Conceptualization of Graph - Oriented Databases Using Digraphs
José Rodrigues Garcia Ribas
UMinho | 2026

Universidade do Minho
Escola de Engenharia

José Rodrigues Garcia Ribas

Conceptualization of Graph – Oriented Databases Using Digraphs

Doctoral Programme in Informatics at the Universities of Minho, Aveiro, and Porto (MAP-i)

universidade de aveiro
Universidade do Minho
U.PORTO

Julho de 2026



Universidade do Minho
Escola de Engenharia

José Rodrigues Garcia Ribas

Conceptualização de Bases de Dados Orientadas a Grafos usando Dígrafos

Programa de Doutoramento em Informática das Universidades do Minho, de Aveiro e do Porto



universidade de aveiro



Universidade do Minho

U.PORTO

Trabalho realizado sob a orientação do

Professor Doutor Orlando Manuel de Oliveira Belo

Julho, 2026

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to the users of this work



Creative Commons Attribution 4.0 International
CC BY 4.0

https://creativecommons.org/licenses/by/4.0/

Acknowledgements

I express my deep gratitude and appreciation to Professor Orlando Manuel Oliveira Belo, my supervisor, in this research. His guidance was fundamental for the completion of this work, thanks to his constant availability, critical accompaniment and faith in my work. The perspicacity of his scientific thinking, the clarity of his observations and the ability to lead me firmly, but also encouragement was decisive for the solidity of this study. Throughout my academic career, he not only represented valuable technical and scientific support, but also an example of intellectual seriousness and dedication to the cause of knowledge.

I would also like to thank all those who are part of the Polytechnic Institute of Viana do Castelo for the institutional support, for the working conditions provided and for the encouragement shown at different times. The academic environment, the collaboration between colleagues and the opportunities made available were essential to face the inevitable phases of attrition and overcome the challenges inherent to a project of this nature and size.

I also address a word of gratitude to my family. To my three children, André, Marco and Mariana, and to my wife Paula, I thank you for your patience, understanding and unconditional support in the face of the inevitable sacrifices and changes that this project has imposed on our family life. To them I owe not only the serenity that I have so often lacked, but also the energy and motivation to continue in the most difficult moments. To my Mother†, who is no longer present, I leave the testimony of my eternal gratitude for the cognitive and human heritage that she has transmitted to me and that has given me the strength to face this challenge and carry it to the end.

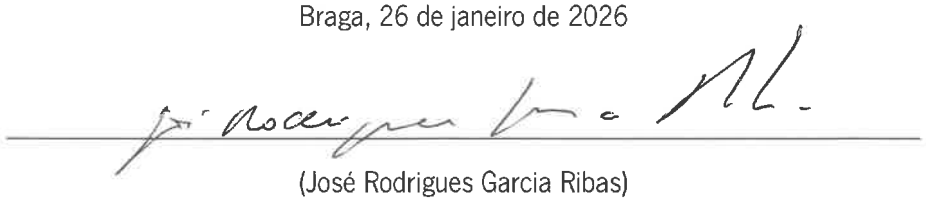
To all of them I owe the strength to never give up and the courage to continue, even in the face of the most arduous life difficulties, faced in parallel with the academic path and the personal demands that accompanied it. This work is, therefore, also the reflection of a collective effort, even if conducted in the solitude of research, and an expression of recognition for all those who, directly or indirectly, contributed to making it possible to complete this stage.

Statement Of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidad do Minho.

Braga, 26 de janeiro de 2026



(José Rodrigues Garcia Ribas)

Resumo

A crescente complexidade dos sistemas de informação contemporâneos, impulsionada pela interligação progressiva de dados, processos e intervenientes em múltiplas escalas, tem evidenciado limitações significativas nas abordagens tradicionais de modelação de dados. Os modelos relacionais clássicos e os paradigmas convencionais baseados em grafos revelam frequentemente dificuldades em capturar adequadamente relações de ordem superior, dependências hierárquicas e mecanismos de propagação semântica em ambientes complexos e dinâmicos. Estes desafios justificam a necessidade de modelos formais mais ricos e expressivos, capazes de representar sistemas complexos de forma coerente, escalável e semanticamente consistente. Esta tese propõe uma nova abordagem à modelação de dados complexos através da introdução de dois modelos formais inovadores: o Dígrafo Multinível (MLD) e o Hipergrafo Multinível (MLHG). Ambos foram concebidos para ultrapassar as limitações estruturais e semânticas dos paradigmas tradicionais de grafos e hipergrafos, nomeadamente no que respeita à representação de relações n-árias, ao encapsulamento semântico e à propagação de informação entre níveis hierárquicos. A investigação foi desenvolvida segundo uma metodologia de Investigação em Design Científico, integrando a análise crítica do estado da arte, a formalização matemática dos modelos, a implementação computacional num ambiente Python e Neo4j, e a validação empírica em diferentes domínios de aplicação. Os resultados demonstram que os modelos MLD e MLHG ampliam significativamente a capacidade de representação, abstração e manipulação de sistemas de elevada complexidade, preservando simultaneamente a coerência semântica e a reversibilidade estrutural. A contribuição científica traduz-se não apenas na formulação teórica rigorosa de novos modelos conceptuais, mas também na disponibilização de ferramentas computacionais que suportam a sua aplicação prática, consolidando um avanço relevante para a ciência de dados, a engenharia de sistemas e a evolução das bases de dados orientadas por grafos.

Palavras-chave: Grafos, Hipergrafos, Hipergrafos Multinível, Modelação de Sistemas Complexos e Modelos de Base de Dados.

Abstract

The increasing complexity of contemporary information systems, driven by the growing interconnection of data, processes, and actors across multiple scales, has revealed important limitations in traditional data modeling approaches. Classical relational models and conventional graph-based paradigms often struggle to adequately capture higher-order relationships, hierarchical dependencies, and semantic propagation in complex and dynamic environments. These challenges motivate the need for richer and more expressive formal models capable of representing complex systems in a coherent, scalable, and semantically consistent manner. This thesis proposes a new approach to modeling complex data through the introduction of two innovative formal models: the Multilevel Digraph (MLD) and the Multilevel Hypergraph (MLHG). Both were designed to overcome the structural and semantic limitations of traditional graph and hypergraph paradigms, specifically in the representation of n-ary relations, semantic encapsulation, and propagation between hierarchical levels. The research was developed under a methodology of Design Science Research, integrating critical analysis of the state of the art, mathematical formalization of the models, computational implementation in a Python and Neo4j environment and empirical validation in different domains. The results demonstrate that the MLD and the MLHG models significantly expand the capacity for representation, abstraction and manipulation of highly complex systems, preserving semantic coherence and structural reversibility. The scientific contribution translates not only into the rigorous theoretical formulation of new conceptual models, but also into the availability of computational tools that support their practical application, consolidating a relevant advance for data science, systems engineering and the evolution of graph-oriented databases.

Keywords: Complex Systems Modeling, Database Models, Graphs, Hypergraphs, and Multilevel Hypergraphs.

