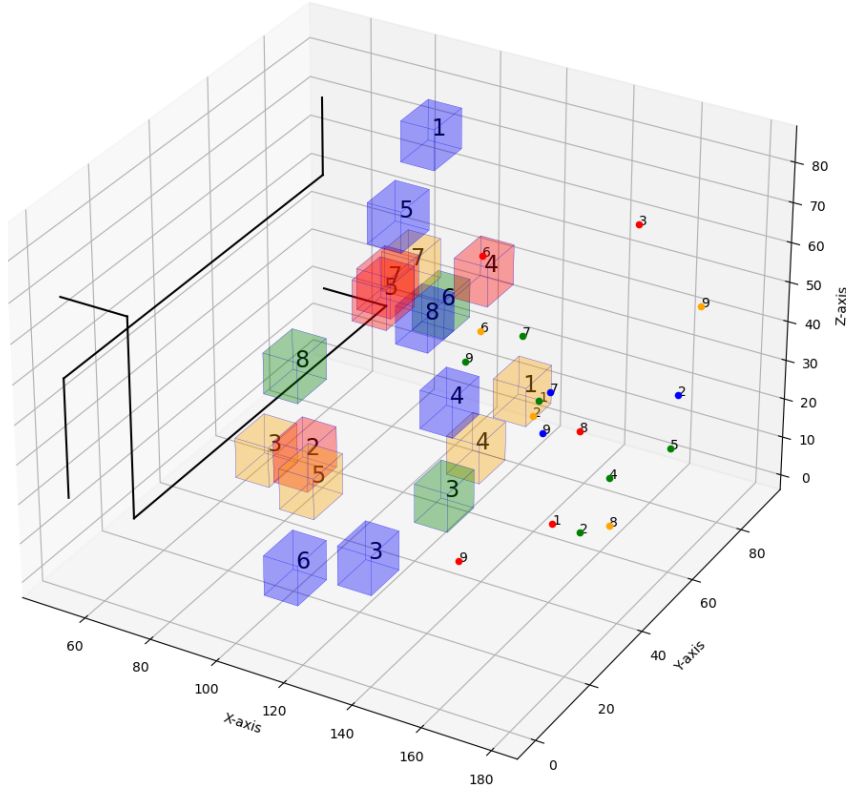


Figure 4.9: An instance for the location regions of each node in the forest 4.8.

- **Intermediate node placement:** exactly one position is selected in each region:

$$\sum_{v \in \mathcal{R}_s} y_v^s = 1, \quad \forall s \in \mathcal{I}(T), T \in \mathcal{F}.$$

- **Flow conservation:** flow must be preserved through all nodes:

$$\begin{aligned} \sum_{a \in \delta^-(v)} f_a^{st} - \sum_{a \in \delta^+(v)} f_a^{st} &= y_v^s, & \forall v \in \mathcal{R}_s, \\ \sum_{a \in \delta^-(v)} f_a^{st} - \sum_{a \in \delta^+(v)} f_a^{st} &= -y_v^t, & \forall v \in \mathcal{R}_t, \\ \sum_{a \in \delta^-(v)} f_a^{st} - \sum_{a \in \delta^+(v)} f_a^{st} &= 0, & \forall v \notin \mathcal{R}_s \cup \mathcal{R}_t. \end{aligned}$$

- **Arc usage control:** flow is only allowed through selected arcs:

$$f_a^{st} \leq x_a, \quad \forall a \in \tilde{\mathcal{A}}(T), s, t \in \tilde{\mathcal{N}}(T), T \in \mathcal{F}.$$

- **Root node constraint:** only one arc leaves the root node of each tree:

$$\sum_{a \in \delta^+(v)} x_a = y_v^{r(T)}, \quad \forall v \in \tilde{\mathcal{N}}(T), T \in \mathcal{F}.$$

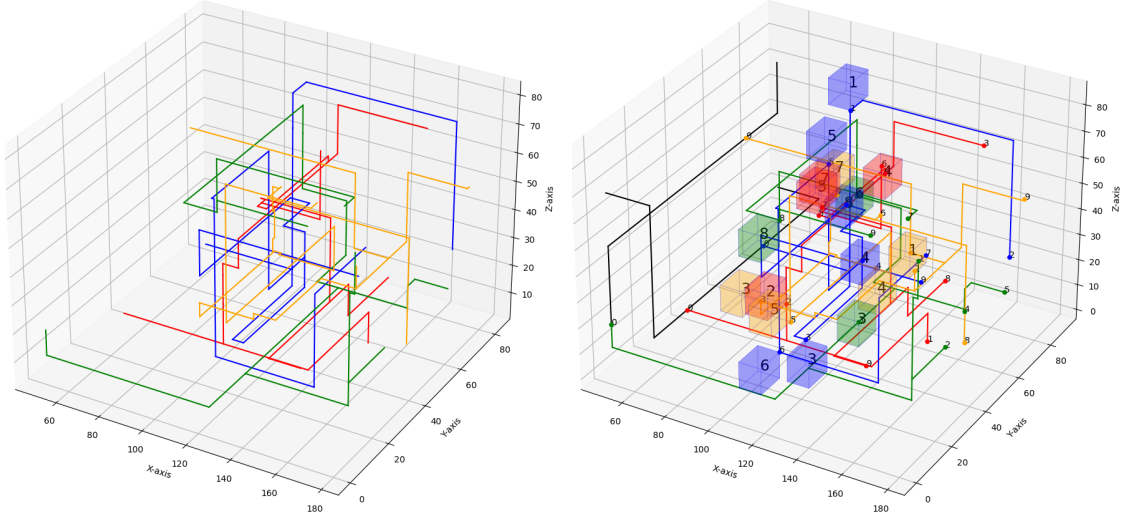


Figure 4.10: Illustration of a solution of the problem.

- **Leaf node entry:** only one arc enters each leaf:

$$\sum_{a \in \delta^-(\ell)} x_a = 1, \quad \forall \ell \in \mathcal{L}(T), T \in \mathcal{F}.$$

- **Leaf node exit restriction:** no arcs leave a leaf node:

$$\sum_{a \in \delta^+(\ell)} x_a = 0, \quad \forall \ell \in \mathcal{L}(T), T \in \mathcal{F}.$$

- **Unidirectional flow enforcement:** opposing directions on the same arc are disallowed:

$$f_a^{st} + f_{a'}^{st} \leq 1, \quad \forall a = (v, w), a' = (w, v) \in \tilde{\mathcal{A}}(T).$$

- **Safety distance between arcs:** arcs must respect spatial constraints:

$$\sum_{c' \in \mathcal{C}} \sum_{\substack{T' \in \mathcal{F}: \\ \phi(T')=c}} \sum_{\substack{a' \in \tilde{\mathcal{A}}(T'): \\ c' \neq c, \delta_{aa'} < \Delta}} x_{a'} \leq M(1 - x_a), \quad \forall a \in \tilde{\mathcal{A}}(T), T \in \mathcal{F}, c \in \mathcal{C}.$$

- **Pipeline-tree interaction:** trees rooted in different pipelines must maintain safety distances:

$$\sum_{\substack{c' \in \mathcal{C}: \\ c' \neq c}} \sum_{\substack{T' \in \mathcal{F}: \\ \phi(T')=c'}} \sum_{\substack{a \in \tilde{\mathcal{A}}(T'): \\ \Delta_a^c < \Delta}} x_a = 0, \quad \forall c \in \mathcal{C}.$$

Finally, the objective function seeks to minimize the total length of the selected arcs:

$$\min \sum_{a \in \tilde{\mathcal{A}}} d_a x_a$$

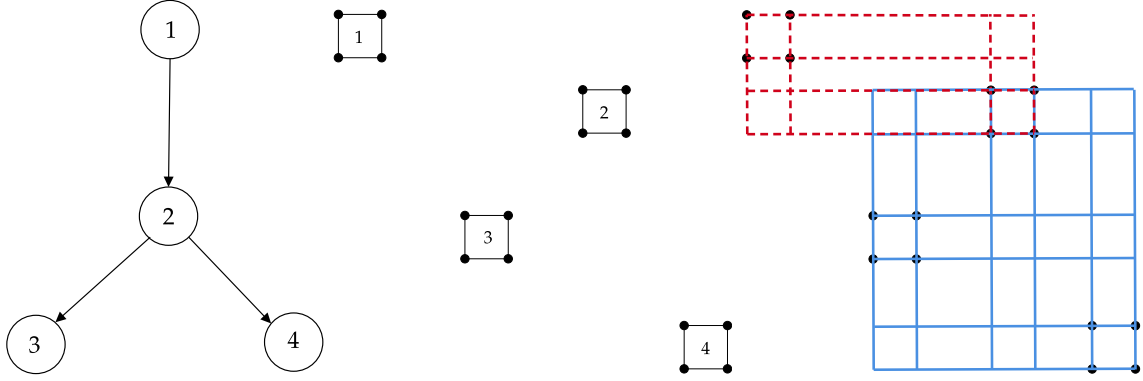


Figure 4.11: Example of solution space discretization. Left: a subtree; center: the spaces associated with the tree's nodes or valves; right: the discretization of the corresponding space.

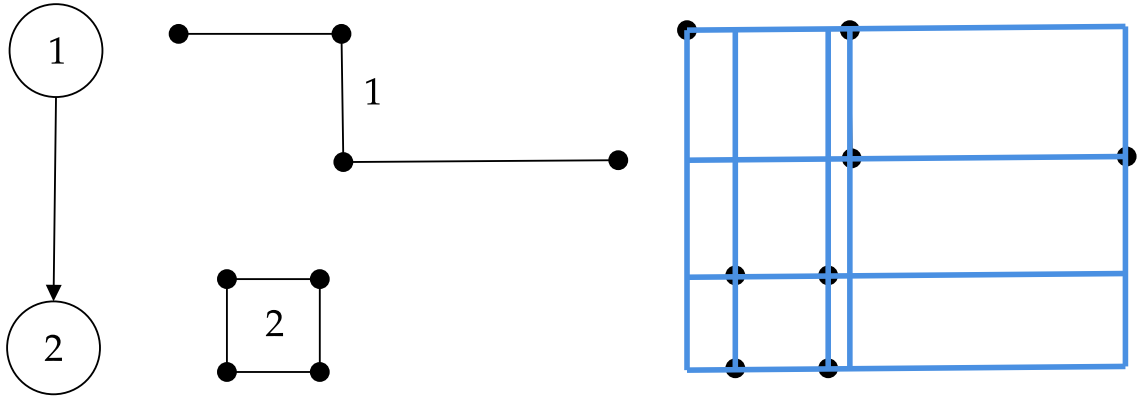


Figure 4.12: Example of discretization at a root node, where the endpoints of the manifold are considered.

4.6 Experiments

In this section, we present the results of our computational experiments, conducted to validate the performance of our approaches for the FTTNP. The specific instances that we generate as well as the detailed results are available at github.com/anticiclon/Fixed-Topology-Minimum-Length-Trees-with-Neighborhoods.

4.6.1 Instances

We generated a series of random instances with different characteristics. Each instance consists of a random tree with n nodes, with $n \in \{20, 50, 100, 200\}$ using the `random_tree` function in the python library `networkx`. The generated trees are relabeled so that node 0 is the root node, and it is directed from root to leaves.

We embed the tree in the square $[0, 2000] \times [0, 2000]$. For each tree, we generate b planar unions of square neighborhoods for $b \in \{1, 2, 3, 4, 5\}$ with different sizes $s \in \{20, 50, 100, 200\}$. Combining the different values for n , b , and s , the total

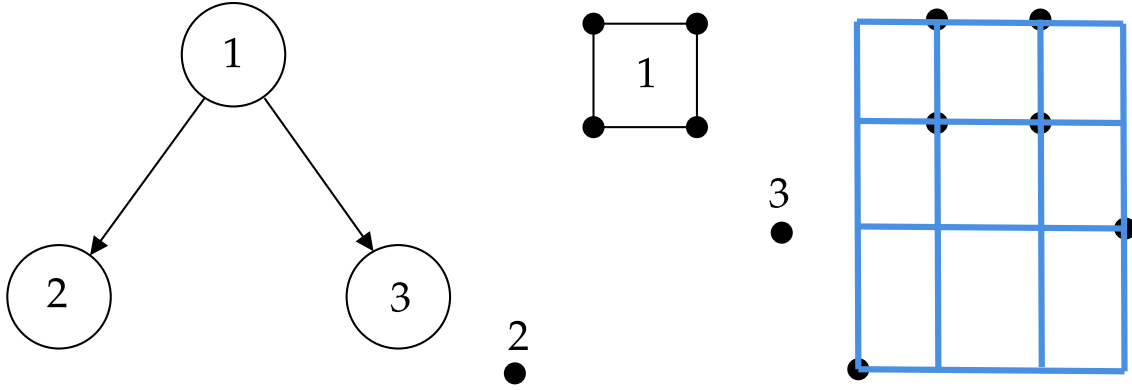


Figure 4.13: Example of discretization at a leaf node, where a single point is considered instead of a valve space.

number of instances generated was 400.

In Figure 5.5, we present an example of a random instance with $n = 20$ nodes, $b = 3$ connected components for the neighborhoods, and square side length $s = 200$, as well as the solution for the Euclidean norm.

All these scenarios were solved using three models:

- C**– ℓ_1 : Continuous-domain FTTNP (model **C**) with ℓ_1 -norm based distances.
- C**– ℓ_2 : Continuous-domain FTTNP (model **C**) with ℓ_2 -norm based distances.
- D**: We construct ad-hoc graphs G based on each instance. For parent-children combinations, we generate a rectangular grid by projecting the vertices of the neighborhoods of parents and children and tracing orthogonal lines. Then, we solve model **D**.

All instances were solved using the Gurobi 10.0.3 optimizer on an Ubuntu 22.04.3 environment, with an AMD EPYC 7042p 24-Core Processor and 64 GB RAM. Initially, all parameters of the Gurobi solver were set to their default values, and a CPU time limit of 7200 seconds was set.

A first tabular summary of the results is reported in Table 4.1. In that table, for each of the approaches, for each node size (n) and each number of components for the neighborhoods (b), we report the average required CPU time to solve the instances (only those up to the 20 per row that were optimally solved), the MIPGAP (of those not optimally solved), and the number of non-optimally solved instances (out of 20). For the **Disc** approach, all the instances were optimally solved, so we do not report the MIPGAP (all are 0%) nor the number of unsolved instances. The flag TL indicates that none of the instances summarized in that row could be optimally solved within the time limit (and then, the MIPgap is the average gap of all the instances in the row).

As expected, one can observe that as the number of nodes increases, the problem becomes more challenging. This also happens with the number of components for the neighborhoods (b).

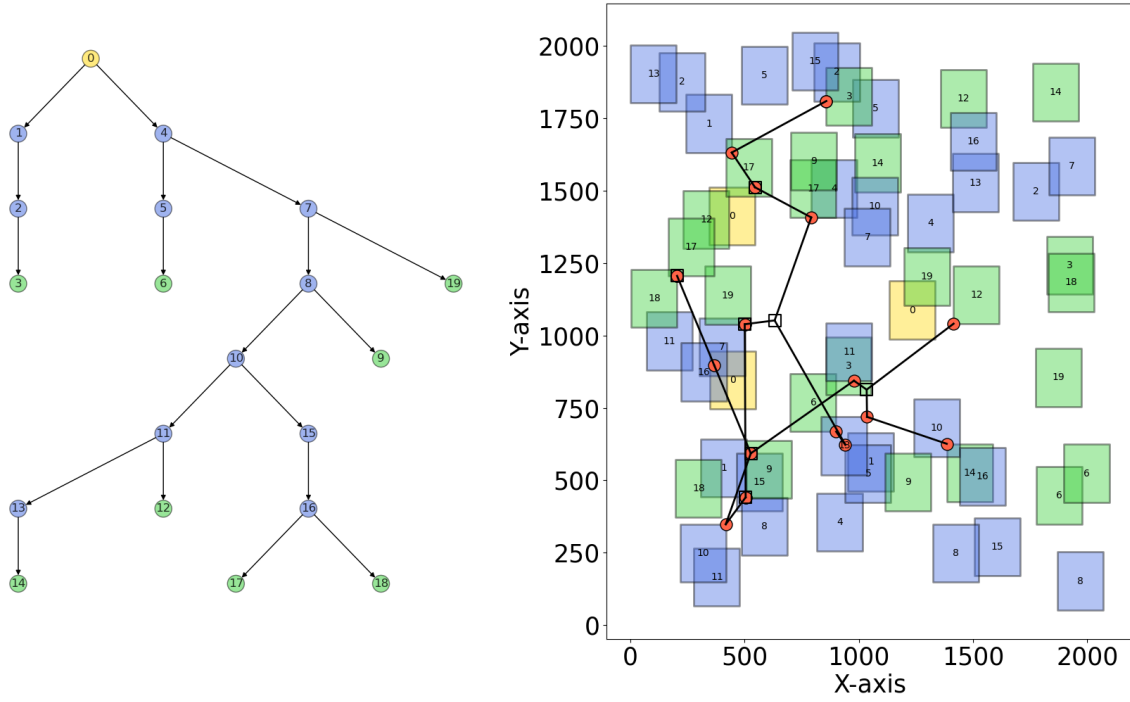


Figure 4.14: An example of a random instance with 20 nodes and 3 spaces per node solved by the continuous model with norm ℓ_2 .

n	b	$C - \ell_1$			$C - \ell_2$			D
		CPUTime	MIPGAP	UnSolved	CPUTime	MIPGAP	UnSolved	CPUTime
20	1	7.29	0%	0	30.04	0%	0	0.06
	2	0.58	0%	0	3.91	0%	0	0.34
	3	4.18	0%	0	62.05	0%	0	1.61
	4	7.72	0%	0	589.73	0%	0	6.43
	5	17.16	0%	0	361.46	13.9%	2	15.49
50	1	81.20	4.99%	1	370.49	11.7%	3	0.19
	2	227.84	22.68%	6	1301.79	7.4%	5	2.14
	3	354.74	13.52%	11	1795.31	8.7%	10	6.33
	4	1157.23	0%	0	3330.50	18.7%	17	21.91
	5	3613.66	0%	0	TL	22.7%	20	20.76
100	1	214.99	0%	0	1822.62	5.4%	16	0.48
	2	2723.87	7.07%	8	16.75	9.6%	17	6.61
	3	1527.23	8.92%	17	20.09	15.5%	19	17.23
	4	TL	22.30%	20	TL	30.7%	20	28.93
	5	TL	31.91%	20	TL	39.5%	20	105.03
200	1	123.87	2.76%	1	16.71	7.7%	20	1.25
	2	684.94	8.44%	17	20.11	11.9%	17	14.96
	3	TL	16.52%	20	TL	23.2%	18	57.38
	4	TL	27.37%	20	TL	35.4%	20	65.95
	5	TL	37.74%	20	TL	47.3%	20	237.23

Table 4.1: Average results of our computational experiments.

The discrete approach was able to find solutions much faster than the continuous approaches.

In Figure 4.15 we plot the performance profiles for the three methods on all the instances. On the x -axis we represent the CPU time. On the y -axis, the percentage of instances that were optimally solved within a time smaller than the one on the x -axis is shown. As already observed in the table, the discrete approach solved the instances in less than 2000 seconds, with most of them (80%) solved in less than 100 seconds. Among the continuous approaches, the ℓ_2 approach is more challenging, with only 40% of the instances optimally solved within the time limit. The ℓ_1 norm approach (whose formulation is a Mixed Integer Linear problem) solved 60% of the instances within the time limit.

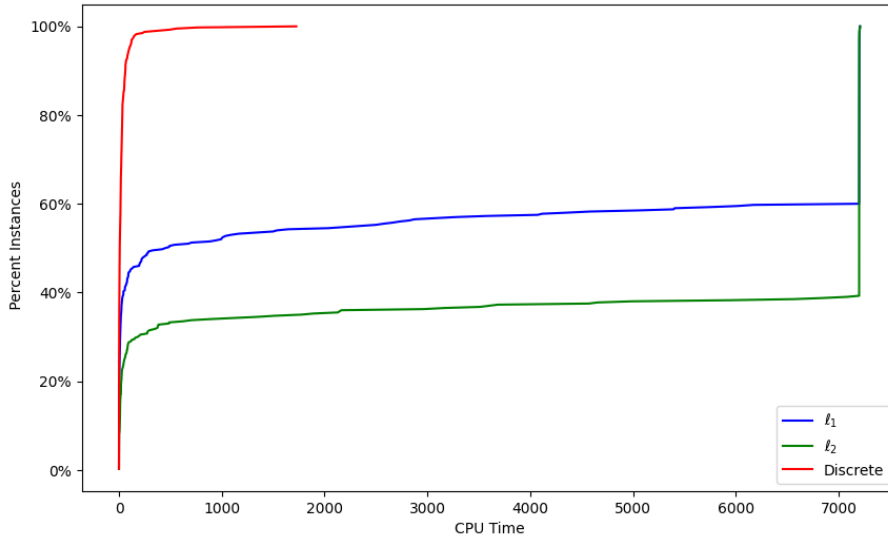


Figure 4.15: Graphical representation of the time-based performance profiles for the three methods.

We show in Figure 4.16 the MIPGAPs aggregated by the number of components in the neighborhoods. One can observe that the MIPGAP clearly increases with the number of components. Note that the components for each node are modeled as disjunctive constraints, introducing binary variables for each component. Therefore, as the number of components increases, the problem becomes more difficult.

Finally, since we construct the graph for the Discrete approach using a grid, we analyze how close the solution of the Discrete approach is to the solution of the continuous ℓ_1 . In Figure 4.17 we plot a boxplot showing the deviation of the objective value of the Discrete approach from the objective value of the continuous ℓ_1 approach, for each size(s) considered in our instances. The deviation is measured as:

$$\text{Dev} = \frac{\text{objective value } \mathbf{Disc} - \text{objective value } \mathbf{Cont } \ell_1}{\text{objective value } \mathbf{Disc}} \times 100$$

Note that the deviations are very small, most of them close to zero. Thus, the discrete approach appears to be a promising method to approximate the solution of the Continuous ℓ_1 problem, solving the instances in much less CPU time.

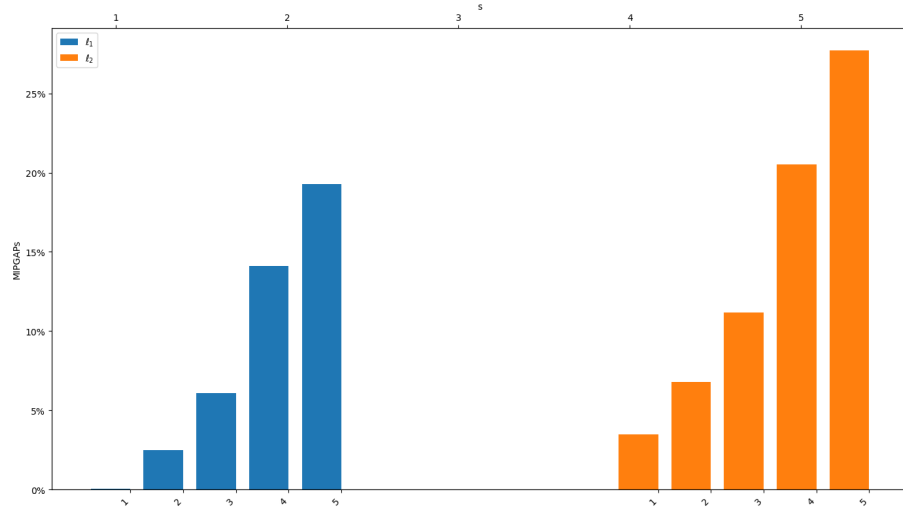


Figure 4.16: Graphical representation of MIPGAPs for the continuous approaches.

Moreover, these results underscore the practical advantages of the discrete method. The ability to approximate continuous solutions with minimal deviation, coupled with the notable reduction in computational effort, makes the discrete approach highly effective for large-scale instances. These features, combined with its efficiency, demonstrate that the discrete method not only achieves comparable objective values to the continuous ℓ_1 approach but also does so in a way that is more adaptable to real-world engineering and design challenges.

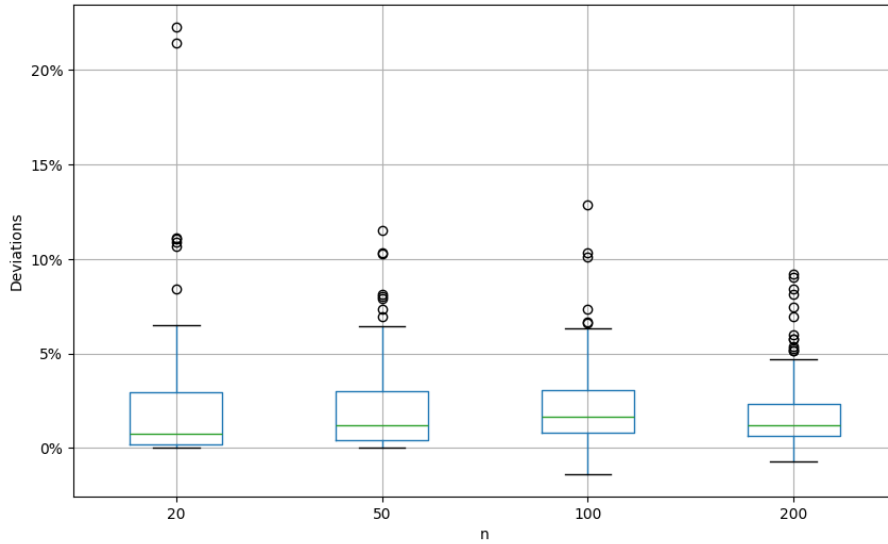


Figure 4.17: Graphical comparison of the objective functions of both models: the Discrete model with grid graph and the continuous model with the ℓ_1 norm.

To summarize, the continuous model with ℓ_2 distance produces the shortest trees but is also the slowest, especially for larger instances. The continuous model with ℓ_1

distance performs well for small to mid-sized instances but also struggles with larger instances. The discrete model is the most efficient in terms of computational time, validating its theoretical and practical strengths. For large instances, all models face challenges with computational time, indicating a need for further optimization or alternative approaches to handle large-scale problems efficiently.

4.7 Conclusions

In this chapter, we introduce a new problem, the FTTNP, motivated by its various industrial applications. We provide a general framework for the problem and study two versions of practical interest: the continuous and the discrete-domain frameworks. We propose mathematical optimization approaches to tackle both versions of the problem. Our approach combines different elements, from the transformation of the given instance to the analysis of the problem as a minimum-length Steiner tree problem. For the continuous case, we develop a formulation that can be rewritten as a Mixed Integer Second Order Cone program, along with simplifications for specific instances of interest. For the discrete FTTNP, we derive a mixed-integer linear programming approach.

Building on this framework, we introduce the Wiring Diagram Problem (WDP), a relevant extension of the discrete-domain FTTNP inspired by real industrial layout applications. In the WDP, the goal is to design spatially feasible, cost-efficient wiring diagrams over complex 3D domains, subject to constructibility, safety, and spatial constraints. We model the problem as a forest of rooted, directed trees, where nodes are constrained to lie in prescribed 3D regions, and arcs represent cable segments. A tailored graph-based discretization of the 3D space enables the formulation of an integer linear programming model that integrates flow-based variables, placement constraints, and detailed geometric feasibility rules. This formulation significantly generalizes the original FTTNP framework and provides a basis for handling more intricate and realistic routing configurations.

Several extensions remain open for future research. On the one hand, many pipeline and cable routing problems require explicit modeling of bends or *elbows*, which may carry installation or maintenance costs. In Chapter 2, the authors propose a graph transformation and modified objective that could be adapted to this context; its integration and the resulting complexity implications warrant further study. On the other hand, as both FTTNP and WDP may lead to large-scale instances, the development of heuristic and decomposition-based methods is advisable. Strengthening the formulations through valid inequalities, dimensionality reduction, and problem-specific relaxations are promising directions for achieving computational scalability in practical settings.

Chapter 5

Identifying Self-Amplifying Hypergraph Structures through Mathematical Optimization

In this chapter, we introduce the concept of *self-amplifying structures* within the framework of hypergraphs, positioning it as a key element for understanding propagation and internal reinforcement in complex systems. To quantify this phenomenon, we define the maximal amplification factor, a metric that captures how effectively a subhypergraph contributes to its own expansion. We then develop an optimization-based methodology to compute this measure. Building on this foundation, we tackle the problem of identifying the subhypergraph that maximizes the amplification factor, formulating it as a mixed-integer nonlinear programming (MINLP) problem. To solve it efficiently, we propose an exact iterative algorithm with proven convergence guarantees. Finally, we report the results of extensive computational experiments on realistic synthetic instances, demonstrating both the relevance and the effectiveness of the proposed approach.

5.1 Introduction

Networks are combinatorial structures composed of nodes and arcs (or edges) that represent the interactions between entities. Typically, nodes model the components of a system, while arcs capture their relationships. Modeling systems as networks has proven essential for uncovering structural and dynamical properties across a wide range of domains. Over time, networks have become a central object of study in both theoretical and applied mathematics, with significant advances from multiple perspectives, including algebraic spectral graph theory [Chung, 1997], geometric graph embeddings [Heinonen, 2003], combinatorial graph theory [Harris, 2008], arithmetic graph theory [Acharya and Hegde, 1990], and combinatorial optimization [Cook et al., 1994]. This extensive toolkit has rendered graph theory indispensable for modeling, analyzing, and optimizing complex systems.

However, a key limitation of classical network models is their inherently pairwise nature: each arc connects exactly two nodes. While this suffices for many applications—such as social networks, where an arc may indicate a "follows" rela-

tionship between users, or logistics networks, where arcs represent transportation routes—many real-world interactions involve more complex, multi-way relationships.

Hypergraphs provide a natural generalization of networks by allowing edges (hyperarcs) to connect multiple nodes simultaneously. This extended framework has recently proven crucial in modeling higher-order interactions across disciplines. For example, in biology and chemistry, reactions often involve multiple reactants and products [Andersen et al.]; in manufacturing and process engineering, multiple input materials may be transformed into several outputs [Nagy et al., 2022]; and in multi-agent systems, tasks may require coordinated actions from several agents to produce multiple outcomes [Wang et al., 2024]. A sophisticated application of hypergraph encoding includes also complex model representation Michelen and Palambros [1997]. Despite their expressiveness, hypergraphs are still comparatively underexplored—particularly when integrated into combinatorial optimization models. Moreover, while many results in graph theory extend naturally to hypergraphs, others break down completely. For instance, the minimum edge cover problem, solvable in polynomial time for graphs [Garey and Johnson, 1979], becomes NP-hard in hypergraphs [Repisky et al., 2007]. Similar complexity jumps occur for maximum-cardinality matching [Berman, 2003], the minimum vertex cover problem [Guruswami and Saket, 2010], and even shortest path computations in directed hypergraphs [Ausiello and Laura, 2017]. The situation becomes even more intricate in directed or multiset hypergraphs, where many foundational questions remain open. Thus, while hypergraphs offer greater modeling power, they also present new challenges and opportunities in the development of combinatorial optimization tools.

In this work, we investigate a combinatorial optimization problem on hypergraphs that arises in the analysis of propagation processes in dynamical systems. In large hypergraphs—used to model temporal evolution—understanding system dynamics involves identifying the key elements and interactions that drive change. This task is particularly challenging in real-world networks due to their complexity and scale. For instance, in biochemical networks, detecting the specific species or molecules responsible for evolutionary transitions can help answer questions related to the *Origin of Life*—that is, which molecules, and under what conditions, catalyzed the emergence of life [Hordijk and Steel, 2018, 2004, Hordijk et al.]. Similarly, in social networks, identifying the most influential users and their interactions helps advertisers effectively target and amplify their campaigns. In manufacturing, understanding how certain inputs contribute disproportionately to output can guide resource allocation and pricing strategies.

We introduce the notion of a *self-amplifying subhypergraph* in the context of multi-directed hypergraphs. A self-amplifying subhypergraph comprises a subset of nodes and hyperarcs from a larger hypergraph, along with a distinguished set of nodes that appear both as inputs and outputs—capturing the idea of "self." This substructure satisfies an amplification property: through appropriate combinations of the included hyperarcs, the net production of the distinguished nodes exceeds their net consumption. This captures the essence of a subsystem capable of reinforcing or expanding itself via internal interactions. While similar notions have appeared in domains such as chemical reaction networks (e.g., autocatalytic sets) and production economics (e.g., Von Neumann growth models), our formulation generalizes these

ideas to arbitrary multi-directed hypergraphs.

Enumerating all such substructures is computationally intractable. As shown by Gagrani et al. [2024], even the number of inclusion-minimal self-amplifying subhypergraphs can grow exponentially with the number of nodes. To overcome this, we propose a quantitative measure inspired by von Neumann [1945] to evaluate the expansion capacity of a subhypergraph. Specifically, we define the *amplification factor* of a subhypergraph, given a set of intensity weights on the hyperedges, as the minimum ratio—across all involved nodes—between the total output and the total input of each node under those intensities. We then define the *maximal amplification factor* (MAF) as the highest such ratio attainable by optimizing the edge weights. While this index can be computed for any subhypergraph, we establish a tight connection between the MAF and the notion of self-amplification: we derive necessary and sufficient conditions under which a subhypergraph is self-amplifying based on its MAF. This provides a practical method for identifying self-reinforcing subsystems without exhaustive enumeration.