Index

1 Introduction 1

1.1 Contextualization 1

1.2 Scientific Motivation 4

1.3 Objectives of the Work 11

1.4 Research Questions and Hypotheses 13

1.5 Methodology Adopted 14

1.6 Articulation with the Thesis Objectives 16

1.6.1 Theoretical Contributions 16

1.6.2 Computational Contributions 17

1.6.3 Methodological Contributions 17

1.6.4 Application Contributions 18

1.7 Thesis Organization 19

2 Graphs and Digraphs 21

2.1 The Structure of Graphs 22

2.1.1 The Adjacency Matrix 24

2.1.2 The Incidence Matrices 25

2.1.3 Graphing graphs 25

2.1.4 Alternative Representation for digraphs 29

2.2 Data structures 35

2.2.1 ACID 40

2.2.2 BASE 41

2.2.3 CRUD Operations 42

2.3 Graph-Based Data Models 42

2.4 Graph-Based Database Models 45

2.4.1 BDOG Model for LEGO Assembly 50

2.4.2 A BDOG for a Reinforced Concrete Structure of a Building 53

2.5 The Adoption of a Graph-Based Data Model 66
2.6 Multilevel digraphs 69
2.6.1 Superior Structure in MLD 74
2.6.2 MLD Data Structure 78
2.6.3 Aggregation algorithm for MLD 83
2.6.4 CRUD Operations in MLD 84
3 Hypergraphs 87
3.1 Hypergraph Modelling 87
3.2 Definition and Characteristics of Hypergraphs 93
3.2.1 Partitioning Graphs Across Hypergraphs 96
3.2.2 Data classification through hypergraphs 98
3.3 Hypergraph Models 99
3.3.1 Uniform hypergraphs 99
3.3.2 Bi-split hypergraphs 99
3.3.3 K-Split Hypergraphs 100
3.3.4 Interval Hypergraphs 101
3.3.5 Weighted hypergraphs 107
3.3.6 Targeted hypergraphs 108
3.3.7 B-hypergraph hypergraphs 110
3.3.8 F-hypergraph hypergraphs 111
3.3.9 Stream Hypergraphs 113
3.3.10 Colored hypergraph 114
3.3.11 Reducible hypergraph 119
3.3.12 Hierarchical Hypergraphs 121
3.4 Models and Matrix Approaches 121
3.5 Current Limitations in Hypergraph Manipulation Tools 123
3.6 Concluding Remarks 124
4 Multilevel Hypergraph 126

4.1 The Structure of an MLHG 130
4.2 Features and Utility of an MLHG 146
4.2.1 Fundamental Operations 148
The Create Operation 150
The Delete operation 152
The update operation 154
The Read Operation 156
4.2.2 Implementing MLHG functions using Python 158
4.3 Relational Inheritance Process 168
4.4 Property Graphs and MLHG 171
5 Evaluation of Multilevel Hypergraphs 175
5.1 Rowing Club 176
5.1.1 Modeling the Club 178
5.1.2 A Graph-Based Database for the Club 181
5.2 Modeling a Building Project 186
5.3 Bidding in Digital Auctions 213
5.4 MLHG Evaluation and Impact 220
6 Conclusions and Future Work 223
6.1 A Summary of the Contributions 223
6.2 Methodological Integration 225
6.3 Discussion of Application Cases 227
6.4 Limitations and challenges identified 230
6.5 Scientific and technological contributions 232
6.6 Practical Implications and Applicability 235
6.7 Future Research 237
6.8 Final considerations 238

List of Figures

Figure 1 – Oriented graph (digraph). 24

Figure 2 - Incidence Matrix Referring to the graph of Figure 1. 25

Figure 3 – Representation of a graph using the American method. 26

Figure 4 – Representation of a graph using the French method. 27

Figure 5 – Example of decomposition of a graph graphed by the French method - figure adapted from (Eppstein and Vazirani 2019). 28

Figure 6 - Graphical representation of the digraphs example, according to the American method, adapted (Ribas 2015). 33

Figure 7 - The representation of entities and relationships - digraph lettered. 37

Figure 8 - BDOG conceptual schema for the construction of a swimming pool. 45

Figure 9 – Partial schema from the construction of a swimming pool model. 46

Figure 10 - BDOG graph of the pool. 48

Figure 11 - LEGO BDOG schema 51

Figure 12 - Global graph of the BDOG_LEGO 52

Figure 13 - Structural schema (3D) of the building - preliminary case study. 53

Figure 14 - Schematic of the BDOG_EBA 54

Figure 15 - Case study representation of the “Footing” entity. 56

Figure 16 – The complete graph of BDOG_EBA. 58

Figure 17 - Consultations at the BDOG_EBA of the previous case study. 59

Figure 18 - Group 2 (previous case study). 62

Figure 19 - Group 1 (previous case study). 64

Figure 20 - Queries of nodes with the property of Group = "1". 64

Figure 21 - Queries of nodes with the property of Group = "2". 65

Figure 22 - Reference intrinsic to all elements that interfere with the P1 pillar 66

Figure 23 - Representative Freehand Sketch of Concept of Digraphs Multilevel, MLD. 69

Figure 24 - Compromise of logical data independence in a BDOG. 72

Figure 25 – digraphs (MLD) resulting from the process of aggregation of other MLD (Ribas 2015). 74

Figure 26 – The universal identity of abstraction. 76

Figure 27 - Formal characterization of a given subgraph (Ribas 2015) 79

Figure 28 -Subdigraphs atomic (Ribas 2015). 80

Figure 29 – Window management tool MLD. 82

Figure 30 - Different categories of hypergraph modelling methods - adapted from (Dai and Gao 2023). 87

Figure 31 - Graphical representation of the various phases of the multilevel algorithm (Karypis et al. 1999). 90

Figure 32 - Different ways of selecting branches and reducing the same graph (Moretti et al. n.d.) 91

Figure 33 - Example of a hypergraph and its incidence matrix. 94

Figure 34 - Hypergraph representation using extra nodes. 94

Figure 35 - Hypergraph modeling scientific collaboration 95

Figure 36 - Example of a graph G=(V, E). 96

Figure 37 - Example of partitioning the graph G. 97

Figure 38 - A hierarchical WBS for planning a two-storey building construction project. 106

Figure 39 – An example of a directed hypergraph H = (V,E). 109

Figure 40 – An incidence matrix of a directed hypergraph. 109

Figure 41 – An example of a B-hypergraph hypergraph. 110

Figure 42 – A F-hypergraph. 112

Figure 43 - Comparison between the classical flow formulation in graphs and the proposed flow formulation in hypergraphs, illustrating the cut structure and flow representation (adapted from Cambini, Gallo and Scutellà 1997). 113

Figure 44 – A colored hypergraph model – adapted from (H. Zhang et al. 2016). 115

Figure 45 - Color-based representation of resources and associated competencies in the coloured hypergraph model. 117

Figure 46 - Gantt chart of a construction project. 117

Figure 47 - Resource allocation tree. 118

Figure 48 - Colored hypergraph of civil construction. 118

Figure 49 - An example of a hierarchical hypergraph. 121

Figure 50 - Methods for the topological transformation of hypergraphs (adapted from Dai and Gao 2023). 127

Figure 51 – An outline draft of the initial structure of the MLHG. 129

Figure 52 – A graphical representation of a generic MLHG. 131

Figure 53. A logical level structure tree ($A_{i,n}$) of an MLHG. 132

Figure 54 – An adjacency matrix of the tree structure of an MLHG. 133

Figure 55 – Showing the inheritance process between consecutive levels. 134

Figure 56 - Multilevel Hypergraph, MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$)..... 137

Figure 57 – Madj (G) matrix, where G is labeled by V_1..... 140

Figure 58 - Madj Matrices(), at level 3. 140

Figure 59 - Processing of the 3 steps, at level 2, for the virtual vertex.$V1,1 *$ 141

Figure 60 - Processing at virtual vertex level 3 $V1,5 *$. 142

Figure 61 - Result of the Madj_MLHG in level 4. 143

Figure 62 - Adjacency Matrices of level 2, 3 and 4 - MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$). 144

Figure 63 - Subtrees, A'$_{i,n}$' of MLHG ($V_{1.4}$, $A_{1.4}$, $V_{i.4}$*, $E_{i.4}$). 145

Figure 64 - A generic model of a MLHG. 146

Figure 65 - An MLHG model for a civil construction project. 148

Figure 66 - Creating a simple vertex. 151

Figure 67 - Excluding the virtual vertex "Foundations*". 153

Figure 68 - Updating the structure of an MLHG. 156

Figure 69 - Node labels pointing to the level of each parent. 160

Figure 70 - Visualization of the different subtrees present in the labels of an MLHG..... 162

Figure 71 - Level 2 graph. 164

Figure 72 – Example for the main outputs of the generate_adjacency_matrix_and_dict function..... 164

Figure 73 – RI Process. 170

Figure 74 - Illustration of the club management MLHG. 179

Figure 75 - The adjacency matrix of the MLHG. 180

Figure 76 - The schema for the club's graph-based database. 182

Figure 77 - Logical structure of the club's levels. 183

Figure 78 - The adjacency matrix of an instance. 183

Figure 79 - Aggregate subgraph of support for instantiation. 185

Figure 80 - Aggregate subgraph of participants' support for the competition. 185

Figure 81 - Initial MLHG configuration. 188

Figure 82 – The final MLHG for the propagated structure. 189

Figure 83 - Schema for BDOG of a civil construction contract. 194

Figure 84 - Level 1 graph. 197

Figure 85 - Level 2 graph - "Foundations" (node '1.1*'). 201

Figure 86 - Level 3 graph "Site Layout" (node '1.1.5*'). 202

Figure 87 - Site Layout entity (node '1.1.5*') of level 3, as constituent of the foundation node of level 2. 203

Figure 88 - Graph resulting from the total integration Intra levels of the "Foundations" node. 208

Figure 89 – The structure of an MLHG instance transformed into a graph. 210

Figure 90 - Evolution of the complex structure of an MLHG. 211

Figure 91 - The level tree (Ai,n) of the auctioneer application. 215

Figure 92 - The MLHG of the auctioneer. 217

Figure 93 - The auctioneer MLHG configuration after the application of the heritage process. 219

List of Tables

Table 1: Data for the construction of the graph example. 26

Table 2 - Entities considered in the BDOG for the construction of a swimming pool. 49

Table 3 - Comparison between other most popular database models (Angles and Gutierrez 2008) 67

Table 4 - Binary incidence matrix of the interval hypergraph. 103

Table 5 - Different types of hypergraphs considered..... 122

Table 6 - The level tree and the complete sequence of formation of an example, MLHG($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$). 138

Table 7 - Final training sequence from level 2 to level 4 of the MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$). 139

Table 8. Matrix of adjacency of the graph presented in Figure 64a. 147

Table 9 – Functions that process and analyze the structure of an MLHG. 158

Table 10 - Obtaining a practical result from the build_matrix_with_dict function. 166

Table 11 - Types of boats. 179

Table 12 - Relationship definition. 184

Table 13 - Characterization of an MLHG hierarchy. 214

Table 14 - The Adjacency Matrix of the MLHG. 216

Abbreviations and Acronyms

ACID	–	Atomicity, Consistency, Isolation, Durability
API	–	Application Programming Interface
BASE	–	Basic Availability, Soft State, and Eventually Consistent
BDOD	–	Digraph-Oriented Databases
BDOG	–	Graph-Oriented Databases
BIM	–	Building Information Modelling
CPM	–	Critical Path Method
CRUD	–	operations (Create, Read, Update, Delete)
DB	–	Databases
DBMS	–	Database Management System
DSR	–	Design Science Research
GFD	–	Graph Functional Dependencies
GNN	–	Neural Network Algorithms
LPA	–	Label Propagation Algorithm
MLD	–	Multilevel digraph
MLHG	–	Multilevel Hypergraph
NoSQL	–	Not Only SQL
OOP	–	Object-Oriented Programming
RI	–	Relational Inheritance
SBD	–	Database System
SQL	–	Structured Query Language
TDA	–	Abstract Data Type
WBS	–	Work Breakdown Structure
Web	–	World Wide Web

1 Introduction

Over the past few decades, we have witnessed an exponential growth in the complexity of information systems, both in structural and functional terms. This evolution has driven a continuous search for more expressive data models, capable of representing multifaceted relationships, multiple levels of abstraction and systemic interdependencies in a comprehensive way. The traditional relational approach, although widely used, proves to be limited when confronted with the requirements of complex, distributed and dynamic systems. In this context, graph-driven approaches have gained prominence by offering a more natural and flexible representation of entities and their interactions in an organic way.

The present work lies at the confluence between the conceptual modeling of complex systems and the evolution of database paradigms. This work proposes and develops a new theoretical and computational structure called *Multilevel Hypergraphs* (MLHG), which, articulated with *the Multilevel Digraph* (MLD), aims to overcome limitations identified in traditional and contemporary approaches to modeling by offering increased flexibility. This work is based on the premise that the integration between mathematical formalism and computational application can lead to innovative solutions in the field of hierarchical and highly complex data representation in a comprehensive way. In this first chapter, the context and motivations that gave rise to this work are introduced, as well as the scientific objectives are stated and the methodology that will support the development of the models and their empirical validation throughout the remaining chapters is presented in detail.

1.1 Contextualization

The traditional paradigm of relational databases has significant limitations in the face of the increasing complexity and volume of modern data. Information that is highly interconnected, semi-structured, or not organized according to a fixed schema is not easily accommodated in classical relational models in a rigid way. This is because the relational model requires an inflexible predefinition of schemas and relationships between tables, which creates a barrier when data does not obey previously known structures. With the exponential increase in the amount of data available and users generating and manipulating information,

the limitations of the *Structured Query Language* (SQL) and the relational model in complex data scenarios have become evident. Such a scenario has boosted the search for new alternatives that are more suitable for dealing with highly interconnected data. These challenges have paved the way for the emergence of non-relational databases, designed not to replace the relational model, but to complement it in applications where it proves to be inefficient - for example, large-scale *World Wide Web* (Web) applications and datasets without fixed schemas - although in some cases it is necessary to sacrifice consistency and concurrency control.

Graph theory has emerged as a fundamental tool for modeling complex relationships in a variety of fields, from computer science to systems engineering. Simple graphs represent entities such as vertices and connections such as edges. Although powerful in many contexts, simple graphs have inherent limitations when dealing with complex domains, since they lack mechanisms to capture higher-order relationships, i.e., interactions involving more than two elements and multilevel hierarchical structures. As a result, graph-only modeling approaches fail to adequately represent systems that require logical segmentation by layers or multiple levels of detail. For example, in an engineering project with tasks and subtasks, or in an organizational network with departments and subunits, a single flat graph becomes insufficient to distinguish distinct levels of abstraction. This limitation motivates the need to evolve towards models that explicitly incorporate notions of hierarchy and complex relationships.

Given the limitations, *graph-oriented database* (GDB) has become prominent as a natural alternative to deal with highly interconnected data. GDB are a category of *Not Only SQL* (NoSQL) database in which data is stored as vertices (nodes) and the relationships between them as edges (connections) of a graph. The main difference from the relational model is the emphasis on connections. While the relational model favors attributes and tables, the graph-oriented model deals with the connections between entities. This change allows exploratory queries of complex paths and neighborhoods to be carried out more directly and efficiently. In fact, a graph database can be formally defined as a system in which both the schema and the instances of the data are represented in the form of graphs (or their generalizations), and in which the manipulation operations follow principles inherent in the graph structure. In other words, the data model of the database itself is a graph, dispensing with complex conversions and multiple joins to recover deep relationships between entities (Angles and Gutierrez 2008).

Historically, graph-based data models have been investigated for decades, with the first papers appearing in the 1970s and a peak of interest in the 1990s. After a minor interregnum, the advent of the

NoSQL movement resurrected and revitalized these ideas, leading to the development of several graph-oriented database platforms. These systems have demonstrated great flexibility and scalability, adapting to large volumes of information and constantly evolving schemas. It is recognized by the community that graph-oriented databases achieve high performance in applications where data relationships are critical – for example, complex queries involving multiple junctions or deep branches of graphs can be answered more efficiently in these systems than in relational database management systems. Reported, for example, significant performance gains when applying GDB in scenarios with intensive consultations on relationship ties. Unfortunately, despite progress, graph-oriented databases do not eliminate the modeling challenges of complex systems. One problem persists in maintaining the logical independence of data in the face of structural changes. In relational databases, logical independence is achieved by separating conceptual schemas from applications. In GDB, any change in the structure of the graph, for example, reorganizing vertices or edges to reflect a new reality of the domain, can be difficult to manage without impacting applications, given the strong link between data and structure.