To compute the MAF, we develop a nonlinear optimization model that falls into the class of generalized fractional programs [Barros et al., 1996, Blanco et al., 2013, Crouzeix and Ferland, 1991], which are closely related to the well-known generalized eigenvalue problem Nishioka et al. [2025]. These problems are typically nonconvex and computationally demanding. To address this, we design an exact iterative algorithm with guaranteed convergence properties.

We then extend our analysis to a higher-level combinatorial problem: identifying, within a given hypergraph, the subhypergraph that maximizes the amplification factor. Solving this problem enables the discovery of the most influential substructures—those most capable of driving system-wide propagation or growth. We formulate this as a mixed-integer nonlinear programming (MINLP) problem, using binary decision variables to encode the selection of nodes and arcs. This model builds on the MAF computation framework, with additional constraints to manage the combinatorial complexity. Furthermore, we introduce application-driven structural constraints to guide the search toward interpretable and practically relevant substructures.

The remainder of this chapter is organized as follows. In Section 5.2, we introduce the necessary notation and formally define the MAF in multi-directed hypergraphs. Section 5.3 presents the optimization framework for computing the MAF of a fixed subhypergraph. In Section 5.4, we extend the model to identify optimal subhypergraphs. In Section 5.5 we add some features to our models that allow us to add specific constraints tailored to particular scenarios. Section 5.6 reports computational results on synthetic datasets simulating chemical reaction networks, illustrating the effectiveness and limitations of our approach. Finally, Section 5.7 summarizes our contributions and outlines directions for future research.

5.2 Preliminaries

In this section, we introduce the notation and core definitions used throughout the chapter.

The primary mathematical object underlying our work is the directed hypergraph, which generalizes the concept of (di)graphs by allowing hyperedges to connect multiple nodes in a directed fashion. Unlike standard graphs, where edges link pairs of nodes, directed hypergraphs enable richer modeling of complex relationships by connecting sets of nodes to other sets of nodes. While various definitions of directed hypergraphs exist in the literature, we adopt the most commonly used formulation.

Definition 5.2.1. *A directed hypergraph is a tuple $\mathcal{H} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of hyperarcs, where each hyperarc $a \in \mathcal{A}$ is represented as an ordered pair $a = (S_a, T_a)$, with $\emptyset \neq S_a, T_a \subseteq \mathcal{N}$, the **source** and **target** sets of hyperarc a , respectively.*

Observe that when both the source and target sets of every hyperarc contain exactly one node (i.e., $|S_a| = |T_a| = 1$ for all $a \in \mathcal{A}$), the structure reduces to a standard directed graph. However, directed hypergraphs offer a more flexible modeling framework, particularly suitable for representing systems involving higher-order dependencies that cannot be captured by digraphs alone. A prominent example arises in production planning, where technologies (represented by hyperarcs) may require multiple input materials and generate multiple output products, often with shared inputs or outputs across technologies. In this context, S_a denotes the set of input nodes, and T_a the set of output nodes associated with hyperarc a .

A further generalization is the multihypergraph, which allows multiple hyperarcs to exist between the same source and target sets. This multiplicity enables modeling scenarios where several instances of the same transformation occur, or where a hyperarc consumes or produces multiple units of a given product. Such modeling is essential in input-output production systems, where technologies may consume or generate multiple units of specific goods. While multihypergraphs can theoretically be simplified into standard hypergraphs, this often results in the loss of valuable structural information—information that can be crucial for analysis and decision-making [Klamt et al., 2009, Battiston et al., 2021, Bick et al., 2023].

Next, we define the input and output incidence matrices of a hypergraph:

Definition 5.2.2. *Let $\mathbb{S}$ (respectively, $\mathbb{T}$) denote the **input** (respectively, **output**) incidence matrix of the hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{A})$. For a hyperarc $a \in \mathcal{A}$ and a node $v \in S_a$ (respectively, $v \in T_a$), the entry $\mathbb{S}_{va}$ (respectively, $\mathbb{T}_{va}$) corresponds to the multiplicity of v in the multiset S_a (respectively, T_a), that is, $\mathbb{S}_{va} = |v|$ in S_a (respectively, $\mathbb{T}_{va} = |v|$ in T_a).*

Figure 5.1 shows a simple multi-directed hypergraph with nodes $\mathcal{N} = \{A, B, C, D, E\}$ and two hyperarcs: $a_1 = (\{A, A, B\}, \{C\})$ and $a_2 = (\{C\}, \{D, D, D, E, E\})$. The corresponding input and output incidence matrices are:

$$\mathbb{S} = \begin{pmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \mathbb{T} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 3 \\ 0 & 2 \end{pmatrix}.$$

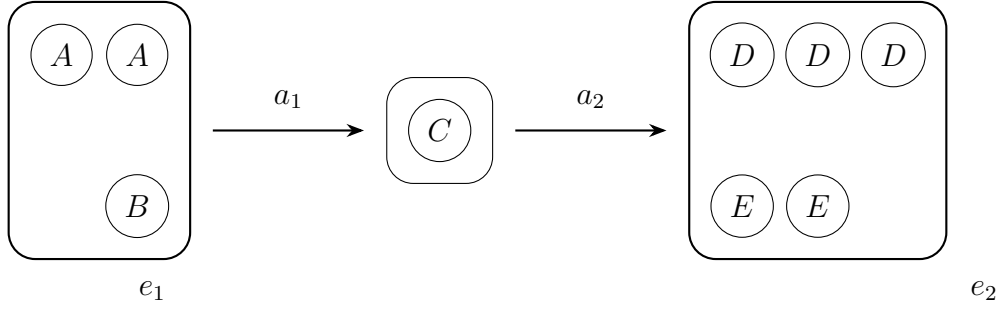


Figure 5.1: Illustration of a simple multi-directed hypergraph with two hyperarcs on the node set $\{A, B, C, D, E\}$.

Definition 5.2.3. The *net incidence matrix* of the hypergraph is defined as the difference between the output and input matrices: $\mathbb{Q} = \mathbb{T} - \mathbb{S}$.

In the example above, the net matrix is:

$$\mathbb{Q} = \begin{pmatrix} -2 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & 3 \\ 0 & 2 \end{pmatrix}.$$

Negative entries in $\mathbb{Q}$ indicate nodes that act as sources, while positive entries denote nodes that act as targets. If a node appears in both the source and target of a hyperarc with the same multiplicity, the corresponding entry in $\mathbb{Q}$ is zero, reflecting a net-neutral contribution. More on visualization and the graph-theoretical interpretation of such hypergraphs can be found in [Kraakman and Stegehuis, 2024].

Each nonzero nonnegative vector $\mathbf{x} \in \mathbb{R}_+^{\mathcal{A}} \setminus \{\mathbf{0}\}$ represents an *intensity profile* that linearly combines the hyperarcs. The vectors $\mathbb{S}\mathbf{x}$ and $\mathbb{T}\mathbf{x}$ represent the aggregate input and output of nodes under this combination. The net aggregate is given by: $\mathbb{Q}\mathbf{x} = (\mathbb{T} - \mathbb{S})\mathbf{x}$, which indicates the overall balance for each node. A negative value for a node indicates it is being depleted (a *drain node*), while a positive value indicates it is being produced more than consumed (an *amplifying node*).

In the example from Figure 5.1, for the intensity vector $\mathbf{x} = (2, 3)$, we obtain: $\mathbb{Q}\mathbf{x} = (-4, -2, -1, 9, 6)^\top$. Thus, nodes A , B , and C act as drains, while D and E are amplifying.

Definition 5.2.4. A hypergraph $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$ is called a subhypergraph of $\mathcal{H} = (\mathcal{N}, \mathcal{A})$, denoted by $\mathcal{H}' \subseteq \mathcal{H}$, if $\mathcal{A}' \subseteq \mathcal{A}$ and all nodes appearing in the input or output of hyperarcs in $\mathcal{A}'$ belong to $\mathcal{N}'$, i.e., for every $a = (\mathcal{S}_a, \mathcal{T}_a) \in \mathcal{A}'$, we have $\mathcal{S}_a, \mathcal{T}_a \subseteq \mathcal{N}'$.

Given a subhypergraph $\mathcal{H}' \subseteq \mathcal{H}$ and a subset $\mathcal{M} \subseteq \mathcal{N}'$, the subhypergraph of $\mathcal{H}'$ induced by the nodes in $\mathcal{M}$ is called the *$\mathcal{M}$ -restricted subhypergraph* and is denoted by $\mathcal{H}'|_{\mathcal{M}}$. A central concept in this work is that of a self-amplifying subhypergraph, defined below.

Definition 5.2.5 (Self-Amplifying Subhypergraph). Let $\mathcal{H}$ be a hypergraph and $\mathcal{H}' = (\mathcal{N}', \mathcal{A}') \subseteq \mathcal{H}$. We say that $\mathcal{H}'$ is **self-amplifying** with respect to a node set $\mathcal{M} \subseteq \mathcal{N}'$ if the following conditions are satisfied:

1. **Self-sufficiency:** Each hyperarc in $\mathcal{A}'$ must have at least one input and one output node in $\mathcal{M}$, and every node in $\mathcal{M}$ must appear at least once in the source and once in the target of some hyperarc in $\mathcal{A}'$:

$$\forall a \in \mathcal{A}', \exists v, v' \in \mathcal{M}: \quad \mathbb{T}_{va} > 0, \quad \mathbb{S}_{v'a} > 0, \quad (A^a)$$

$$\forall v \in \mathcal{M}, \exists a, a' \in \mathcal{A}': \quad \mathbb{T}_{va} > 0, \quad \mathbb{S}_{va'} > 0. \quad (A^v)$$

2. **Net-positive realizability:** There exists a strictly positive intensity vector $\mathbf{x} \in \mathbb{R}_{>0}^{\mathcal{A}'}$ such that the net aggregate for every node in $\mathcal{M}$ is strictly positive:

$$\sum_{a \in \mathcal{A}'} \mathbb{Q}_{va} x_a > 0, \quad \forall v \in \mathcal{M}.$$

These notions are closely related to autonomy and productivity concepts studied in the context of Chemical Reaction Networks [Blokhuys et al., 2020, Gagrani et al., 2024].

For a self-amplifying subhypergraph over $\mathcal{M}$, the set $\mathcal{M}$ is referred to as the set of *self-amplifying nodes*. The $\mathcal{M}$ -restricted subhypergraph is said to be **minimally self-amplifying** if it does not contain a proper self-amplifying subhypergraph. The matrix $\mathbb{Q}|_{\mathcal{M}, \cdot}$ associated with such a minimal subhypergraph is called an *amplification submatrix*.

Figure 5.2 illustrates a hypergraph represented as a bipartite graph, where square nodes correspond to hypergraph nodes and circular nodes represent hyperarcs. For instance, the topmost circular node corresponds to a hyperedge that consumes three instances of node A and produces one instance of node A and one instance of node B. This is an example of a minimally self-amplifying hypergraph since:

1. It is a self-sufficient hypergraph, as every node appears in both the source and target sets of at least one hyperarc.
2. It is also net-positive realizable, as there exists a intensity vector $\mathbf{x} = (0, 0, 1)^\top$ for which:

$$\mathbb{Q}\mathbf{x} = \begin{pmatrix} -2 & -1 & 3 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ -1 \end{pmatrix},$$

$$\text{and } \sum_{a \in \mathcal{A}} \mathbb{Q}_{va} x_a = 2 > 0.$$

3. It does not contain any other self-amplifying subhypergraph.

This is an example of a self-sufficient hypergraph, as every node appears in both the source and target sets of at least one hyperarc. Moreover, it is minimally self-amplifying, as it is also net-positive realizable and does not contain any self-amplifying subhypergraph.

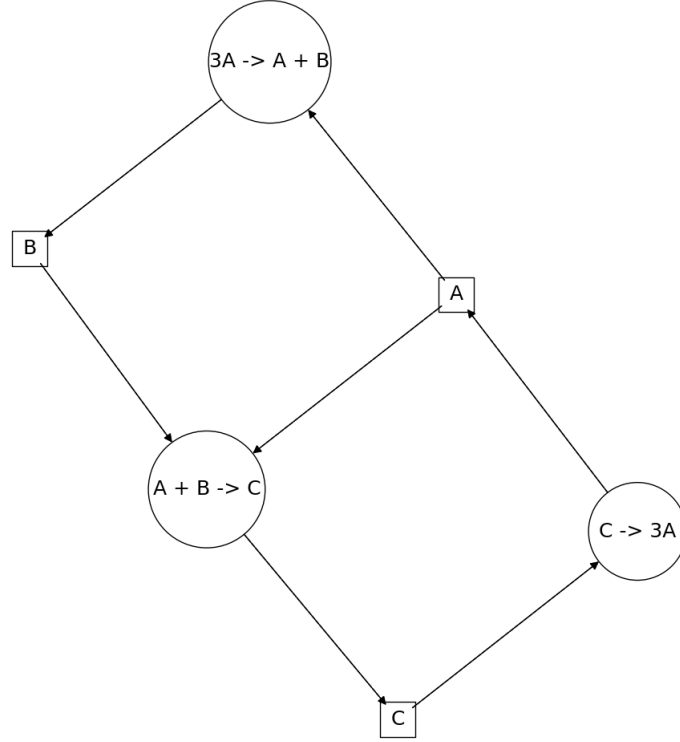


Figure 5.2: A hypergraph represented as a bipartite graph.

Multi-directed subhypergraphs can simulate dynamical systems that evolve over discrete time. Let $\mathcal{H}$ be a hypergraph, and let $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$ be a subhypergraph that evolves through a sequence of time-indexed intensity vectors $\mathbf{x}^t \geq 0$, for $t \in \mathbb{Z}_+$, applied to the hyperarcs in $\mathcal{H}'$.

Suppose that the nodes in a subset $\mathcal{M} \subseteq \mathcal{N}'$ are positively amplified over time. This means that, for every $v \in \mathcal{M}$ and every $a \in \mathcal{A}'$, we have:

$$\mathbb{T}_{va}\mathbf{x}_a^t \geq \mathbb{S}_{va}\mathbf{x}_a^t.$$

where here $\geq$ indicates the componentwise order in $\mathbb{R}^{|\mathcal{M}|}$.

Let $\mathbb{S}_{\mathcal{M}}$ and $\mathbb{T}_{\mathcal{M}}$ denote the submatrices of $\mathbb{S}$ and $\mathbb{T}$ (row-restricted to the nodes in $\mathcal{M}$). Considering that the output of nodes at time t becomes the input at time $t + 1$, the following inequality holds:

$$\mathbb{T}_{\mathcal{M}}\mathbf{x}^t \geq \mathbb{S}_{\mathcal{M}}\mathbf{x}^{t+1}. \quad (1)$$

If we further assume that the intensity at time $t + 1$ is a scalar multiple of the intensity at time t , i.e., $\mathbf{x}^{t+1} = \alpha\mathbf{x}^t$, thereby preserving the system's structure, we obtain:

$$\mathbb{T}_{\mathcal{M}}\mathbf{x}^t \geq \alpha\mathbb{S}_{\mathcal{M}}\mathbf{x}^t.$$

We then say that the subhypergraph $\mathcal{H}'$ with initial intensity $\mathbf{x}^0$ *amplifies* if $\alpha \geq 1$, and *contracts* if $\alpha < 1$.

The focus of this chapter is the analysis of how to determine initial intensity vectors that induce maximal amplification within the system. This leads to the following key definition:

Definition 5.2.6 (Amplification Factor). *Let $\mathcal{H} = (\mathcal{N}, \mathcal{A})$ be a hypergraph, let $\mathcal{H}' = (\mathcal{N}', \mathcal{A}') \subseteq \mathcal{H}$ be a subhypergraph, and let $\emptyset \neq \mathcal{M} \subset \mathcal{N}'$. Given an intensity vector $\mathbf{x} \in \mathbb{R}_{>0}^A$, the amplification factor of $\mathcal{H}'$ with respect to $\mathbf{x}$ is defined as:*

$$\alpha(\mathcal{H}', \mathcal{M}; \mathbf{x}) = \inf_{v \in \mathcal{M}} \frac{\sum_{a \in \mathcal{A}'} \mathbb{T}_{va} \mathbf{x}_a}{\sum_{a \in \mathcal{A}'} \mathbb{S}_{va} \mathbf{x}_a}.$$

This value represents the smallest expansion rate achieved by any node in $\mathcal{M}$ under the intensity vector $\mathbf{x}$. Since $\mathbf{x} \neq \mathbf{0}$, and $\mathbb{S}$ is a nonzero, nonnegative integer matrix, the infimum above can be replaced by a minimum.