In essence, the flexibility of a graph-oriented model is still limited by its flat nature – there is no mechanism in the data model pattern to group or abstract parts of the graph into larger hierarchical units, while keeping the internal relationships encapsulated. Thus, questions of hierarchy remain open in traditional GDB, motivating investigations that seek to incorporate new levels of abstraction into these models.

The complexity of real-world systems often emerges through multiple levels of organization and complex relationships between their components. If we consider a social network, for example, in addition to simple binary relationships (friendships between pairs of individuals), there may be group interactions (communities, interest groups), which involve several participants at the same time, characterizing n-ary relationships. Similarly, in an engineering project, tasks can be grouped into larger phases or modules, establishing a natural hierarchy, in which each level aggregates a set of elements from the next level below. So far, none of the models analyzed is able to fully deal with these characteristics: conventional graphs model only binary relationships, ignoring group interactions. Both the relational model and the plane graph model lack native support for defining multilevel hierarchies. This gap has been the subject of recent research in the field of Data Science and Complex Network Theory. Some studies have shown that representations that consider higher-order interactions (hyperedges that connect multiple nodes simultaneously) and layered structures provide a more faithful mapping of diverse complex phenomena. For example, they discuss how hypergraphs capture collective dynamics that do not emerge in traditional

graphs, highlighting the importance of structures beyond the vertex-edge pair to properly model social and biological systems. Hypergraphs emerge as a generalized mathematical solution that overcomes the limitation of conventional graphs by introducing hyperedges capable of simultaneously relating arbitrary sets of vertices. This generalization is fundamental when the interdependencies are not restricted to pairs of entities, but involve entire groups or interconnected subsystems, for example, explored applications of hypergraphs exactly in this sense, evidencing gains in the representation of scenarios in which several elements interact together and need to be considered as a unified relationship (Y. Zhang, Lucas, and Battiston 2023, Lotito et al. 2024 and Praggastis et al. 2024a).

1.2 Scientific Motivation

The inherent complexity of systems has always forced models to evolve, recognizing previously neglected strata of organization. In the so-called classical graphs, the vertices connect in single planes, without indicating whether they internally aggregate structures. However, domains require layered views: an element can represent a subnet at a lower level, forming a nested architecture. In a large-scale project, for example, a phase can unfold into specific tasks and their interdependencies. Conceptually, this process of detailing or abstraction in layers can be repeated at multiple theoretical levels. Herbert Simon (1969) argued that facing high complexity requires a change in the analytical paradigm, replacing reductionist perspectives with approaches that recognize the organization and relationships between entities in different strata. In real problems, it is necessary to find the optimal point of abstraction, the hierarchical limit level to stabilize the model and focus the analysis that is intended to be carried out. Therefore, identifying the need for structures implies simultaneously recognizing the intrinsic complexity of systems and assuming an epistemological choice of representation: instead of reducing all details to a single flat level, one explicitly admits the existence of layers, each encapsulating details that may be irrelevant at the moment or that must be treated in aggregate form at higher levels.

Another aspect is the sophisticated relational semantics that such models can provide. Introducing hierarchies and allowing vertices or edges to represent complete substructures opens up the possibility of embedding semantics internally into these elements. This means that a vertex at a given top level is not just an abstract identifier, but an element that can carry aggregate information about the structure it encapsulates, such as inherited attributes or summary indicators of a substructure. Similarly, relationships between elements of different levels can follow *relational inheritance* (RI) rules, such as

dependencies or constraints defined at one level that propagate or specialize in the associated lower levels. The ability to formally define these inheritances and dependencies at multiple levels configures a richer relational semantics than available in conventional models, bringing the computational representation closer to the complexity of the real domain in a more precise way. In summary, the search for multifaceted structures with advanced semantics arises from the need to improve the alignment of the data model to the stratified and complex nature of the systems, in order to offer more expressive representations and more important operations of consultation and manipulation on the data.

It is in this challenging context, in which data is deeply intertwined, and relationships are complex at multiple levels, that innovative MLHG and MLD models emerge. These conceptual frameworks emerge as creative responses to the identified boundaries, extending classical graph frameworks to be able to natively support intricate hierarchies and associations. The main motivation for creating an MLHG, for example, lies in the need to overcome the limitations of usual data structures when dealing with vast and interconnected information. In domains such as Civil Engineering and Project Management, in which resources and entities interact widely and intricately, it has become urgent to have a model capable of representing different levels of detail, maintaining relational coherence between them.

The hierarchical structure of an MLHG can be understood as a hierarchical extension of classical hypergraphs, which incorporates explicit mechanisms of aggregation at different levels, inheritance of relationships, and internal semantics of elements. In an MLHG, the vertices and hyperedges are not just discrete units. These are elements that can contain other entire hypergraphs at lower levels, in which each level provides an aggregate panorama of the level below. This design provides a natural bridge between the macro and micro view of a system – minute details are encapsulated in the lower levels, while the upper ones provide a summarized but accessible representation of those structures. Fundamentally, an MLHG preserves and propagates connections across levels: relationships established at a given level reflect (or inherit) links between elements at the corresponding internal levels, ensuring relational coherence throughout the hierarchy. This property meets the need for semantic consistency across the aggregation layers—a meaningful association between two high-level items, for example, must correspond to a coherent set of interactions between their internal constituents. By formalizing an MLHG, this research defines operators and properties that make this maintenance of coherence possible. In this way, an MLHG functions as a high-level conceptual model for databases and complex systems, being a generalized structure in which conventional graphs and hypergraphs are particular cases (those without hierarchy). It is worth noting that the introduction of the MLHG formalism was a pioneer in bringing

hierarchical semantics to the heart of conceptual data modeling, demonstrating how *operations (Create, Read, Update, Delete)* (CRUD) can be performed at multiple levels in a prototype database (Ribas and Belo 2025b).

An MLD can be understood as a multilevel generalization of a directed graph, such that any edge of the graph can itself represent an entire encapsulated subgraph. That is, MLD introduces hierarchical levels in directed graphs, in which a high-level edge acts as a "portal" to a lower-level graph, detailing internal relationships corresponding to that abstract higher-level connection. This structure allows for formal and controlled segmentation of complex directed networks into nested modules, each module encapsulating a set of directed interactions (a sub-digraph) that can, in turn, contain other submodules, and so on. The concept of MLD is rooted in the practical need to represent hierarchical precedence and dependencies in project schedules. Traditional planning approaches, such as the *Work Breakdown Structure* (WBS), organize activities into hierarchies, but often lose track of interdependence between activities from different blocks. To solve this limitation, holocode was initially proposed, a formal notation designed to represent acyclic digraphs in a modular and scalable way. MLD expands holocode, providing a broader mathematical-computational basis that includes any hierarchical directed networks, not restricted to acyclic digraphs. Its application in project planning systems is highlighted in the context of the *Critical Path Method* (CPM), where properly capturing the hierarchy of phases, tasks, and subtasks is essential to managing deadlines, resources, and the execution of a project. With MLD, it is possible to integrate the macro view of a project – for example, large phases or deliverables – with the micro view of its component activities, keeping explicit the precedence relationships both between phases and within each phase in detail. In conceptual terms, MLD provides a way for integrated governance of directed networks. The resulting structure is not a monolithic graph, but a network of graphs, in which each level of abstraction establishes the organization of the level below. This form of hierarchical encapsulation has proven to be able to add value in domains in which complex processes need simultaneous monitoring at different granularities, from strategy to operational detail (Ribas 2015).

Both the generalized heterogeneous network and the hierarchically modular directed network represent complementary advances in the modeling of complex systems. The heterogeneous network offers an extremely adaptable model by generalizing the notion of link to relationships of any kind and allowing multiple levels of aggregation. Directed networking, on the other hand, although focused on oriented binary links, brings the benefit of hierarchical modularization in contexts where the orientation of links matters (process flows, chains of causal dependencies, etc.). In both cases, the objective is the

same: to reconcile the need for detail and a broad vision in a single model, providing conceptual and operational bases to manipulate complex structures in a coherent and effective way. The relevance of the generalized heterogeneous network and hierarchically modular directed network models becomes particularly evident when we examine real contexts in which multilevel structures and intricate relationships are inherent to the problem.