*The **Maximal Amplification Factor** (MAF) of $\mathcal{H}'$ over $\mathcal{M}$ is the maximum amplification factor attainable over all strictly positive flows $\mathbf{x} \in \mathbb{R}_{>0}^A$:*

$$\alpha(\mathcal{H}', \mathcal{M}) = \max_{\mathbf{x} \in \mathbb{R}_{>0}^A} \alpha(\mathcal{H}', \mathcal{M}; \mathbf{x}).$$

A key contribution of this chapter is the development of computational strategies for analyzing expanding hypergraphs using this metric. The following result establishes a foundational link between the amplification factor and the concept of self-amplifying subhypergraphs (Definition 5.2.5). In particular, we show that the amplification factor provides a certificate for identifying self-amplifying structures.

Lemma 6. *If $\mathcal{H}'$ is self-sufficient in $\mathcal{M}$, then $0 < \alpha(\mathcal{H}', \mathcal{M}) < \infty$.*

Proof. Self-sufficiency of $\mathcal{H}'$ in $\mathcal{M}$ implies that each node in $\mathcal{M}$ is both produced and consumed by at least one hyperarc in $\mathcal{A}'$. Under these conditions, Gale [1989, Theorem 9.8] guarantees the existence of a strictly positive amplification factor α . Since every hyperarc consumes at least one node, the denominator $\sum_{a \in \mathcal{A}'} \mathbb{S}_{va} \mathbf{x}_a > 0$ for any intensity vector $\mathbf{x}$, ensuring that $\alpha(\mathcal{H}', \mathcal{M}) < \infty$. Similarly, because each node is an output of at least one hyperarc, the numerator $\sum_{a \in \mathcal{A}'} \mathbb{T}_{va} \mathbf{x}_a > 0$, guaranteeing $\alpha(\mathcal{H}', \mathcal{M}) > 0$. $\square$

Lemma 7. *The subhypergraph $\mathcal{H}'$ is net-positive realizable in $\mathcal{M}$ if and only if $\alpha(\mathcal{H}', \mathcal{M}) > 1$.*

Proof. The result follows directly from the definition. If $\mathcal{H}'$ is net-positive realizable in $\mathcal{M}$, then there exists an intensity vector $\mathbf{x}$ such that $\sum_{a \in \mathcal{A}'} \mathbb{T}_{va} \mathbf{x}_a > \sum_{a \in \mathcal{A}'} \mathbb{S}_{va} \mathbf{x}_a$ for all $v \in \mathcal{M}$, which implies $\alpha(\mathcal{H}', \mathcal{M}; \mathbf{x}) > 1$. Conversely, if $\alpha(\mathcal{H}', \mathcal{M}) > 1$, then such an $\mathbf{x}$ exists and the inequality is satisfied. $\square$

Lemma 8. *If $0 < \alpha(\mathcal{H}', \mathcal{M}) < \infty$, then $\mathcal{H}'$ is entity-self-sufficient.*

Proof. Since $\mathbb{S}$ is nonnegative, and for each $v \in \mathcal{M}$ we have that $\sum_{a \in \mathcal{A}'} \mathbb{T}_{va} \mathbf{x}_a > 0$ and $\sum_{a \in \mathcal{A}'} \mathbb{S}_{va} \mathbf{x}_a > 0$, there must exist at least $a, a' \in \mathcal{A}'$ such that $\mathbb{S}_{va} > 0$ and $\mathbb{T}_{va'} > 0$ (entity self-sufficiency).

On the other hand, for all $a \in \mathcal{A}'$ such that $\mathbb{T}_{va} = 0$ for all $v \in \mathcal{M}$, due to the maximization criterion, the optimal value would always be obtained when $x_a = 0$ for these hyperarcs (which is equivalent to excluding them from the subnetwork).

Let $Q = \{a \in \mathcal{A}' : \mathbb{S}_{va} = 0, \forall v \in \mathcal{M}\}$, then the expression for the amplification factor for a given $\mathbf{x}$ is:

$$\inf_{v \in \mathcal{M}} \frac{\sum_{a' \in \mathcal{A}'} \mathbb{T}_{va'} \mathbf{x}_{a'}}{\sum_{a' \in \mathcal{A}'} \mathbb{S}_{va'} \mathbf{x}_{a'}} = \inf_{v \in \mathcal{M}} \left(\frac{\sum_{a' \in \mathcal{A}' \setminus Q} \mathbb{T}_{va'} \mathbf{x}_{a'}}{\sum_{a' \in \mathcal{A}' \setminus Q} \mathbb{S}_{va'} \mathbf{x}_{a'}} + \frac{\sum_{a \in Q} \mathbb{T}_{va} \mathbf{x}_a}{\sum_{a' \in \mathcal{A}' \setminus Q} \mathbb{S}_{va'} \mathbf{x}_{a'}} \right).$$

Since the term $\sum_{a \in Q} \mathbb{T}_{va} \mathbf{x}_a \geq 0$ only appears in the numerator of the expression, when maximizing over $\mathbf{x}$, the solution is reached for values $x_a = \infty$ for $a \in Q$, leading to $\alpha(\mathcal{H}', \mathcal{M}) = \infty$, contradicting the assumption that the amplification factor is finite. $\square$

Combining the above results, we arrive at the central connection between the concepts introduced in this work.

Theorem 9. *Let $\mathcal{H} = (\mathcal{N}, \mathcal{A})$ be a hypergraph, $\mathcal{H}' = (\mathcal{N}', \mathcal{A}') \subseteq \mathcal{H}$ a subhypergraph, and $\emptyset \neq \mathcal{M} \subset \mathcal{N}'$. Then:*

1. *If $\mathcal{H}'$ is self-amplifying over $\mathcal{M}$, then $1 < \alpha(\mathcal{H}', \mathcal{M}) < \infty$.*
2. *Conversely, if $\mathcal{H}'$ is self-sufficient (i.e., $0 < \alpha(\mathcal{H}', \mathcal{M}) < \infty$) and net-positive realizable (i.e., $\alpha(\mathcal{H}', \mathcal{M}) > 1$), then $\mathcal{H}'$ is self-amplifying in $\mathcal{M}$.*

5.3 Computing the Maximal Amplification Factor

In this section we describe an algorithmic approach to exactly compute the MAF of a given subhypergraph. Note that this rate cannot be directly evaluated since it requires determining the intensity vector reaching the higher amplification factor. Here, we provide an optimization-based framework for its computation.

The MAF of a subnetwork $\mathcal{H}'$ can be formulated as the following mathematical optimization problem:

$$\begin{aligned} \alpha(\mathcal{H}', \mathcal{M}) &:= \max \alpha \\ \text{s.t. } &\mathbb{T}_{\mathcal{M}} \mathbf{x} \geq \alpha \mathbb{S}_{\mathcal{M}} \mathbf{x}, \\ &\mathbb{S}_{\mathcal{M}} \mathbf{x} \geq 1, \\ &\mathbf{x} \in \mathbb{R}_+^{\mathcal{A}'}, \alpha \geq 0. \end{aligned} \tag{MAF}$$

In the above formulation α represents the MAF, whereas $\mathbf{x}$ indicates the intensity vector. The objective function maximizes the amplification factor, which is adequately defined by the two sets of constraints. Observe that the constraints $\mathbb{T}_{\mathcal{M}} \geq \alpha \mathbb{S}_{\mathcal{M}} \mathbf{x}$ enforce α to be properly defined. The constraint $\mathbb{S}_{\mathcal{M}} \mathbf{x} \geq \mathbf{1}$, together with $\mathbf{x} \in \mathbb{R}_+^{|\mathcal{A}'|}$ is equivalent to assuring that $\mathbf{x} \neq \mathbf{0}$.

Note that the problem above can be interpreted as a robust choice of intensities that maximizes the minimum amplification factor among the nodes of the subhypergraph.

The problem (MAF) is neither convex nor concave, and thus, standard convex optimization tools are not directly applicable. However, the MAF can also be formulated as a generalized fractional programming problem, which is computationally challenging (e.g., Barros et al. [1996], Blanco et al. [2013], Crouzeix and Ferland [1991]). To solve (MAF), we adapt a Dinkelbach-type procedure (e.g., Crouzeix and Ferland [1991]), which iteratively computes $\alpha(\mathcal{H}', \mathcal{M})$ and guarantees convergence to the actual value.

In what follows, we analyze the computation of the MAF for a given subhypergraph, $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$, and a given subset $\mathcal{M} \subseteq \mathcal{N}'$. A specific instance of this occurs when $\mathcal{N}' = \mathcal{M}$, meaning all nodes in the subhypergraph are amplification factor-inducing nodes.

The algorithm that we propose for computing the MAF is based on solving exactly the mathematical optimization model in (MAF) by applying an ad-hoc Dinkelbach-type procedure. Specifically, we begin by initializing the intensity vector to $\mathbf{x}_0$. For each intensity vector obtained during the procedure, we compute the amplification factor induced by that intensity, denoted as α_{it} . Next, we solve the following auxiliary linear program to compute the optimal values ρ and $\mathbf{x}_{\text{it}}$:

$$\begin{aligned} (\rho^*, \mathbf{x}_{\text{it}}^*) &\in \arg \max \rho \\ \text{s.t. } \rho &\leq \mathbb{T}_{\mathcal{M}} \mathbf{x} - \alpha_{\text{it}} \mathbb{S}_{\mathcal{M}} \mathbf{x}, \\ \mathbb{S}_{\mathcal{M}} \mathbf{x} &\geq \mathbf{1}, \\ \mathbf{x} &\in \mathbb{R}_+^{|\mathcal{A}'|}, \rho \in \mathbb{R}_+. \end{aligned}$$

In case $\rho^* = 0$, the obtained solution, $(\alpha_{\text{it}}, \mathbf{x})$ is optimal, and the amplification factor for this intensity vector, α_{it} , corresponds to the MAF. Otherwise, the intensity vector is updated based on the solution of the linear program, $\mathbf{x}_{\text{it}}$, and the process is repeated.

A pseudocode for our proposed algorithm is detailed in Algorithm 5.

The convergence of this procedure described in Algorithm 5 is guaranteed by Crouzeix et al. [1985].

Theorem 10. *If the feasible intensity vectors $\mathbf{x}$ are upper bounded, Algorithm 5 converges linearly (in terms of the number of iterations) to the optimal solution.*

Proof. The proof follows by applying the results in Crouzeix et al. [1985], considering that the MAF algorithm we propose satisfies the hypotheses for their application. Specifically, it is not difficult to verify that the feasible regions of the problems under study are compact and that the functions involved in the quotients defining the MAF are linear (and hence continuous). $\square$

Algorithm 5: Computation of the amplification factor of subnetwork $\mathcal{H}'$ with a given set of self-amplifying entities $\mathcal{M}$.

input : $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$, $\mathcal{M} \subset \mathcal{N}'$, $\text{it} = 0$, and $\mathbf{x}_0 \in \mathbb{R}_+^{|\mathcal{A}'|}$.
output: The amplifying factor $\alpha(\mathcal{H}', \mathcal{M})$
 Stop=False
while $\text{Stop}=False$ **do**
 Define $\alpha_{\text{it}}^t = \alpha(\mathbf{x}_{\text{it}})$ and solve the linear program:
 $(\rho, \bar{\mathbf{x}}_{\text{it}}) \in \arg \max \rho$
 s.t. $\rho \leq \mathbb{T}_{\mathcal{M}}\mathbf{x} - \alpha_{\text{it}}\mathbb{S}_{\mathcal{M}}\mathbf{x}$,
 $\mathbb{S}_{\mathcal{M}}\mathbf{x} \geq \mathbf{1}$,
 $\mathbf{x} \in \mathbb{R}_+^{|\mathcal{A}'|}, \rho \in \mathbb{R}_+$.
 if $\rho = 0$ **then**
 Stop=True,
 $\alpha(\mathcal{H}', \mathcal{M}) = \alpha_{\text{it}}$
 else
 $\mathbf{x}_{\text{it}+1} = \bar{\mathbf{x}}_{\text{it}}$,
 $\text{it} = \text{it} + 1$

5.4 Optimal Subhypergraphs under maximal amplifying factors

The main methodological contribution of this chapter is the mathematical model we propose to construct self-sufficient subnetworks that maximize the amplification factor. Note that, if the set $\mathcal{M}$ is either unknown or not all nodes in the sub-hypergraph satisfy the conditions for amplification induction, determining this set will become an integral part of the decision-making process for computing the amplification factor. In this case, both the subhypergraph $\mathcal{H}'$ and the set $\mathcal{M}$ must be constructed as part of the procedure. In this section, we detail the methodology that we develop for computing the subhypergraph that maximizes the amplifying factor.

Given a hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{A})$, our goal is to construct a subnetwork $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$ and a set $\mathcal{M} \subset \mathcal{N}$ that maximizes the amplification factor. By Theorem 9, if the obtained subnetwork has an amplification factor greater than or equal to one, it is considered self-amplifying, and $\mathcal{M}$ would be a proper self-amplifying set. Otherwise, the subnetwork with the highest amplification factor is computed, but $\mathcal{M}$ would merely be a self-sufficient set without net-positive realizability.

We propose a mathematical optimization approach for constructing a subnetwork within a given hypergraph. The proposed mathematical program uses the following decision variables, corresponding to the four main decisions to be made:

$$y_v = \begin{cases} 1 & \text{if node } v \text{ is in } \mathcal{M} \\ 0 & \text{otherwise} \end{cases}, \quad \forall v \in \mathcal{N}, \quad z_a = \begin{cases} 1 & \text{if } a \text{ is in } \mathcal{A}' \\ 0 & \text{otherwise} \end{cases} \quad \forall a \in \mathcal{A}.$$

$\mathbf{x} \in \mathbb{R}_+^{|\mathcal{A}|}$: intensity vector, $\alpha \in \mathbb{R}_+$: amplification factor.

These variables determine which nodes belong to the distinguished subset $\mathcal{M}(y)$, which hyperarcs are selected in $\mathcal{H}'(z)$, the amplification factor of the subnetwork (α), and the intensity vector ensuring the amplification factor ($\mathbf{x}$).

These variables are combined through the following inequalities to ensure the adequate construction of the desired subnetwork:

- The amplification factor is well-defined:

$$\sum_{a \in \mathcal{A}} \mathbb{T}_{va} z_a \mathbf{x}_a \geq \alpha y_v \sum_{a \in \mathcal{A}} \mathbb{S}_{va} z_a \mathbf{x}_a, \quad \forall v \in \mathcal{N}. \quad (2)$$

Note that in case v is chosen to be part of $\mathcal{M}$ and $a \in \mathcal{A}'$, the above constraints assure that $\mathbb{T}_M \mathbf{x} \geq \alpha \mathbb{S}_M \mathbf{x}$, as required to properly defined the MAF. Otherwise, either $y_v = 0$, in whose case the constraint is redundant ($\sum_{a \in \mathcal{A}} \mathbb{T}_{va} z_a \mathbf{x}_a \geq 0$, which is always true by the nonnegativity of $\mathbb{T}$ and the variables), or $z_a = 0$, in whose case the hyperarc a does not account in the computation of the MAF.

- The subset $\mathcal{M}$ is not empty:

$$\sum_{v \in \mathcal{N}} y_v \geq 1. \quad (3)$$

To assure that the MAF is well defined, at least one node in the subhypergraph must belong to $\mathcal{M}$.

- The denominator in the amplification factor is non zero:

$$\sum_{a \in \mathcal{A}} \mathbb{S}_{va} z_a x_a \geq y_v, \quad \forall v \in \mathcal{M}, \quad (4)$$

ensuring that if a node is selected in $\mathcal{M}$ ($y_v = 1$), its contribution to the amplification factor is strictly positive.

- The nodes in $\mathcal{M}$ must be self-sufficient (A^v):

$$y_v \leq \left(\sum_{\substack{a \in \mathcal{A}: \\ \mathbb{T}_{va} > 0}} z_a \right) \left(\sum_{\substack{a \in \mathcal{A}: \\ \mathbb{S}_{va} > 0}} z_a \right), \quad \forall v \in \mathcal{N}. \quad (5)$$

This condition ensures that nodes in $\mathcal{M}$ appear in at least one selected hyperarc both as a source and as a target. Note that this nonlinear constraint can be adequately linearized as follows:

$$y_v \leq \sum_{\substack{a \in \mathcal{A}: \\ \mathbb{T}_{va} > 0}} z_a \quad (6)$$

$$y_v \leq \sum_{\substack{a \in \mathcal{A}: \\ \mathbb{S}_{va} > 0}} z_a, \quad (7)$$

Thus, in case v is selected to be in $\mathcal{M}$, at least one hyperarc with v as target and at least one are with v as source are enforced to be chosen in $\mathcal{A}'$.