Traditionally, in the management of a large civil construction project, a hierarchy of phases and tasks and dependency graphs (such as PERT or CPM) are used for project planning. However, these representations fail to holistically capture the problem: the WBS shows hierarchical decomposition without fully explaining all the interdependencies between tasks in different branches, while the network diagram of activities presents dependencies without a clear organization. A hypergraph or MLD model can unify these two perspectives, representing phases, subprojects, tasks, and milestones in a single hierarchical graph. For example, an upper vertex can denote the completion of the structure of a building, which internally unfolds into subgraphs that illustrate activities such as the execution of foundations, columns, beams and slabs, with their respective technical connections.

On digital platforms and corporate applications, we are often faced with multiple levels of information and interaction. An illustrative case is found in e-commerce platforms or online auctions. Imagine a digital auction house in which several auctions take place simultaneously, each involving multiple items, bids, and participants. There is a global level (the entire platform, with all auctions in progress), intermediate levels (each specific auction), and local levels (items and users interacting in each auction). A multi-level hypergraph can map this reality: at the top level, each node could represent an active auction; The hyperedge connecting an auction to multiple participants and items would represent the joint relationship "participation in the auction". When going down a level, such a hypergraph "opens" into a sub-hypergraph detailing all the moves, times and interactions of that event. This macro-micro framework allows for broad or focused analysis as needed – from the overall health of the platform (how many active auctions, which aggregated interactions) to the detailed bid history of a specific item. In fact, it has been shown that multi-level hypergraphs provide great flexibility in this type of scenario: one can switch between a macro perspective, considering all auctions as an aggregate, and a micro perspective, examining the details of a particular auction, according to the analytical need of the moment. This ability to navigate between levels is important for managing dynamic and complex systems. In this case, managers can detect global patterns (e.g., identify which auctions attract the most competitive) without getting lost in the details, while also being able to quickly dig deeper into any problematic auctions to

inspect their internal transactions. Social media platforms, marketplaces, and business workflow systems have a similar structure, combining global and local interactions, and can equally take advantage of multi-level models to organize their data semantically richly. It should be noted that this scenario of electronic auctions was one of the case studies applied within the scope of this PhD, serving to validate the adequacy of the proposed model.

The application contexts for MLHG and MLD range from classical engineering problems (project planning, product modeling) to modern information systems and network science. In them, the common characteristic is the presence of complex relationship structures, often organized in a hierarchical or multi-scalar way, for which conventional models are inadequate. By demonstrating examples in these diverse scenarios, this research seeks to highlight the versatility and practical usefulness of the proposed models. The ability to represent data and knowledge more closely to reality (respecting both group interactions and levels of aggregation) paves the way for more important computational solutions, whether in databases capable of multilevel queries or in analysis algorithms that consider multiscale dynamics. Thus, MLHG and MLD are not limited to theoretical contributions, but are configured as tools applicable to real problems, with the potential to improve the way we design information systems and analyze complex systems.

The present work, seeking to understand complexity in a holistic way. We were inspired by Simon (1969), for whom modeling is fundamental to understanding complex systems. Instead of dismembering them into isolated parts, we seek to explicitly represent the interconnections and layers that constitute them, since it is precisely the links that often determine emergent behaviors. We recognize that ignoring levels of abstraction limits the understanding of the richness of these systems, a lesson reinforced over time. In this way, we have adopted a multi-level approach, in which each element can be seen as a whole and part of a larger whole, echoing the concept of holon[1]. Such an epistemological stance is reflected in the design of the MLHG and MLD: we assume that any phenomenon or modeled entity may require descriptions on more than one scale, and it is essential to provide formal mechanisms for this.

The work is thus in keeping with the tradition of mathematical abstraction applied to real-world problems. Graphs and hypergraphs are taken here not only as computational structures with the idea that computational modeling is the basis for solving complex problems. In this sense, the proposal of a MLHG that we propose in this thesis can be seen as a conceptual artifact developed according to the principles of *Design Science Research* (DSR), in which a solution (a new data model) is created to address

[1] A holon is an entity that is simultaneously a whole and a part of a larger system.

a relevant problem (the lack of adequate representations for certain types of structural complexity). The epistemological position here is twofold: on the one hand, constructivist, as we have built a new way of representing and interacting with data; on the other hand, pragmatic, because the validation of this construction is due to its explanatory and functional utility in the chosen domains. By formalizing MLHG and MLD, we are essentially proposing a new formal vocabulary for describing systems—a vocabulary that extends that available in traditional graphs and relational models. This expands the ontology of concepts with which researchers and practitioners can articulate complex problems, bringing terms such as "virtual vertex" (vertex that encapsulates an internal graph) or "RI" into the lexicon of data modeling.

In this work we adopted a theoretical approach to define these models, axiomatizing their properties, limits and operations, while we sought to connect theory with practice. This stance follows the epistemological line that knowledge in computing - especially in subareas such as databases and information systems - arises both from the formal foundation of models and from the empirical demonstration of their application. The MLHG and MLD models were designed based on existing foundations (graph theory, hypergraphs, hierarchies, etc.) but reorganized in an original way to account for new phenomena. Thus, this work contributes to the theoretical-scientific body by presenting an unprecedented structure and, simultaneously, is part of a larger debate on how to better represent complex knowledge. Epistemologically, we recognize that any model is an abstraction from reality—in the words of George Box, "all models are wrong, but some are useful." (Box, 1979). The main objective was to increase the usefulness of data models in the context of complex systems, incorporating into them additional layers of meaning (hierarchy and multi-relationship) that make them closer to the "real" that one wishes to represent.

Several isolated investigations have addressed the topic over the last few decades. However, we present here an integrative approach that articulates these perspectives in a new light. We propose a paradigmatic shift in the way we model data, highlighting not individual entities, but relationships and levels of complexity. In this way, allying ourselves with a systemic view of knowledge where understanding a whole implies considering its constituent parts and interrelations at multiple scales. This guiding belief permeated both our scientific motivation and methodological choices throughout the research. Finally, by contextualizing and justifying our motives in the light of this reorientation, we make it clear that our contribution intends to go beyond the technical, bringing data descriptions closer to the richness and complexity of real phenomena, thus broadening the horizons of what can be represented and inferred from models.

In recent years, the continuous advancement of information technologies, combined with the exponential growth of data sources and the growing interactions between entities, has revealed the structural challenges of classic database models. The relational model, dominant since the 70s, has been demonstrating limitations to represent complex high-order relationships, especially in domains that require multifaceted hierarchies, dynamic semantic aggregations, and simultaneous interactions between multiple objects. Its table-based abstractions impose a rigid and fragmented structure, making it difficult to model realities composed of several levels of abstraction or densely connected networks. In particular, the representation of n-ary relations often requires the use of auxiliary relations and composite keys that reduce semantic clarity and compromise the scalability of the database. As a partial response to these limitations, graph-oriented models and graph-oriented databases such as Neo4J[2], ArangoDB[3], OrientDB,[4] or TigerGraph[5] have emerged, which offer a more flexible approach to model nonlinear, cyclical, or recursive relationships. These technologies enable native representations for entities and relationships across nodes and edges, as well as specific query languages that enable traverse graphs with greater expressiveness than traditional SQL (Robinson, Webber, and Eifrem 2015a). However, even the most modern graph-oriented databases remain grounded in binary semantics (vertex-edge-vertex), which represents a substantial limitation for modeling higher-order interactions. Attempts to simulate n-ary relations in these models require the use of artificial constructions, such as "junction nodes" or "abstract entities", which compromise the naturalness of the representation, increase the complexity of the structure, and make it difficult to read and maintain the data.