- The hyperarcs in the subnetwork must be self-sufficient ([A^a](#)):

$$z_a \leq \left(\sum_{\substack{v \in \mathcal{N}: \\ \mathbb{S}_{va} > 0}} y_v \right) \left(\sum_{\substack{v \in \mathcal{N}: \\ \mathbb{T}_{va} > 0}} y_v \right), \quad \forall a \in \mathcal{A}. \quad (8)$$

These constraints ensure that a hyperarc is only selected if there are nodes in $\mathcal{M}$ both as inputs and outputs.

These nonlinear constraints can be rewritten as linear constraints as follows:

$$\sum_{\substack{v \in \mathcal{N}: \\ \mathbb{T}_{va} \geq 1}} y_v \geq z_a, \quad \forall a \in \mathcal{A}, \quad (9)$$

$$\sum_{\substack{v \in \mathcal{N}: \\ \mathbb{S}_{va} \geq 1}} y_v \geq z_a, \quad \forall a \in \mathcal{A}. \quad (10)$$

That is, a hyperarc cannot be selected ($z_a = 0$) if it does not connect nodes in $\mathcal{M}$.

- The constructed subnetwork contains at least one hyperarc:

$$\sum_{a \in \mathcal{A}} z_a \geq 1. \quad (11)$$

This condition avoids the trivial solution where no hyperarcs are selected.

The proposed mathematical optimization model is then:

$$\begin{aligned} & \max \alpha \\ & \text{s.t. } (2) - (11), \\ & \quad \mathbf{x} \in \mathbb{R}_+^{\mathcal{A}}, \\ & \quad y \in \{0, 1\}^{\mathcal{N}}, \\ & \quad z \in \{0, 1\}^{\mathcal{A}}, \\ & \quad \alpha \geq 0. \end{aligned}$$

As noted for problem ([MAF](#)), this optimization model is neither convex nor concave. Furthermore, it involves binary variables, increasing its computational complexity. Despite these challenges, we propose several techniques to solve the problem exactly and efficiently.

We develop an extension of Algorithm 5 that allows the selection of the optimal subhypergraph and the set $\mathcal{M}$ as part of the decision. The solution strategy that we propose is an iterative approach that computes the optimal subnetwork in a finite number of steps, with guaranteed convergence to the optimal solution. The details of these algorithms are provided in Algorithm 6.

In the proposed methodology, we initialize the intensity vector to $\mathbf{x}_0$ in iteration $\text{it} = 0$, and compute the amplification factor for this vector for the whole network,

α_{it} . With this value, we solve the auxiliary problem:

$$\begin{aligned} (\rho^*, \bar{\mathbf{x}}_{it}) &\in \arg \max \rho \\ \text{s.t. } \rho &\leq (\mathbb{T}\mathbf{x})_v y_v \mathbf{z} - \alpha_{it} (\mathbb{S}\mathbf{x})_v y_v \mathbf{z}, & \forall v \in \mathcal{N}, \\ \mathbf{x} &\in \mathbb{R}_+^{|\mathcal{A}'|}, \rho \geq 0, \\ (y, z) &\in \mathcal{X} \end{aligned}$$

where $\mathcal{X} = \{(y, z) \in \{0, 1\}^{\mathcal{N}} \times \{0, 1\}^{\mathcal{A}} : (y, z) \text{ verifies (3) -- (11)}\}$. In case $\rho = 0$, it implies that the obtained solution is optimal for the MAF problem, and we are done. Otherwise, the intensity vector is updated to the $\mathbf{x}$ -solution in the above problem, and the process is repeated.

Algorithm 6: Computation of the subnetwork $\mathcal{H}'$ and the set of self-amplifying entities $\mathcal{M}'$ maximizing the amplification factor in a given hypergraph.

input : $\mathcal{H} = (\mathcal{N}, \mathcal{A})$, $it = 0$, and $\mathbf{x}_0 \in \mathbb{R}_+^{|\mathcal{A}'|}$
output: A subnetwork $\mathcal{H}' = (\mathcal{N}', \mathcal{A}') \subseteq \mathcal{H}$ self-sufficient on $\mathcal{M}$ with MAF $\alpha(\mathcal{H}', \mathcal{M})$.

Stop=False

while Stop=False **do**

 Define $\alpha_{it}^t = \alpha(\mathbf{x}_{it})$ and solve the linear program:

$$\begin{aligned} (\rho, \bar{\mathbf{x}}_{it}) &\in \arg \max \rho \\ \text{s.t. } \rho &\leq (\mathbb{T}\mathbf{x})_v y_v \mathbf{z} - \alpha_{it} (\mathbb{S}\mathbf{x})_v y_v \mathbf{z}, & \forall v \in \mathcal{N}, \\ \mathbf{x} &\in \mathbb{R}_+^{|\mathcal{A}'|}, \rho \geq 0, \\ (y, z) &\in \mathcal{X} \end{aligned}$$

if $\rho = 0$ **then**

 Stop=True,

$\mathcal{A}' = \{a \in \mathcal{A} : z_a = 1\}$,

$\mathcal{M} = \{v \in \mathcal{N} : y_v = 1\}$, and

$\alpha(\mathcal{H}', \mathcal{M}) = \alpha_{it}$

else

$\mathbf{x}_{it+1} = \bar{\mathbf{x}}_{it}$

$it = it + 1$

Note that a mixed-integer linear programming problem must be solved at each iteration. Consequently, the complexity of the mathematical optimization method that we propose to extract a subnetwork from a given dynamic system with MAF is highly influenced by the number of variables and constraints involved. In particular, the time required to solve the problem depends on the number of hyperarcs and nodes in the system. To reduce the search space, we derive two strategies that allow us to accelerate the algorithm: a preprocessing procedure that removes nodes and hyperarcs that can never participate in such a subnetwork, and a divide-et-conquer approach to detect independent substructures in the given hypergraph.

Reducing the Set of Potential Nodes and Hyperarcs

The first dimensionality reduction that we propose to solve more efficiently Algorithm (6) consists of pre-processing the nodes and hyperarcs of $\mathcal{H}$ to remove those that will never participate in the optimal subnetwork.

The procedure iteratively removes non-self-sufficient rows and columns from the source matrix M_1 and the target matrix M_2 . Note that those nodes that do not appear as both sourced and targeted by the hyperarcs are not required in our search. Once these nodes are removed from the list, it may happen that some hyperarcs only act as sources or targets but not both, and then they can also be avoided in the list. This process can be iteratively applied until no hyperarcs or nodes are removed.

This procedure is outlined in Algorithm 7,

Algorithm 7: Algorithm that removes all non-self-sufficient rows and columns from the input and output matrix.

```

input :  $M_1, M_2$ 
output:  $M_1, M_2$ 
 $M_1, M_2 \in \mathbb{N}^{m \times n}$ 
 $rows\_to\_check = \{1, \dots, m\}$ 
 $columns\_to\_check = \{1, \dots, n\}$ 
while True do
     $new\_null\_rows = \{\}$ 
     $new\_null\_columns = \{\}$ 
    for  $i$  in  $rows\_to\_check$  do
        if  $\sum_{i=1}^m (M_1)_{ij} < 0$  or  $\sum_{i=1}^m (M_2)_{ij} < 0$  then
             $new\_null\_rows.insert(i)$ 
    for  $j$  in  $columns\_to\_check$  do
        if  $\sum_{j=1}^n (M_1)_{ij} < 0$  or  $\sum_{j=1}^n (M_2)_{ij} < 0$  then
             $new\_null\_columns.insert(j)$ 
    if  $new\_null\_rows = \emptyset$  and  $new\_null\_columns = \emptyset$  then
        break
    for  $row$  in  $new\_null\_rows$  do
        remove  $row$  from  $\bar{M}_1$ 
        remove  $row$  from  $M_2$ 
    for  $column$  in  $new\_null\_columns$  do
        remove  $column$  from  $\bar{M}_1$ 
        remove  $column$  from  $M_2$ 
     $M_1, M_2 \in \mathbb{N}^{m \times n}$ 
     $rows\_to\_check = \{1, \dots, m\}$ 
     $columns\_to\_check = \{1, \dots, n\}$ 
return  $M_1, M_2$ 

```

The procedure begins by initializing all rows and columns for inspection (lines 1-3). It then enters a loop to iteratively detect and remove null rows and columns (lines 4-27). At each iteration, null rows are identified by checking if the sum of entries in a row is zero in both M_1 and M_2 (lines 7-11). Similarly, null columns are detected by examining the column sums in M_1 and M_2 (lines 12-16). If no new null

rows or columns are found, the algorithm terminates (lines 17-19).

When null rows or columns are detected, they are removed from M_1 and M_2 (lines 20-27). The sets of rows and columns to inspect are updated accordingly, ensuring that any newly created null rows or columns are included in subsequent iterations. This process repeats until no further null rows or columns are found, at which point the simplified matrices are returned (line 28).

Splitting the hypergraph in connected components

The second strategy we employ consists of detecting whether the hypergraph contains multiple connected components. If so, the subnetwork with MAF problem can be addressed by applying Algorithm 6 separately to each component, and then selecting the one where the maximum MAF was obtained. Although this approach involves solving multiple instances of the problem, each instance is of smaller dimension, which leads to a significant reduction in overall CPU time.

The correct performance of this strategy is assured by the following result.

Theorem 11. *Let $\mathcal{G}_1 = (\mathcal{N}_1, \mathcal{A}_1), \dots, \mathcal{G}_k = (\mathcal{N}_k, \mathcal{A}_k)$ be the different connected components for a system $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. Let $\mathcal{G}' = (\mathcal{N}', \mathcal{A}')$ be the MAF subnetwork for $\mathcal{G}$, with the set of self-sufficient nodes M . Then:*

$$\alpha(\mathcal{G}) = \max_{l=1, \dots, k} \alpha(\mathcal{G}_l),$$

where $\mathcal{M}_l$ is the optimal set of self-sufficient nodes for the subnetwork $\mathcal{G}_l$, for $l = 1, \dots, k$.

Proof. The proof follows by noting that for any set $\mathcal{M} \subset \mathcal{N}'$ and any intensity $\mathbf{x} = (\mathbf{x}^1, \dots, \mathbf{x}^k)$, decomposed by connected components:

$$\alpha(\mathcal{G}', \mathcal{M}, \mathbf{x}) = \min_{n \in \mathcal{M}} \frac{\sum_{a \in \mathcal{A}'} \mathbb{T}_{na} \mathbf{x}_a}{\sum_{a \in \mathcal{A}'} \mathbb{S}_{na} \mathbf{x}_a} = \min_{l=1, \dots, k} \min_{n_l \in \mathcal{M}_l} \frac{\sum_{a \in \mathcal{A}_l} \mathbb{T}_{n_l a} \mathbf{x}_a^l}{\sum_{a \in \mathcal{A}_l} \mathbb{S}_{n_l a} \mathbf{x}_a^l} = \min_{l=1, \dots, k} \alpha(\mathcal{G}_l, \mathcal{M}_l, \mathbf{x}^l).$$

Thus,

$$\alpha(\mathcal{G}') = \max_{\substack{\mathcal{M} \subset \mathcal{A}' \\ \mathbf{x} \in \mathbb{R}_+^T}} \alpha(\mathcal{G}', \mathcal{M}, \mathbf{x}) = \max_{l=1, \dots, k} \max_{\substack{\mathcal{M}_l \subset \mathcal{A}_l \\ \mathbf{x}_l \in \mathbb{R}_+^T}} \alpha(\mathcal{G}_l, \mathcal{M}_l, \mathbf{x}_l) = \max_{l=1, \dots, k} \alpha(\mathcal{G}_l).$$

□

Thus, in view of the result above, we propose a divide-et-conquer approach to solve, exactly, the problem. Specifically, the proposed procedure consists of projecting the hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{A})$ to a directed graph $\mathcal{G} = (V, A)$, where $V = \mathcal{N} \cup \mathcal{A}$, and

$$A = \{(v, a) \in \mathcal{N} \times \mathcal{A} : \mathbb{S}_{va} > 0\} \cup \{(a, v) \in \mathcal{A} \times \mathcal{N} : \mathbb{T}_{va} > 0\},$$

similar to the graph represented in Figure 5.2. That is, the hyperarcs in H link nodes in $\mathcal{N}$ with the hyperarcs where they participate as source, and hyperarcs in $\mathcal{A}$ with the nodes that participate as target.

This strategy is outlined in Algorithm 8, which identifies weakly connected components in a directed bipartite graph representing nodes and hyperarcs. The graph is constructed from the source matrix M_1 and the target matrix M_2 , with the algorithm outputting submatrices corresponding to each connected component.

The graph is built by interpreting M_1 and M_2 as adjacency matrices. Nodes represent nodes and hyperarcs in the hypergraph, and directed edges are established as follows:

- A directed edge from an node-node to a hyperarc-node is added if the corresponding entry in M_1 is positive.

- A directed edge from a hyperarc-node to an node-node is added if the corresponding entry in M_2 is positive.

Once the graph is constructed, a traversal method is used to identify weakly connected components—subsets of nodes connected via paths, regardless of edge direction. Each component groups nodes and hyperarcs that are directly or indirectly related.

For each weakly connected component, the algorithm determines the nodes and hyperarcs it contains. It then extracts the corresponding input and output incidence submatrices, representing the interactions within the component. These submatrices, along with their associated node and hyperarc indices, form the output. The algorithm continues until all components are processed, returning a comprehensive list of submatrices for all components. This decomposition simplifies the analysis by breaking down complex networks into smaller, manageable subsystems.

Algorithm 8: Algorithm to find weakly connected components and generate corresponding submatrices.

```

input :  $M_1, M_2$ 
output: Set of submatrices corresponding to each connected component
 $M_1, M_2 \in \mathbb{N}^{m \times n}$ 
 $number\_of\_nodes = \{1, \dots, m\}$ 
 $number\_of\_hyperarcs = \{1, \dots, n\}$ 
for  $i \leftarrow 1$  to  $number\_of\_nodes$  do
    for  $j \leftarrow 1$  to  $number\_of\_hyperarcs$  do
        if  $source\_matrix[i, j] > 0$  then
             $\_ \text{Add edge } (i, j) \text{ in } G$ 
        if  $target\_matrix[i, j] > 0$  then
             $\_ \text{Add edge } (j, i) \text{ in } G$ 
components  $\leftarrow$  List of weakly connected components of  $G$ 
components_all  $\leftarrow$  empty list
for  $component$  in  $components$  do
     $node\_indices \leftarrow$  empty list
     $hyperarc\_indices \leftarrow$  empty list
    for  $node$  in  $component$  do
        if  $node$  is  $Nodes$  then
             $\_ \text{Append node to nodes\_indices}$ 
        else if  $node$  in  $Hyperarc$  then
             $\_ \text{Append node to hyperarc\_indices}$ 
     $source\_submatrix \leftarrow$ 
         $source\_matrix[node\_indices, hyperarc\_indices]$ 
     $target\_submatrix \leftarrow$ 
         $target\_matrix[node\_indices, hyperarc\_indices]$ 
     $\_ \text{Append } (source\_submatrix, target\_submatrix) \text{ to } components\_all$ 
return  $components\_all$ 

```

5.5 Incorporating features to the mathematical model

In a self-amplifying subnetwork $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$, it is not required that every node in the set $\mathcal{N}'$ exhibits self-amplification. Instead, only the nodes in the designated *self-amplifying set* $\mathcal{M} \subseteq \mathcal{N}'$ must satisfy this property. Some nodes, although part of the source and target sets of hyperarcs in the subhypergraph that is optimal under our methodology, may never become self-sufficient or satisfy the net-positive realizability condition.

Given a subhypergraph $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$, together with its optimal self-amplifying set $\mathcal{M}$ and the corresponding intensity vector $\mathbf{x}$, the nodes in $\mathcal{N}' \setminus \mathcal{M}$ can be classified into three types:

Source Nodes: Nodes that appear only as sources in the hyperarcs of $\mathcal{A}'$, but

never as targets. That is, all $v \in \mathcal{N}' \setminus \mathcal{M}$ such that $\mathbb{T}_{va} = 0$ for all $a \in \mathcal{A}'$ and $\mathbb{S}_{va} > 0$ for some $a \in \mathcal{A}'$.

Sink Nodes: Nodes that appear only as targets in the hyperarcs of $\mathcal{A}'$, but never as sources. That is, all $v \in \mathcal{N}' \setminus \mathcal{M}$ such that $\mathbb{S}_{va} = 0$ for all $a \in \mathcal{A}'$ and $\mathbb{T}_{va} > 0$ for some $a \in \mathcal{A}'$.