These approaches do not natively support semantic hierarchization or structural encapsulation between levels of abstraction, which limit their application in systems that require multifaceted modeling, such as that required by systems involving tasks and subtasks, components and subsets, or complex processes with cross-dependencies. In domains such as Bioinformatics, Civil Construction, Systems Engineering or Project Management, this gap is particularly critical, as entities operate in hierarchical contexts with dense, overlapping and dynamic connections. Given this scenario, the need for new approaches that overcome the constraints of existing models becomes evident, offering conceptual and computational structures capable of representing, manipulating and analyzing complex systems with greater expressiveness and coherence. The proposal presented in this thesis – centered on the

[2] https://neo4j.com/
[3] https://arangodb.com/
[4] https://orientdb.dev/
[5] https://www.tigergraph.com/

development and application of MLD and MLHG models – emerges as a response to these structural gaps, proposing a new class of functional abstractions capable of capturing high-order relationships, connections between layers and specialized query and update operations.

1.3 Objectives of the Work

Based on the scientific motivations and research gaps identified above, the overall objective of this work is defined as follows: to propose, formalize and validate two innovative models of representation of complex *systems* – MLD and MLHG – as alternatives to classical graphs and hypergraphs in data modeling. These models aim to incorporate hierarchical abstraction, RI, and internal semantics into graph structures, to allow multiscale representations and efficient computational manipulations of complex information. In general terms, we sought to overcome limitations of relational structures and traditional graphs when dealing with phenomena of multiple levels of detail, ensuring greater expressiveness and multilevel CRUD capacity – creation, reading, updating and removal in multiple layers – in graph-oriented databases. To achieve the general objective, a set of more specific objectives were outlined, each of them associated with a concrete challenge of modeling complex systems, namely:

- To formalize the MLHG model by developing its theoretical foundations and extending classical hypergraph theory through the introduction of hierarchical levels, relationship inheritance mechanisms, and internally defined semantics for vertices and aggregated edges. This objective addressed the lack of conceptual models capable of representing multi-level hierarchies in an integrated way in complex systems, providing a basis that allows capturing n-ary relationships at different scales of abstraction. In summary, we sought to establish MLHG as a robust formalism that presents *intrinsic hierarchical semantics*, overcoming the limitations of traditional graphs and hypergraphs in the representation of complex data.
- To define and structure the MLD model, to conceive of MLD as an extension of classical digraphs, incorporating the hierarchical encapsulation of targeted subgraphs within top-level graphs. This goal aimed to overcome the limitations of traditional digraphs in representing information flows distributed over multiple levels and complex directional dependencies. Specifically, MLD should enable modular segmentation of targeted networks, being able to model hierarchical structures of processes and workflows (e.g., in *Critical Path Method planning methodologies*) that involve multiple layers of interdependent decisions. With this structure, it is expected to offer an

innovative paradigm for hierarchical directed graphs, allowing organizations and dependency networks with variable granularity in a coherent and integrated way.

- To develop computational mechanisms and algorithms for MLHG and MLD to implement software prototypes and specialized algorithms that support effective manipulation of the proposed models. This includes defining multilevel CRUD operations for MLHG, as well as hierarchical aggregation and changing propagation algorithms for MLD, ensuring that encapsulated sub-graphs can be combined and updated according to defined dependencies. This specific objective aims to ensure that the MLHG and MLD structures would not remain only in the theoretical scope, but that they would be accompanied by computational tools to prove their practical feasibility and efficiency in the management of complex data. In this way, it was expected to demonstrate the scalability of the models through simulations and prototypes, showing that operations at multiple hierarchical levels can be performed consistently and with acceptable performance.
- To apply and validate MLHG and MLD in real scenarios of complex systems, to use the models developed in the modeling of concrete case studies, to evaluate their expressiveness and benefits compared to existing approaches. This validation contemplated, on the one hand, *static domains* with multiple layers of information, such as in the management of construction engineering projects, in which phases, resources and technical dependencies are hierarchically represented. On the other hand, it included *dynamic* domains in which data structures undergo frequent changes, such as dynamic data management systems, characterized by the reconfiguration of layers, multiple semantic roles and evolving dependencies. Through the application of MLHG and MLD in these contexts, it was intended to demonstrate the ability of models to represent real multiscale structures more clearly and effectively, as well as to empirically evaluate their flexibility and scalability. The results obtained showed the gains in relation to traditional modeling (such as plane graphs or relational structures), showing that the proposed multilevel extensions bring tangible advantages in the representation and manipulation of highly complex systems.

Together, these specific objectives covered a large part of the work carried out, from the theoretical construction of the MLD and MLHG models to their computational implementation and practical evaluation in complex scenarios. This comprehensive approach ensured that the overall objective was achieved with rigor, articulating the initial motivations with concrete results and contributing to the advancement of the state of the art in complex systems modeling.

1.4 Research Questions and Hypotheses

The present work aimed to overcome the current limits of data structures used in GDB, particularly regarding their ability to represent complex hierarchical relationships, composite structures, and contextual functional dynamics. The increasing adoption of graph models in Data Science, Artificial Intelligence, and Systems Modeling has exposed deficiencies in graphs, trees, or traditional relational models, in dealing with heterogeneous n-ary relationships and nested entities in a coherent and scalable manner. Several research questions (Lee et al. 2019) were formulated, each aiming to evaluate different dimensions of the efficacy and potential of the use of MLHG in the field of GDB. The following are the questions raised, as well as their answers, now that the work is finished. See then:

Q1 – *Will it be possible to conceptualize a GDB, capable of representing complex hierarchical systems, in a more expressive way than that of classical models, through the integrated MLHG structure?*

A1 - Yes. An MLHG, by supporting multiple levels of abstraction and n-ary relations with explicit semantics, increases the expressiveness and readability of models, allowing them to capture complexity without loss of formalism. (J. Ribas and Belo 2025b).

Q2 – *How can an MLD-based model contribute to functional encapsulation, semantic compression, and modular manipulation of relational substructures in a graph-oriented system?*

A2 - An MLD is designed to allow hierarchical aggregation and abstraction with semantic preservation. This structure allows optimizing query and manipulation operations, through operators such as holocode and hierarchical scanning, with significant performance gains.

Q3 – *Is the integration of MLHG and MLD effective in the simultaneous representation of static (structural) and dynamic (functional) relationships, in order to allow unified modeling of entities and flows?*

A3 – Yes. The complementarity between the two models provides a holistic approach to the conceptualization and evolution of complex data, as required in domains such as engineering, social networks, legal systems, or biomedicine. (Geng et al. 2021).

Q4 – *Will MLHG and MLD models allow tangible gains in terms of scalability, maintainability, and clarity of structure, when compared to other classical approaches in graph databases, such as Property Graph Models or RDF?*

A4 – Through experimental tests and case studies, it was estimated that the proposed models demonstrate superiority in multilevel CRUD operations, modular segmentation and semantic management of composite nodes, overcoming limitations of the typical structures of Property Graph models (Angles and Gutierrez 2008).

1.5 Methodology Adopted

The methodology of this research followed a mixed approach – theoretical, computational, and empirical – structured according to the principles of DSR (Hevner, March, Park, & Ram, 2004). This approach, with a constructivist root, aims to develop innovative artifacts to solve complex problems, based on an iterative cycle of design, implementation and evaluation (Hevner et al. 2004). In this work, we define the formal MLHG and MLD models and their implementation and validation mechanisms as artifacts. The methodological approach adopted was also inspired by principles of Knowledge Engineering and Data Science, to promote the proper articulation between theoretical formalism and computational practical application. The following are the four methodological phases that structured the investigation done:

- Phase I – **Rationale and Comparative Analysis**. In this phase, an extensive literature review was carried out on graphs, hypergraphs and hierarchical data modeling (Lara, Vangheluwe and Alfonseca 2004). Classical structures (simple and directed graphs), generalizations such as hypergraphs, and more classical proposals such as those presented by, which introduced hierarchical aggregations, were analyzed. This review also covered the domain of GDB, with special attention to the Property Graph and RDF models, and to query languages such as Cypher. The comparative analysis included visualization and manipulation tools and libraries (Neo4j, NetworkX), to evaluate their ability to represent multilevel structures. Through this analysis, some important gaps were identified, such as the absence of native support for RI, linkage between levels of abstraction, and representation of hyper relationships at multiple levels. In this phase, the exploratory objectives were met and the need for new models (MLHG and MLD) to overcome these structural and semantic deficiencies was justified (Bretto 2013a, Lara, Vangheluwe and Alfonseca 2004, Angles et al. 2018).