Non-amplifying Nodes: Nodes that appear both as sources and targets in $\mathcal{A}'$, but whose net balance under the intensity vector $\mathbf{x}$ is negative. That is, there exist $a_1, a_2 \in \mathcal{A}'$ such that $\mathbb{S}_{va_1} > 0$ and $\mathbb{T}_{va_2} > 0$, but $\sum_{a \in \mathcal{A}'} \mathbb{Q}_{va} \mathbf{x}_a < 0$.

This classification has previously appeared in the context of Chemical Reaction Networks (CRNs), where nodes represent chemical species and hyperarcs represent reactions. In that literature, the three categories above are referred to as *Food*, *Waste*, and *Non-core* species, respectively.

Note that, with these types, source and non-amplifying nodes exhibit negative net aggregate in $\mathcal{H}'$, that is, for $v \in \mathcal{N}'$ classified as input or non-amplifying, $\sum_{a \in \mathcal{A}'} \mathbb{Q}_{va} \mathbf{x}_a < 0$. Thus, we refer to these as *drain nodes* under $\mathbf{x}$, denoted by $\mathcal{N}^-$. Analogously, the sink nodes will be denoted by $\mathcal{N}^+ := \mathcal{N}' \setminus (\mathcal{M} \cup \mathcal{N}^-)$, and, together with the self-amplifying nodes, these are *amplifying nodes*. Thus $\mathcal{N}' = \mathcal{M} \cup \mathcal{N}^- \cup \mathcal{N}^+$.

With all the above, although the construction of MAF subhypergraphs is a valuable tool for identifying the hyperarcs and nodes that exhibit the highest self-amplification potential and therefore contribute most significantly to system expansion, in some applications it may be required to construct MAF subhypergraphs under specific constraints tailored to particular scenarios. In this section, we present several interesting cases and explain how they can be explicitly incorporated into the mathematical optimization model we propose for constructing MAF subnetworks. Specifically, we detail how to designate certain nodes as sources, sinks, or non-amplifying within the resulting subnetwork using linear inequalities involving the decision variables already defined in our model.

Although one can incorporate different features to the model that we already presented above, such an approach focuses on constructing $\mathcal{M}$ and the hyperarcs, but there are no special conditions on the source or target nodes of those hyperarcs. Thus, in order to account for those nodes not in $\mathcal{M}$, it is required to incorporate to our model decision variables that allow us to identify those nodes, namely:

$$t_v = \begin{cases} 1 & \text{if } v \in \mathcal{N}', \forall v \in \mathcal{N} \\ 0 & \text{otherwise.} \end{cases}$$

that is, $t_v = 1$ if node v is in the selected hypergraph (that will maximize the MAF). This set of variables is adequately defined by the following set of inequalities:

$$t_v \geq z_a, \quad \forall v \in \mathcal{N} \text{ such that } v \in T_a \cup S_a, \quad (12)$$

$$t_v \leq \sum_{a \in \mathcal{A}: v \in T_a \cup S_a} z_a, \quad \forall v \in \mathcal{N}, \quad (13)$$

$$y_v \leq t_v, \quad \forall v \in \mathcal{N}. \quad (14)$$

That is, if hyperarc a is part of the subnetwork ($z_a = 1$), then all the nodes involved in a (as source or target) must be *activated* to be part of the subhypergraph $\mathcal{H}'$; and if all the hyperarcs in which a node v is either source or target are not in the subhypergraph, then v will not be a node of such a subhypergraph. The last set of constraints assures that the set of self-amplification nodes in $\mathcal{M}$ (those with $t_v = 1$) are chosen from the whole set of activated nodes $\mathcal{N}'(y_v = 1)$. So the $\mathcal{M}$ is a subset of $\mathcal{N}'$.

Those constraints can be additionally incorporated into our model, to construct, based on its solution $(\bar{y}, \bar{z})$, the optimal subhypergraph as $\mathcal{H}' = (\mathcal{N}', \mathcal{A}')$ with:

$$\mathcal{N}' = \{v \in \mathcal{N} : \bar{y}_v = 1\}, \mathcal{A}' = \{a \in \mathcal{A} : \bar{z}_a = 1\}.$$

With this encoding, one can require the obtained subhypergraph different conditions related to the types of nodes:

- **Subhypergraphs with specific source nodes:** Particular nodes can be considered as source nodes, which are assumed to participate as source but not as targets. In case one desires to construct subhypergraphs with MAF among those that uses a fixed subset of nodes as source nodes, $\mathcal{F} \subset \mathcal{N}$. This condition can be imposed to our model with the following sets of constraints:

$$t_v = 1, \quad \forall v \in \mathcal{F}, \quad (15)$$

$$z_a = 0, \quad \forall a \in \mathcal{A} : \exists v \in \mathcal{F} \text{ such that } v \in T_a \quad (16)$$

The first equations enforce that the source nodes are activated in the subhypergraph, whereas the second set of constraints avoid the use of source nodes as target of any hyperarc in the subnetwork.

- **Subhypergraphs with specific sink nodes:** Similarly to the previous consideration, some nodes are considered to be only target nodes of the hyperarcs but never source nodes. Thus, in one one desires to construct subhypergraphs with MAF among those that have a fixed sink set of nodes, $\mathcal{W} \subset \mathcal{N}$. This condition can be imposed to our model with the following sets of constraints:

$$t_v = 1, \quad \forall v \in \mathcal{W}, \quad (17)$$

$$z_a = 0, \quad \forall a \in \mathcal{A} : \exists v \in \mathcal{W} \text{ such that } v \in S_a. \quad (18)$$

Analogously to the previous specifications, the first equations enforce that the sink nodes are activated in the subhypergraph, whereas the second set of constraints avoid the use of those nodes as source of the hyperarcs.

- **Subhypergraphs with specific non-amplifying nodes:** In case a set of nodes $\mathcal{B} \subset \mathcal{N}$ is assumed to be source and target of the hyperarcs in the subhypergraph but with a negative net aggregation, we can impose this condition by the following constraints:

$$t_v = 1, \quad \forall v \in \mathcal{B}, \quad (19)$$

$$z_a = 0, \quad \forall a \in \mathcal{A} : \exists v \in \mathcal{B} \text{ such that } v \in T_a \setminus S_a \cup S_a \setminus T_a, \quad (20)$$

$$\mathbb{Q}_v \mathbf{x} \leq 0, \quad \forall v \in \mathcal{B}. \quad (21)$$

The first equations enforce that the non-amplifying nodes are activated in the subhypergraph. The second set of constraints avoid only the use of the nodes in $\mathcal{B}$ only as source or target of any hyperarc. Finally, the net aggregation under the intensity vector of these nodes is enforced to be nonpositive.

All these constraints can be combined; that is, if $(\mathcal{F}, \mathcal{W}, \mathcal{B})$ is a triplet of desired food, waste, and non-self-amplifying desired nodes, respectively, all the above conditions can be imposed together in our model.

5.6 Computational Experiments

To validate the performance of the mathematical optimization models, we conducted a series of computational experiments. These experiments also enable the comparison of the performance of the proposed algorithms on the same input data.

We run a series of computational experiments designed to evaluate the performance of the proposed algorithms across various synthetic instances. These instances are carefully generated using the Python software `SBbadger` proposed by Kochen et al. [2022] to reflect diverse problem characteristics, ensuring a thorough assessment of each algorithm’s efficiency, scalability, and robustness of random hypergraphs simulating chemical reaction networks. The primary objective of these experiments is to validate the proposed method for computing the MAF of a given subhypergraph. The results provide insights into the strengths and limitations of each algorithm and guide further optimization efforts.

We generated hypergraph-based systems of different sizes as follows: for each $n \in \{10, 20, 30, 40, 50\}$, we randomly simulated 8 instances with $|\mathcal{A}| = n$, using the function `generate_serial.models`. The software provides the incidence matrices $\mathbf{T}$ and $\mathbf{S}$ required to compute the MAF using the different methodologies.

We ran our algorithms until they identified the optimal solutions of the MAF problems. However, in the context of searching for self-amplifying subhypergraphs, Theorem 9 ensures that—even if the solution is not yet proven optimal—any iteration of Algorithm 5 or Algorithm 6 that yields a value of $\alpha > 1$ produces a subnetwork that satisfies the self-amplification conditions. Therefore, in our experiments, in addition to recording the CPU time required to certify optimality, we also report the first self-amplifying subhypergraph identified during the process, along with the time taken to find it.

We applied our approaches to all the generated instances, using Gurobi 11.03 as the optimization solver on an Ubuntu 22.04.03 environment with an AMD EPYC 7042p 24-Core Processor and 64 GB RAM. The maximum number of iterations for the algorithm was set to 1000.

For each instance, we recorded the following metrics for Algorithms 5 and 6: the amplification factor obtained at each iteration until convergence, the CPU time required for each iteration, and the corresponding feasible intensity vector $(\mathbf{x})$ achieving the amplification factor. Additionally, for Algorithm 6, we collected the nodes and hyperarcs comprising the optimal subhypergraph.

Table 5.1 presents the averaged results of the experiments. For each number of nodes in the system (first column) and each algorithm (second column), the

table shows the average number of iterations and CPU time (in seconds)—listed in columns `it` and `Time`, respectively—required to reach the optimal MAF. It also displays the average number of iterations and CPU time needed to identify the first self-amplifying subnetwork, shown in columns `First_it` and `Time_First`.

As observed, the computational demands increase as the number of decisions required by the algorithms grows. As expected, identifying the optimal subhypergraph is more challenging and time-consuming than other approaches. Nevertheless, while obtaining the optimal solution may be computationally expensive, detecting a self-amplifying subhypergraph (and determining whether one exists) can be achieved within reasonable CPU time.

# Nodes	Alg.	it	First_it	Time (secs.)	Time_First (secs.)
10	5	10.3	2.8	0.0882	0.0203
	6	3.9	2.3	0.0921	0.0375
20	5	11.1	4.7	0.4811	0.1416
	6	5.5	2.5	1.9684	0.0857
30	5	13.9	5.0	1.5921	0.2945
	6	8.3	4.2	81.8758	24.7882
40	5	10.0	3.5	1.1954	0.3067
	6	7.9	4.7	139.1985	114.9406
50	5	15.1	5.4	13.7698	2.6652
	6	8.2	5.1	2566.3931	121.2335

Table 5.1: Average results of the synthetic experiments.

As previously mentioned, detecting and enumerating all self-amplifying subhypergraphs is a challenging problem (NP-complete). However, the use of the MAF appears to mitigate the computational cost of these tasks, as illustrated in Figure 5.3. The left plot shows boxplots of the CPU times (in log scale) required to compute the subnetwork with the MAF for synthetic instances with node counts ranging from 10 to 50 (x-axis). As observed, certifying the optimality of the subhypergraph can require, in the worst-case scenario, hours of CPU time. In contrast, the right plot displays boxplots for the same instances, but focusing on the time required in the iterative procedure to find, for the first time, a subhypergraph with an MAF greater than 1 (i.e., a self-amplifying subnetwork). In this case, the procedure took only a few seconds to identify a self-amplifying subnetwork.

The second interesting observation drawn from our experiments is that although the MAF increases with the number of iterations of our algorithm, it stabilizes to a close-to-optimal value in a few iterations. This means that one could find a high-quality subhypergraph (in terms of the MAF) in a relatively small number of iterations. In Figure 5.4, we show the trend of these MAFs as the number of iterations increases across varying numbers of nodes in the system.

In Figure 5.5, we present the average CPU time required for each tested number of nodes. This information is further summarized in Figure 5.6, where the performance of all algorithms is plotted in the same figure, using a log scale for CPU time to facilitate interpretation. While Algorithm 5 requires fewer time, Algorithm 6 is significantly more demanding in terms of time.

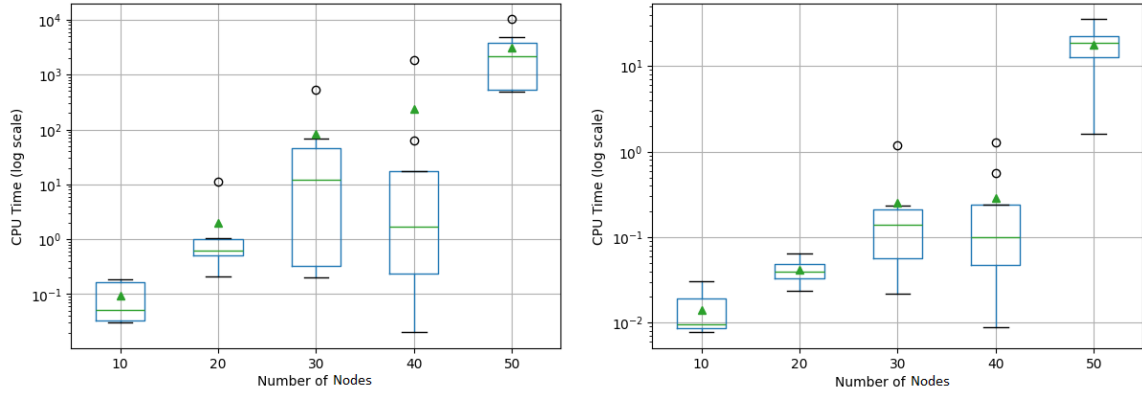


Figure 5.3: Boxplots of CPU times (in log scale) for two tasks across synthetic instances with 10 to 50 nodes. Time to compute the optimal MAF subhypergraph (left). Time to first find a self-amplifying subhypergraph ($\text{MAF} > 1$) (right).

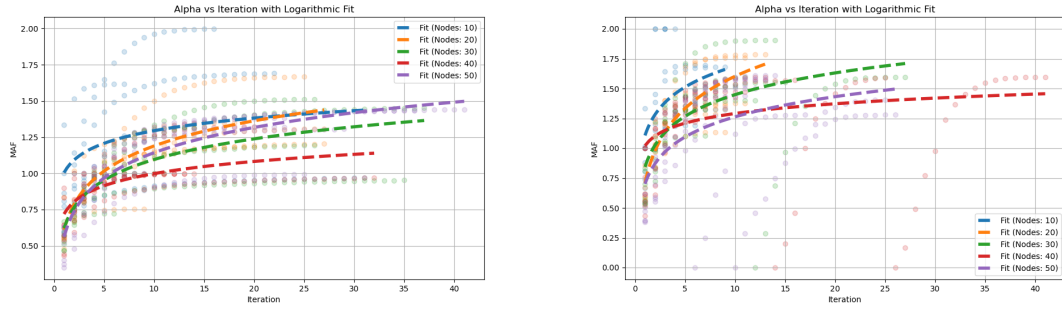


Figure 5.4: MAF values at each iteration for Algorithm 5 (left) and Algorithm 6 (right).

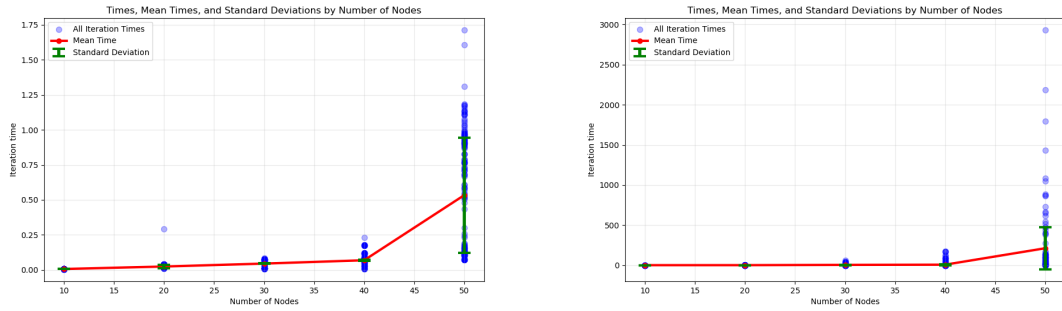


Figure 5.5: Variation in iteration time with the number of nodes for Algorithm 5 (left) and Algorithm 6 (right).