- Phase II – **Formalization of Work**. Based on the systematized concepts, the formal definition of the MLD and MLHG models was carried out. The MLD model was built as a hierarchical extension of a digraph, allowing us to encapsulate directed substructures at multiple levels. MLHG, in turn, allows the concept of hyperedge to be generalized to multilevel contexts, in which n-ary connections can interconnect subsets of vertices in different hierarchical layers. Each model was described as a formal tuple with well-defined semantic operators, including specialized CRUD operations, inheritance and propagation mechanisms, and support structures, such as multilevel adjacency matrices ($A_{i,n}$). The properties of semantic consistency, reversibility, encapsulation, and extensibility were demonstrated by construction and illustrated with formal examples. In this phase, the work was developed to achieve the central objective of proposition and theoretical foundation of the MLHG and MLD models (J. Ribas and Belo 2025b) .
- Phase III – **Computational Implementation.** To validate the feasibility of the models, a functional prototype in Python was developed. CRUD operations, as well as hierarchical expansion or collapse algorithms and semantic manipulation, were implemented as native methods of the MLHG and MLD classes. The integration with the Neo4j system allowed us to store, consult and visualize the proposed structures, exploring the potential of the Cypher language and the property graph model. The implementation followed an iterative approach, with progressive operator refinements as new scenarios and use cases were developed and tested. In this phase, the computational feasibility and practical application of the models were proven (Johnson and Rosebrugh 2014).
- Phase IV – **Validation of the Work Performed**. The evaluation of the work carried out, and the results achieved were carried out based on three case studies: (i) a sports organizational structure, (ii) a digital auction platform, and (iii) a civil construction engineering project. Each scenario was modeled using MLHG or MLD, with instances constructed to reflect the hierarchies and relationships of the domain. The qualitative evaluation focused on criteria such as semantic expressiveness, ease of manipulation, and ability to adapt to changes in requirements. The results demonstrated that MLHG or MLD models can efficiently and naturally represent highly complex scenarios, surpassing classical models in flexibility and readability (Gregor and Jones 2007).

1.6 Articulation with the Thesis Objectives

The methodology adopted ensures a correspondence between scientific purposes and the means used to achieve them, promoting coherence between theory, practice and evaluation – as required in the methodological frameworks of computational science and knowledge engineering. The increasing complexity of today's systems requires modeling approaches capable of capturing multilevel hierarchies and interdependencies that conventional models do not contemplate. In response to these gaps, this thesis presents a set of innovative contributions, grouped into four categories – theoretical, computational, methodological and applicational – each aligned with the research objectives and representing advances compared to the state of the art in modeling complex systems, graphs and hypergraphs. Each of these contributions is then analyzed (March and Smith 1995).

1.6.1 Theoretical Contributions

The model of an MLHG, a multilevel extension of classical hypergraphs, capable of representing complex hierarchical structures, intrinsic modularity and multi-scale interrelationships, was proposed and formalized. An MLHG introduces concepts of RI between elements of different levels and virtual vertices to encapsulate substructures, allowing a coherent representation of multilayer systems. This theoretical contribution fills a gap in literature by providing formalism with hierarchical abstraction and internal semantics, overcoming limitations of traditional models in terms of processing and visualization of complex data. The model of an MLD was also proposed, which extends conventional digraphs, introducing functional encapsulation of subdigraphs within higher-level graphs. This model allows for structural compression and modular segmentation of complex directed networks. In contexts such as project management with directed dependencies, MLD offers a first-of-its-kind hierarchical structure and graph aggregation algorithms with controlled subgraph compression, without loss of relevant information. In this thesis, the semantics of the MLHG and MLD models are defined, including multilevel manipulation operators, virtual vertices, RI rules and projection operators. This formalization allows to navigate, modify and design multilevel structures in a consistent way, filling the absence of formal operators dedicated to hierarchical graphs (Bretto 2013a) (Healy and Nikolov n.d.).

1.6.2 Computational Contributions

In the development of this thesis, data structures were designed, such as the multilevel adjacency matrix ($A_{i,n}$) and vertex semantic dictionaries, as well as propagation and compression hierarchical algorithms. These computational resources ensure the structural and operational integrity of the models, making them robust and scalable. Additionally, a functional integration with the Neo4j system was implemented, with support for multilevel CRUD operations that respect hierarchies and inheritance relationships. Each operation automatically triggers updates at higher or lower levels, allowing for transparent and efficient manipulation of hierarchical hypergraphs (J. Ribas and Belo 2025b).

1.6.3 Methodological Contributions

The research outlines a methodological framework that ensures alignment between scientific objectives and the means used to achieve them, consistent with the classical principles of DSR. The proposed process is based on the progressive definition of artifacts — models, methods, and instances — designed, applied, and evaluated iteratively, ensuring consistency between theory, practice, and empirical validation. Thus, the methodological contribution of this thesis translates into the formalization of a path that guides the translation of real phenomena into MLHG and MLD structures, ensuring that the models represent not only the static structure of the entities, but also the dynamics of the functional interdependencies, reinforcing the fidelity of the modeling.

The proposed methodology also defines a set of formal operators — projection, compression, expansion and RI — that guide the multilevel manipulation of structures, functioning as normative mechanisms that ensure logical consistency in the navigation between levels and prevent the loss of meaning during successive transformations. This formalization responds to a gap identified in the literature, where methods for hierarchical graphs tend to be *ad hoc* and difficult to generalize. Finally, the methodological framework includes analysis procedures based on matrix projections and hierarchical decomposition algorithms, complemented by specialized visualizations for the exploration of multilevel structures. These techniques increase the intelligibility of representations and allow the identification of global patterns and critical interdependencies, with direct benefits for the interpretation and exploitation of complex data.

Taken together, these methodological contributions not only structure the construction and manipulation of MLHG and MLD models but also provide a replicable guide for future research and applications in domains that require hierarchical and relational modeling. As recommended in the

methodological frameworks of computer science, the approach balances formal rigor and practical relevance, ensuring that the evolution of the models remains underpinned by criteria of scientific validity and applied utility (Gregor and Jones 2007; March and Smith 1995).

1.6.4 Application Contributions

The functional abstractions MLHG and MLD have been applied in different real-world contexts that require high capacity to represent complex relational structures, including the management of civil construction projects, dynamic electronic auctions and the sport of rowing. In these scenarios, it was demonstrated that the models simultaneously capture static relationships, such as structural dependencies or hierarchical links, and dynamic relationships, such as functional, temporal or performance flows.

In project management, the models allowed the visualization of precedence relationships, resources, and construction phases in an integrated way, facilitating the analysis of risks and critical interdependencies. In electronic auctions, they favored the hierarchical analysis of bidding strategies and the detection of interactions between multiple agents in real time, providing support for decision-making in competitive and highly variable environments. In rowing sports, the application of the models proved to be effective in representing the interactions between athletes, boats, environmental conditions and competitive results, allowing the analysis of performance at multiple levels — from the individual contribution of each athlete to the aggregate performance of the team.

In all these contexts, the approach demonstrated gains in expressiveness, by enabling the modeling of multiple levels of abstraction within the same formal structure, and scalability, by allowing hierarchical expansion or compression without loss of semantic information. From an analytical point of view, there was greater efficiency in the exploration of the data, allowing the identification of global patterns, critical dependencies and opportunities for optimization that are difficult to detect in conventional representations.