5.7 Conclusions

In this work, we introduce a novel framework for identifying and analyzing self-amplifying substructures in multi-directed hypergraphs, a generalization motivated by complex systems where multi-way interactions play a central role. We formalized

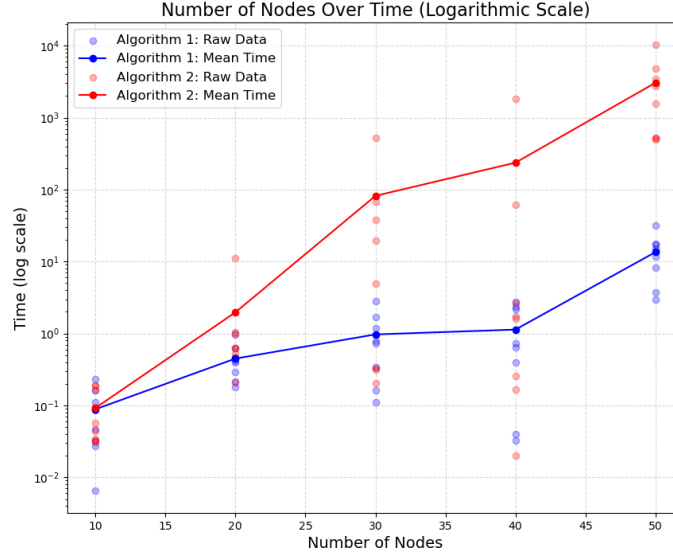


Figure 5.6: Total CPU time required by Algorithms 5 and 6 to solve the instances, plotted for comparison.

the concept of self-amplification through the lens of a newly proposed index—the Maximal Amplification Factor (MAF)—which quantifies the net self-reinforcing potential of a subhypergraph under optimal intensity vectors on its hyperarcs.

To compute the MAF, we formulate a generalized fractional optimization model and develop an exact iterative algorithm with convergence guarantees. We then extend this analysis to the combinatorial task of constructing subhypergraphs with the highest amplification factor, yielding a Mixed-Integer Nonlinear Programming (MINLP) model capable of identifying key subsystems that drive propagation dynamics. Our formulation accommodates real-world interpretability through additional structural constraints, making it adaptable to domain-specific applications such as chemistry, manufacturing, and social influence networks.

Computational experiments on synthetic hypergraphs simulating realistic chemical reaction networks demonstrated both the practical feasibility and potential of our approach in identifying influential subsystems. These results underscore the utility of MAF-based optimization in exploring emergent behavior in complex dynamical systems.

Overall, our study contributes to the growing body of work that seeks to extend combinatorial optimization into the realm of higher-order network structures. The notion of self-amplifying subhypergraphs offers a unifying abstraction for a wide array of processes involving self-sustaining or catalytic dynamics. Future directions include improving scalability through decomposition techniques, extending the framework to stochastic or time-evolving hypergraphs, and exploring domain-specific applications in greater depth—particularly in systems biology, process design, and network epidemiology.

Chapter 6

Conclusions

This thesis contributes to the advancement of mathematical optimization in the design of complex networks, particularly those arising in constrained, real-world engineering contexts. By introducing new network design problems and formulating rigorous mathematical models to address them, this work provides both theoretical innovations and practical tools for the engineering design community. Grounded in real industrial challenges—especially in naval engineering—our results bridge the gap between abstract mathematical models and deployable design methodologies.

We have rigorously validated our proposed methodologies through extensive computational experiments using both randomly generated instances and real-world data. Each chapter of this thesis (Chapters 2 to 5) stands as a self-contained contribution and includes its own conclusions. Below, we provide a summary of the key findings, analyze our findings, and discuss potential avenues for future research.

6.1 Summary of Contributions

This section consolidates the main contributions of the thesis, encompassing novel problem formulations, exact and heuristic optimization techniques, and extensive computational validations. Each contribution is aligned with the objectives defined in Section 1.1, and corresponds to the research works detailed in Chapters 2 to 5.

Chapter 2: Optimal Routing of Spatial Pipelines

This chapter addresses the automatic routing of spatial pipelines in constrained 3D environments, motivated by a real-world collaboration with Ghenova, a leading naval engineering firm. A comprehensive methodology based on a minimum-cost multi-commodity flow model is introduced, enhanced by constructability constraints. Due to the model's complexity, a branch-and-cut algorithm is proposed, dynamically incorporating technical constraints as lazy constraints. Two heuristic approaches are developed: a matheuristic on trimmed problem instances and a decomposition heuristic using iteratively modified shortest paths. Both methods efficiently produce high-quality solutions for medium-sized instances. The approach is validated through computational experiments and a real-world case study, confirming its practical utility for engineering design.

Chapter 3: Cable Tray and Pipeline Routing

This chapter presents the Pipelines and Cable Trays Location Problem (PCTLTP), an optimization framework for routing pipelines and cable trays in complex ship layouts. The model accounts for practical constraints such as elbow distances, separation of services, and the possibility of shared trays. An exact relax-and-cut method is proposed, supported by heuristics for larger instances. Experiments on synthetic and real-world layouts demonstrate the efficiency and flexibility of the methodology. A key contribution is the integration of routing with the connection problem, ensuring compliance with system labeling and constructibility requirements. This integration leads to more efficient, realistic, and cost-effective designs.

Chapter 4: Fixed Topology Minimum-Length Tree with Neighborhood Problem (FTTNP)

This chapter introduces the FTTNP, a novel network design problem where a rooted tree-shaped graph must be embedded in a d -dimensional metric space, minimizing total length while satisfying neighborhood constraints for node placement. The problem has direct applications in cable and pipeline layout. Two formulations are considered: a continuous version using Mixed-Integer Second Order Cone Programming (MISOCP) and a discrete version using Mixed-Integer Linear Programming (MILP). The proposed optimization strategies leverage transformations inspired by the Steiner tree problem and enable practical applications in industrial design contexts.

The chapter also addresses the industrial schematic layout problem, formulated as a constrained forest embedding in 3D space. The MILP-based framework incorporates realistic routing constraints, including bend radii, spatial and safety distances, and obstacle avoidance. A hierarchical decomposition into trees reflects the structure of industrial wiring systems and aids in managing complexity. Local Hanan grids are employed to discretize the feasible routing space effectively.

Chapter 5: Maximal Amplification in Hypergraphs

This chapter introduces the Maximal Amplification Factor (MAF), a novel metric for quantifying the self-reinforcing capacity of subhypergraphs. A generalized fractional optimization model is proposed, with an exact iterative solution algorithm that guarantees convergence. The method is extended to a Mixed-Integer Nonlinear Programming (MINLP) model for identifying subhypergraphs with the highest amplification potential under structural constraints. Computational experiments with synthetic chemical reaction networks validate the approach and reveal its ability to isolate catalytically active subsystems. This framework is adaptable to various applications in systems biology, network science, and reaction dynamics.

Concluding Remarks

Across the chapters, this thesis introduces and addresses several novel problems in complex network design with applications in engineering and systems biology. Key

contributions include:

- The formulation of five original optimization problems related to spatial routing, network embedding, and dynamic system analysis;
- Development of exact solution approaches using MILP, MISOCP, and fractional programming;
- Design of heuristic and matheuristic algorithms for scalability and practical performance;
- Validation of methods through synthetic benchmarks and real-world industrial case studies.

Collectively, the contributions illustrate how mathematical programming—when enriched by domain-specific constraints and guided by industrial requirements—can deliver scalable and realistic design methodologies.

6.2 Future Research

The findings in this thesis raise several open questions and promising research directions. These future avenues span both methodological advances and application-driven refinements. We outline below the main opportunities under each theme, drawing from the contributions of Chapters 2 through 5.

Scalability and Computational Complexity

Improving computational efficiency for large-scale instances is a recurring challenge across all chapters. Future research could focus on developing more advanced heuristics and approximation methods to handle complex topologies, such as non-orthogonal or multi-level networks. Hybrid approaches that combine exact optimization with metaheuristics or decomposition techniques could offer promising directions, especially for large-scale ship layout and infrastructure problems. In problems such as the Fixed Topology Tree with Neighborhood Problem (FTTNP) and 3D wiring layout, introducing additional structural elements (e.g., elbows, directional costs) could demand tailored heuristics to preserve tractability. For hypergraph-based models, scalability could be enhanced through decomposition, relaxation, or surrogate strategies to extend applicability to larger real-world networks.

Integration of Domain Knowledge and Constraints

The integration of domain-specific knowledge could significantly improve the realism and utility of optimization models. Future work could enhance the pipeline and cable routing framework by combining it with ship cabin layout optimization, enabling joint spatial and connectivity design. Richer connection logic, including labeling and port types, could support better integration with existing ship design tools. For multi-commodity routing frameworks, adding support for inter-deck or tiered

infrastructures could introduce realistic constraints, while incorporating engineering standards—such as safety margins and maintainability rules—could streamline regulatory approval and reduce costly design revisions in industrial practice.

Learning-Augmented Optimization

Incorporating machine learning techniques into optimization models could reduce manual input and improve adaptability to different engineering settings. For instance, learning-based models could predict routing cost parameters from historical data, reducing reliance on designer intuition and enhancing the responsiveness of optimization frameworks.

Uncertainty and Dynamic Systems

Robust design under uncertainty and adaptation to dynamic environments are crucial for many real-world systems. In the context of 3D wiring and network layout, future models could incorporate stochastic formulations or robust optimization techniques to account for variability in obstacle locations or endpoint configurations. Similarly, for dynamic hypergraphs, extending the MAF framework to time-evolving or probabilistic structures could support the analysis of processes such as biochemical pathways, social interactions, or adaptive networks under uncertainty.

Multi-objective and Holistic Design Strategies

Beyond cost minimization, future optimization frameworks could support multiple, often competing objectives. For instance, in routing problems, integrating criteria such as safety, redundancy, maintainability, and total lifecycle cost could offer a more comprehensive and practically relevant design strategy. This could be particularly valuable in applications like naval architecture, where modular and reconfigurable designs are increasingly common, and trade-offs between short-term cost and long-term flexibility are central to decision-making.

By pursuing these research directions, future work could further advance the state of the art in automatic network design, with significant impact across naval engineering, infrastructure design, and the analysis of complex systems.

6.3 Final Remarks

In summary, this work advances network design by formulating previously unexplored optimization challenges and crafting tailored solution strategies grounded in mathematical programming. The thesis also underscores the interdisciplinary potential of optimization, bridging fields as diverse as naval engineering, systems biology, and infrastructure design.

We have addressed real-world challenges, particularly in naval engineering and industrial design, by creating efficient algorithms for complex network problems. These methodologies promise real-world impact—offering both theoretical insight and tangible design enhancements.

This work establishes a robust foundation for future exploration that can enhance both theoretical models and practical deployments. These include extending existing models to handle more complex network topologies, incorporating machine learning to improve parameter predictions, and further exploring hybrid and heuristic algorithms for large-scale instances. Additionally, the integration of different design elements, such as ship cabin layouts or network parameters, into the optimization process would contribute to more comprehensive and effective network designs.

In an increasingly interconnected world, the design of efficient, resilient, and intelligent networks is more crucial than ever. This thesis lays the groundwork for further exploration at the intersection of theory and application, where optimization continues to shape the infrastructure of the future.

Bibliography

- The palgrave encyclopedia of strategic management, 2020. URL <https://link.springer.com/referencework/10.1057/978-1-349-94848-2>.
- B Devadas Acharya and Suresh Manjanath Hegde. Arithmetic graphs. *Journal of Graph Theory*, 14(3):275–299, 1990.
- Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993. ISBN 9780136175490.
- Farid Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995. doi: 10.1137/0805002. URL <https://doi.org/10.1137/0805002>.
- Jakob L. Andersen, Christoph Flamm, Daniel Merkle, and Peter F. Stadler. Chemical transformation motifs—modelling pathways as integer hyperflows. 16(2):510–523.
- Y. Ando and H. Kimura. An automatic piping algorithm including elbows and bends. In *International Conference on Computer Applications in Shipbuilding (ICCAS)*, pages 153–158, Trieste, Italy, 2011.
- Y. P. Aneja. An integer linear programming approach to the steiner problem in graphs. *Networks*, 10(2):167–178, 1980.
- O. Arslan, C. Archetti, O. Jabali, G. Laporte, and M. Grazia Speranza. Minimum cost network design in strategic alliances. *Omega*, 96:102–079, 2020.
- A. Asmara. *Pipe routing framework for detailed ship design*. PhD thesis, Delft University of Technology, Netherlands, 2013.
- A. Asmara and U. Nienhuis. Automatic piping system in ship. In *International Conference on Computer and IT Applications (COMPIT)*, Delft, Netherlands, 2006a.
- A. Asmara and U. Nienhuis. Automatic piping system in ship. In *International Conference on Computer and IT Application (COMPIT)*, pages 269–280. Citeseer, 2006b.
- Giorgio Ausiello and Luigi Laura. Directed hypergraphs: introduction and fundamental algorithms—a survey. *Theoretical Computer Science*, 658:293–306, 2017.

- Ana Isabel Barros, JBG Frenk, Siegfried Schaible, and Shuzhong Zhang. A new algorithm for generalized fractional programs. *Mathematical Programming*, 72: 147–175, 1996.
- Federico Battiston, Enrico Amico, Alain Barrat, Ginestra Bianconi, Guilherme Ferraz de Arruda, Benedetta Franceschiello, Iacopo Iacopini, Sonia Kéfi, Vito Latora, Yamir Moreno, et al. The physics of higher-order interactions in complex systems. *Nature Physics*, 17(10):1093–1098, 2021.
- Piotr Berman. Improved approximation lower bounds on small occurrence optimization. *ECCC Report*, 2003.
- Christian Bick, Elizabeth Gross, Heather A Harrington, and Michael T Schaub. What are higher-order networks? *SIAM review*, 65(3):686–731, 2023.
- V. Blanco, G. González, Y. Hinojosa, D. Ponce, M.A. Pozo, and J. Puerto. Network flow based approaches for the pipelines routing problem in naval design. *Omega*, 111:102659, 2022.
- Víctor Blanco and Miguel Martínez-Antón. On minimal extended representations of generalized power cones. *SIAM Journal on Optimization*, 34(3):3088–3111, 2024.
- Víctor Blanco and Justo Puerto. On hub location problems in geographically flexible networks. *International Transactions in Operational Research*, 29(4):2226–2249, 2022.
- Víctor Blanco, Safae ElHaj-BenAli, and Justo Puerto. Minimizing ordered weighted averaging of rational functions with applications to continuous location. *Computers & Operations Research*, 40(5):1448–1460, 2013.
- Victor Blanco, Justo Puerto, and Safae El Haj Ben Ali. Revisiting several problems and algorithms in continuous location with ℓ_r norms. *Computational Optimization and Applications*, 58(3):563–595, 2014.
- Víctor Blanco, Elena Fernández, and Justo Puerto. Minimum spanning trees with neighborhoods: Mathematical programming formulations and solution methods. *European Journal of Operational Research*, 262(3):863–878, 2017.
- Víctor Blanco, Gabriel González, Yolanda Hinojosa, Diego Ponce, Miguel A Pozo, and Justo Puerto. The pipelines and cable trays location problem in naval design. *Ocean Engineering*, 286:115525, 2023.
- Alex Blokhuis, David Lacoste, and Philippe Nghe. Universal motifs and the diversity of autocatalytic systems. *Proceedings of the National Academy of Sciences*, 117(41):25230–25236, 2020.
- Marcus Brazil, Doreen A Thomas, Benny K Nielsen, Pawel Winter, Christian Wulff-Nilsen, and Martin Zachariasen. A novel approach to phylogenetic trees: d-dimensional geometric steiner trees. *Networks: An International Journal*, 53(2): 104–111, 2009.

- Marcus Brazil, Michael Hendriksen, Jae Lee, Michael S Payne, Charl Ras, and Doreen Thomas. An exact algorithm for the euclidean k-steiner tree problem. *Computational Geometry*, 121:102099, 2024.
- P. M. Pardalos (Eds.) C. A. Floudas. *Encyclopedia of Optimization*. Springer, 2009.
- V. Castorani, P. Cicconi, M. Mandolini, A. Vita, and M. Germani. A method for the cost optimization of industrial electrical routings. *Computer-Aided Design and Applications*, 15:747–756, 2018.
- Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., United States, 1997.
- William J Cook, William H Cunningham, William R Pulleyblank, and Alexander Schrijver. Combinatorial optimization. *Unpublished manuscript*, 10:75–93, 1994.
- Jean-Pierre Crouzeix and Jacques A Ferland. Algorithms for generalized fractional programming. *Mathematical Programming*, 52:191–207, 1991.
- JP Crouzeix, JA Ferland, and S Schaible. An algorithm for generalized fractional programs. *Journal of Optimization Theory and Applications*, 47(1):35–49, 1985.
- F. Cuervas, A. Tordera, A. Fontán, P. Brenes, C. Alejo, A. Tovar, J. Puerto, E. Conde, F. Ortega, and Y. Hinojosa. Ariadna: sistema automático de trazado de tuberías y canalizaciones en ingeniería. *Ingeniería Naval*, 963:75–84, 2017.
- Francisco Saldanha da Gama and Shuming Wang. *Facility Location Under Uncertainty: Models, Algorithms and Applications*, volume 356 of *International Series in Operations Research & Management Science*. Springer, 2024. ISBN 978-3-031-55927-3. doi: 10.1007/978-3-031-55927-3. URL <https://link.springer.com/book/10.1007/978-3-031-55927-3>.
- George Bernard Dantzig, Alex Orden, and Philip S. Wolfe. Notes on linear programming: Part i: The generalized simplex method for minimizing a linear form under linear inequality restraints. Technical Report RM-1264, RAND Corporation, 1954. URL https://www.rand.org/pubs/research_memoranda/RM1264.html.
- Mamadi Diané. An integer programming formulation of the steiner problem in graphs. *Zeitschrift für Operations Research*, 37:107–111, 1993.
- E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
- Yann Disser, Matúš Mihalák, Sandro Montanari, and Peter Widmayer. Rectilinear shortest path and rectilinear minimum spanning tree with neighborhoods. In *International Symposium on Combinatorial Optimization*, pages 208–220. Springer, 2014.
- Z.r. Dong, X.y. Bian, and S. Zhao. Ship pipe route design using improved multi-objective ant colony optimization. *Ocean Engineering*, 258:111789, 2022.

- Reza Dorrigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum-and maximum-weight minimum spanning trees with neighborhoods. *Theory of Computing Systems*, 56: 220–250, 2015.
- Zvi Drezner, Kathrin Klamroth, Anita Schöbel, and George O Wesolowsky. The weber problem. *Facility location: Applications and theory*, pages 1–36, 2002.
- Ding-Zhu Du, Biao Gao, Ronald L Graham, Zi-Cheng Liu, and Peng-Jun Wan. Minimum steiner trees in normed planes. *Discrete & Computational geometry*, 9: 351–370, 1993.
- Inmaculada Espejo, Raúl Páez, Justo Puerto, and Antonio M. Rodríguez-Chía. Minimum cost b-matching problems with neighborhoods. *Computational Optimization and Applications*, 83:525–553, 2022.
- S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science (SFCS 1975)*, pages 184–193. IEEE, 1975.
- Marcia Fampa, Jon Lee, and Nelson Maculan. An overview of exact algorithms for the euclidean steiner tree problem in n-space. *International Transactions in Operational Research*, 23(5):861–874, 2016.
- Sándor P Fekete, Joseph SB Mitchell, and Karin Beurer. On the continuous fermat-weber problem. *Operations Research*, 53(1):61–76, 2005.
- Martina Fischetti and David Pisinger. Optimizing wind farm cable routing considering power losses. *European Journal of Operational Research*, 270(3):917–930, 2018.
- Matteo Fischetti, Juan-José Salazar-Gonzalez, and Paolo Toth. *The Generalized Traveling Salesman and Orienteering Problems*, pages 609–662. Springer US, Boston, MA, 2007. ISBN 978-0-306-48213-7. doi: 10.1007/0-306-48213-4_13. URL https://doi.org/10.1007/0-306-48213-4_13.
- Henry Fleischmann, Surya Teja Gavva, and Karthik C. S. On approximability of steiner tree in ℓ_p -metrics. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1669–1703, 2024.
- Praful Gagrani, Victor Blanco, Eric Smith, and David Baum. Polyhedral geometry and combinatorics of an autocatalytic ecosystem. *Journal of Mathematical Chemistry*, pages 1–67, 2024.
- David Gale. *The theory of linear economic models*. University of Chicago press, United States, 1989.
- Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

- Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. 1979.
- Michael R Garey, Ronald L Graham, and David S Johnson. The complexity of computing steiner minimal trees. *SIAM journal on applied mathematics*, 32(4): 835–859, 1977.
- M. Garg and J. Smith. Models and algorithms for the design of survivable multi-commodity flow networks with general failure scenarios. *Omega*, 36:1057–1071, 2008.
- L.R. Guimarães, J.P. de Sousa, and B.d.A. Prata. Variable fixing heuristics for the capacitated multicommodity network flow problem with multiple transport lines, a heterogeneous fleet and time windows. *Transportation Letters*, 14:84–93, 2022.
- R. Guiradello and R. S. Swaney. Optimization of process plant layout with pipe routing. *Computers and Chemical Engineering*, 30:99–114, 2005.
- R. Guirardello and R.E. Swaney. Optimization of process plant layout with pipe routing. *Computers & chemical engineering*, 30:99–114, 2005.
- Venkatesan Guruswami and Rishi Saket. On the inapproximability of vertex cover on k-partite k-uniform hypergraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 360–371. Springer, 2010.
- M Hanan. On steiner’s problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14:255–265, 3 1966. ISSN 0036-1399.
- John M Harris. *Combinatorics and graph theory*. Springer, Science+Business Media, 2008.
- Juha Heinonen. *Geometric embeddings of metric spaces*. University of Jyväskylä, Finland, 2003.
- D. Hightower. A solution to line routing problems on the continuous plane. In *Proceedings of Sixth Annual Design Automation Conference. IEEE*, pages 1–24, 1969a.
- D.W. Hightower. A solution to line-routing problems on the continuous plane. In *Proceedings of the 6th annual Design Automation Conference*, pages 1–24, 1969b.
- Wim Hordijk and Mike Steel. Detecting autocatalytic, self-sustaining sets in chemical reaction systems. *Journal of theoretical biology*, 227(4):451–461, 2004.
- Wim Hordijk and Mike Steel. Autocatalytic networks at the basis of life’s origin and organization. *Life*, 8(4):62, 2018.
- Wim Hordijk, Mike Steel, and Peter Dittrich. Autocatalytic sets and chemical organizations: modeling self-sustaining reaction networks at the origin of life. 20 (1):015011.

- S. Ikehira, H. Kimura, and H. Kajiwaru. Automatic design for pipe arrangement using multi-objective genetic algorithms. In *International Conference on Computer Applications in Shipbuilding (ICCAS)*, pages 97–110, Busan, Korea, 2005.
- T. Ito. A genetic algorithm approach to piping route planning. *Journal of Intelligence Manufacturing*, pages 103–114, 1999.
- W.Y. Jiang, Y. Lin, M. Chen, and Y.Y. Yu. A co-evolutionary improved multi-ant colony optimization for ship multiple and branch pipe route design. *Ocean Engineering*, 102:63–70, 2015.
- Mark Johnson and Graham Stevens. Integrating the supply chain... 25 years on. *International Journal of Physical Distribution & Logistics Management*, 46, 02 2016. doi: 10.1108/IJPDLM-07-2015-0175.
- William Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, Chicago, IL, USA, 1939.
- L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980. ISSN 0041-5553. doi: [https://doi.org/10.1016/0041-5553\(80\)90061-0](https://doi.org/10.1016/0041-5553(80)90061-0). URL <https://www.sciencedirect.com/science/article/pii/0041555380900610>.
- Annika Kristina Kiefner. Minimizing path lengths in rectilinear steiner minimum trees with fixed topology. *Operations Research Letters*, 44(6):835–838, 2016.
- S. H. Kim, W. S. Ruy, and B. S. Jang. The development of a practical pipe auto-routing system in a shipbuilding cad environment using network optimization. *Int. J. Naval Archit. Ocean Eng.*, 5:468–477, 2013.
- H. Kimura. Automatic designing system for piping and instruments arrangement including branches of pipes. In *International Conference on Computer Applications in Shipbuilding (ICCAS2011)*, volume 3, pages 93–99, 2011.
- H. Kimura. Automatic piping arrangement design considering piping supports and curved surfaces of building blocks. In *International Conference on Computer Applications in Shipbuilding, ICCAS 2017*, volume 3, pages 107–114, 2017.
- Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS computational biology*, 5(5):e1000385, 2009.
- Michael A Kochen, H Steven Wiley, Song Feng, and Herbert M Sauro. Sbbadger: biochemical reaction networks with definable degree distributions. *Bioinformatics*, 38(22):5064–5072, 2022.
- Yanna J Kraakman and Clara Stegehuis. Configuration models for random directed hypergraphs. *arXiv preprint arXiv:2402.06466*, 2024.

- Harold W. Kuhn and Albert W. Tucker. Nonlinear programming. *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, 1951. URL <https://econgrowth.github.io/notebooks/papers/kuhntucker1950.pdf>.
- C. Y. Lee. An algorithm for path connections and its applications. *IEEE Transactions on Electronic Computers*, 10(3):346–365, 1961.
- X. Li, S. Lin, P. Tian, and Y. Aneja. Models and column generation approach for the resource-constrained minimum cost path problem with relays. *Omega*, 66: 79–90, 2017.
- Chao Liu, Lei Wu, Guangxin Li, Hao Zhang, Wensheng Xiao, Dengpan Xu, Jingjing Guo, and Wentao Li. Improved multi-search strategy a* algorithm to solve three-dimensional pipe routing design. *Expert Systems with Applications*, 240:122313, 2024.
- Ivana Ljubić. Solving steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021.
- Ivana Ljubić, René Weiskircher, Ulrich Pferschy, Gunnar W Klau, Petra Mutzel, and Matteo Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Mathematical programming*, 105:427–449, 2006.
- Wang Lusheng and Xiaohua Jia. Fixed topology steiner trees and spanning forests. *Theoretical computer science*, 215(1-2):359–370, 1999.
- Nelson Maculan. The steiner problem in graphs. In Silvano Martello, Gilbert Laporte, Michel Minoux, and Celso Ribeiro, editors, *Surveys in Combinatorial Optimization*, volume 132 of *North-Holland Mathematics Studies*, pages 185–211. North-Holland, 1987.
- Nelson Maculan, Philippe Michelon, and Adilson E. Xavier. The euclidean steiner tree problem in $\mathbb{R}^n$: A mathematical programming formulation. *Annals of Operations Research*, 96:209–220, 2000.
- Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.
- Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. doi: 10.1111/j.1540-6261.1952.tb01525.x. URL <https://onlinelibrary.wiley.com/doi/10.1111/j.1540-6261.1952.tb01525.x>.
- Nestor F Michelena and Panos Y Papalambros. A hypergraph framework for optimal model-based decomposition of design problems. *Computational Optimization and Applications*, 8:173–196, 1997.
- Seda Milos. Steiner tree problem in graphs and mixed integer linear programming-based approach in gams. *WSEAS Transactions on Computers*, 21:257–262, 07 2022.

- J. Min, W. Ruy, and C. Park. Faster pipe auto-routing using improved jump point search. *International Journal of Naval Architecture and Ocean Engineering*, pages 596–604, 2020.
- T. Mitsuta, Y. Kobayashi, Y. Wada, T. Kiguchi, and T. Yoshinaga. A knowledge-based approach to routing problems in industrial plant design. In *Proceedings of the Sixth International Workshop Expert System and Their Applications*, pages 237–256, Avignon, France, March 1986.
- H. Mostafaei, P. M. Castro, S. Relvas, and I. Harjunoski. A holistic milp model for scheduling and inventory management of a multiproduct oil distribution system. *Omega*, 98:102–110, 2021.
- László Nagy, Tamás Ruppert, Andreas Löcklin, and János Abonyi. Hypergraph-based analysis and design of intelligent collaborative manufacturing space. *Journal of Manufacturing Systems*, 65:88–103, 2022.
- Yu. Nesterov and A. Nemirovsky and. Conic formulation of a convex programming problem and duality †. *Optimization Methods and Software*, 1(2):95–115, 1992. doi: 10.1080/10556789208805510. URL <https://doi.org/10.1080/10556789208805510>.
- Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied and Numerical Mathematics. Society for Industrial and Applied Mathematics, 1994. doi: 10.1137/1.9781611970791. URL <https://epubs.siam.org/doi/book/10.1137/1.9781611970791>.
- R. G. Newell. *Algorithms for the design of chemical plant layout and pipe routing*. PhD thesis, Imperial College, London, England, 1973.
- Mark Newman. *Networks*. Oxford University Press, 07 2018. ISBN 9780198805090. doi: 10.1093/oso/9780198805090.001.0001. URL <https://doi.org/10.1093/oso/9780198805090.001.0001>.
- Stefan Nickel and Justo Puerto. *Location theory. A unified approach*, volume 66. 01 2009. ISBN 978-3-540-24321-2. doi: 10.1007/3-540-27640-8.
- Akatsuki Nishioka, Mitsuru Toyoda, Mirai Tanaka, and Yoshihiro Kanno. On a minimization problem of the maximum generalized eigenvalue: properties and algorithms. *Computational Optimization and Applications*, 90(1):303–336, 2025.
- J.B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78:109–129, 1997.
- Hacene Ouzia and Nelson Maculan. Mixed integer nonlinear optimization models for the euclidean steiner tree problem in $\mathbb{R}^d$. *Journal of Global Optimization*, 83: 119–136, 2022.
- Guillermo Owen. *Game Theory*. Saunders, Philadelphia, 1968. URL <https://archive.org/details/gametheory0000owen>.

- J. Park. *Pipe-routing algorithm development for a ship engine room design*. PhD thesis, Washington University, 2002.
- J. H. Park and R. L. Storch. Pipe-routing algorithm development: case study of a ship engine room design. *Expert Systems with Applications*, 23:299–309, 2002.
- M. Park, B. Jeong, and M. Kim. Decision-making for cable routing at detailed ship design through life cycle and cost assessment. *Journal of International Maritime Safety, Environmental Affairs, and Shipping*, 4:93–107, 2020.
- Renan Vicente Pinto and Nelson Maculan. A new heuristic for the euclidean steiner tree problem in $\mathbb{R}^n$. *Top*, 31(2):391–413, 2023.
- Renan Vicente Pinto, Hacène Ouzia, and Nelson Maculan. A new second-order conic optimization model for the euclidean steiner tree problem in $\mathbb{R}^d$. *International Transactions in Operational Research*, 30(6):3886–3903, 2023.
- S. Pogorelskiy and I. Kocsis. Automation for structured cabling system in data centers using building information modelling. *International Review of Applied Sciences and Engineering*, 2022.
- Justo Puerto and Carlos Valverde. The hampered travelling salesman problem with neighbourhoods. *Computers & Industrial Engineering*, 188:109889, 2024.
- X. Qian, T. Ren, and C. Wang. A survey of pipe routing design. In *Control and Decision Conference, Chinese IEEE*, pages 3994–3998, Yantai, Shandong, China, 2008.
- Vladimir Repisky et al. Covering edges of a hypergraph: complexity and applications. 2007.
- P. W. Rourke. *Development of a three-dimensional pipe routing algorithm*. PhD thesis, Lehigh University, 1975.
- K. Salimifard and S. Bigharaz. The multicommodity network flow problem: state of the art classification, applications, and solution methods. *Operational Research*, 2020.
- David Sankoff and Pascale Rousseau. Locating the vertices of a steiner tree in an arbitrary metric space. *Mathematical Programming*, 9:240–246, 1975.
- H. Schmidt-Traub, T. Holtkötter, M. Lederhose, and P. Leuders. An approach to plant layout optimization. *Chemical Engineering Technology*, 22(2):105–109, 1999.
- M. T. Shing and T. C. Hu. Computational complexity of layout problems. In *Layout design and verification*, pages 267–294. Elsevier Science Publishers B.V., North Holland, 1986.
- J. Singh and J. Cheng. Automating the generation of 3d multiple pipe layout design using bim and heuristic search methods. In *18th International Conference on Computing in Civil and Building Engineering. ICCCBE 2020*, volume 98, pages 54–72, 2021.

- J von Neumann. A model of general economic equilibrium. *The Review of Economic Studies*, 13(1):1–9, 1945.
- Weizheng Wang, Aniket Bera, and Byung-Cheol Min. Hyper-samarl: Hypergraph-based coordinated task allocation and socially-aware navigation for multi-robot systems. *arXiv preprint arXiv:2409.11561*, 2024.
- Z. Wang, X.q. Yang, Y.h. Zheng, W.C. Chen, P. Lv, B. Zhou, and M.l. Xu. Interactive ship cabin layout optimization. *Ocean Engineering*, 270:113647, 2023.
- F. Xiaoning, L. Yan, and J. Zhuoshang. The ant colony optimization for ship pipe route design in 3d space. In *World Congress on Intelligent Control and Automation (WCICA)*, pages 3103–3108, Dalian, China, 2006.
- F. Xiaoning, L. Yan, and J. Zhuoshang. Ship pipe routing design using the aco with iterative pheromone updating. *Journal of Ship Production*, 23(1):36–45, 2007.
- Yang Yang, Mingen Lin, Jinhui Xu, and Yulai Xie. Minimum spanning tree with neighborhoods. In *International Conference on Algorithmic Applications in Management*, pages 306–316. Springer, 2007.
- C. Yeates, C. Schmidt-Hattenberger, W. Weinzierl, and D. Bruhn. Heuristic methods for minimum-cost pipeline network design: A node valency transfer meta-heuristic. *Networks and Spatial Economics: a journal of infrastructure modeling and computation*, 21:839–871, 2021.
- D. Zhu and J. Latombe. Pipe routing-path planning (with many constraints). In *IEEE International Conference on Robotics and Automation (ICRAS)*, pages 1940–1947, Sacramento, California, 1991.