The relevance and originality of these contributions were confirmed through four peer-reviewed scientific publications, whose evaluation process provided evidence supporting the validity of the proposed models and contributed to their progressive refinement as new use cases were explored. This validation by the academic community constitutes evidence of the practical applicability of the models, demonstrating that MLHG and MLD offer an innovative and robust foundation for the next generation of complex systems modeling and analysis tools. In summary, the contributions of the thesis – theoretical, computational, methodological and applicational – constitute significant advances in graph-driven

modeling of complex systems. The combination of MLHG and MLD structures, supported by formal operators, computational implementation and empirical validation, lays a foundation for the next generation of hierarchical graph-oriented databases (J. Ribas and Belo 2025c, 2025b, 2025a).

1.7 Thesis Organization

This thesis is organized into five additional chapters, which were designed to reflect the logical and epistemological sequence of the work carried out. The structure adopts an incremental and progressive approach, guided by the principles of DSR, allowing to follow the evolution of the research from the formulation of the questions to the empirical validation of the contributions. Each chapter is intrinsically linked to one or more phases of the adopted methodology — exploratory analysis, theoretical formalization, computational implementation and practical validation — ensuring the overall coherence of the thesis. Let's see what each of these chapters includes:

- Chapter 2 – Graphs and digraphs. This chapter introduces the mathematical and computational foundations of graph theory, with emphasis on digraphs and their structural properties. It explores concepts such as encapsulation, functional composition, interlayer coupling and hierarchical aggregation, essential for the modeling of directed systems with multiple levels of abstraction. From this base, the original proposal of the MLD is presented, including its formal definition, semantic operators and algorithmic properties. The multilevel aggregation algorithm is also introduced, allowing the transformation of internal substructures into encapsulated abstract vertices, with preservation of the directed relationships. This contribution establishes an alternative paradigm to plane digraphs, enabling the efficient and organized hierarchical manipulation of complex directed networks.
- Chapter 3 – Hypergraphs and Rationale for MLHG. In this chapter we extend the discussion to the domain of hypergraph, analyzing in detail their historical evolution, structural variants (such as directed and weighted hypergraphs), and the role they play in the representation of higher-order relations (n-arias). In addition, a review of the main approaches of multilevel partitioning, modular aggregation and encapsulation of substructures applied to hypergraphs is made, highlighting their theoretical advantages, but also their practical limitations, especially regarding the lack of support for the semantic hierarchy. From this analysis, the need for a new MLHG model is founded, capable of filling the gaps identified in literature. A set of formal, computational

and semantic requirements that guide the design of the model is also defined, preparing the ground for its formal definition in the next chapter.

- Chapter 4 – Formal and Computational Development of MLHG. This part of the thesis presents the technical and innovative core of the work carried out. Here we present the proposal of the formal definition of the MLHG model, including its structural components (virtual vertices, multilevel hyperedges), consistency rules and formal hierarchical manipulation operators (inheritance, projection, encapsulation and propagation). The mathematical formalization is complemented with the computational implementation of a prototype in Python, including auxiliary structures such as adjacency matrices and semantic dictionaries. The implementation integrates with Neo4j, a *graph database management system* (DBMS), in which MLHG or MLD structures can be maintained, queried and visualized. In this chapter, the computational feasibility of the proposal is demonstrated, and its operational behavior is validated through controlled simulations.
- Chapter 5 – Case Studies and Evaluation. In this fifth part, the results of the application of the proposed models to three real case studies are presented: a sports organizational structure, a digital auction platform and a civil construction engineering project. In each scenario, MLHG and MLD models were used to represent and manipulate complex data with multiple hierarchical layers and functional interdependencies. Qualitative metrics, such as usability, readability, or semantic adaptability, are applied to evaluate the performance of models in relation to traditional approaches. This empirical evaluation highlights the benefits of multilevel modeling, such as structural compression, semantic coherence between levels and the ability to adapt to different domains.
- Chapter 6 – Conclusions and Future Work. In this final chapter, the main results of the research are summarized, reviewing the objectives initially formulated and verifying their achievement throughout the work. In addition, the contribution of the thesis in the creation of a formal and computational framework for hierarchical modeling with MLHG and MLD is reinforced, analyzing its impact on the domain of graph-based systems. It also reflects on the limitations of the work, including scalability, usability and integration challenges with legacy systems. Finally, some lines of future research are proposed, such as the generalization of MLHG to dynamic hypergraphs, the development of specific hierarchical query languages and the application of the models to emerging contexts.

2 Graphs and Digraphs

This chapter establishes the conceptual basis necessary for the in-depth understanding of graphs as structuring instruments in the representation of complex systems. More than a simple mathematical abstraction, graphs are a formal modeling language with great versatility and application in several domains — from the analysis of social networks, information flows and critical infrastructures to the hierarchical organization of projects, production chains or decision structures. In this sense, graph theory provides the foundations on which the advanced models proposed in this thesis are built, namely the MLD and the MLHG. Throughout this chapter, we seek not only to rescue the classical foundations that sustain graph theory, but, above all, to analyze its limitations when confronted with systems that present multiple levels of organization and relationships of high semantic density. Indeed, by addressing structures with distinct layers of abstraction, vertical and horizontal interdependencies, and real-time update dynamics, traditional models reveal weaknesses in terms of expressiveness, scalability, and functional encapsulation capacity. The investigation starts from this observation to justify the need for expanded and semantically enriched structures, such as those that will be formalized in the following chapters. In this framework, different forms of graph structuring in computational and symbolic space are examined, highlighting methodologies of graphic and textual representation that serve as support for their manipulation. The choice between graph methods – such as the American method or the French method – is not merely aesthetic or historical – since it translates relevant differences in the way systems are interpreted, visualized and operated. This methodological diversity is particularly critical when one intends to evolve from flat representations to hierarchical configurations, in which multiple layers of information coexist and interact. Thus, following this analysis, original contributions are introduced that derive from the research carried out, such as holocode, a type of structured coding of acyclic graphs with strong practical implications in the area of project management, which offers a compact way to represent precedence, segmentation and simultaneity relationships between tasks or events, allowing greater precision and ability to update the modeled structures. Its general functioning, articulating it with the challenges encountered in real environments, such as the planning of works or the organization of complex operational flows, will also be described in this chapter.

The discussion also extends to MLD, a model that introduces a structure to encapsulate subdigraphs within composite vertices, facilitating the modeling of modular and highly scalable systems.

By integrating this concept, the chapter transcends the limits of classical graphs and proposes a new paradigm of representation, oriented towards the logical decomposition and progressive analysis of directed networks. The MLD, by allowing the collapse and expansion of substructures based on well-defined operators, constitutes a decisive step towards the construction of a general formalism of hierarchical modeling, paving the way for the subsequent definition of the MLHG model.

Within this framework, the computational implications of these structures are also addressed, from the efficient management of data to its internal representation in *graph-oriented databases* (BDOG), namely the use of adjacency matrices and multilevel manipulation algorithms. This approach reinforces the importance of the integration between theory and practice, and between symbolic representation and computational execution, in a perspective that will be deepened in the following chapters with the formalization of the proposed models and their empirical validation. In short, this chapter constitutes an essential bridge between the foundations of graph theory and the advanced multilevel structures developed in this research. By consolidating the operational concepts, representation methodologies and original contributions that extend the capabilities of classical graphs, the ground is prepared for the introduction of MLHG, which will further expand the expressiveness and application of models in real contexts.

2.1 The Structure of Graphs

A graphical representation of a set of interconnected elements is called a graph. Formally, a graph is defined as G = (V, E), where V represents a finite set of vertices (or nodes) and E a set of edges (or bonds) between those vertices, where $E \subseteq V \times V$. Each edge (Mathew, John, and Garg 2020) is represented by an ordered pair (u, v) of distinct vertices u, v, $\in$ V, indicating that the edge leaves you and arrives at v. If the order of the vertices does not matter, the edge is considered undirected.

Distinct types of graphs exhibit unique properties and can model numerous real-world situations. In undirected graphs, the connections between vertices are bidirectional, as in a social network where users are vertices and friend requests create symmetric edges between them. The degree of a vertex quantifies the number of edges incident to it, revealing its topological importance. Grade-based indices aid simplified structural analysis (Schermann et al. 2014). This metric is essential for understanding complex networks, such as the degree to which a user reveals their social circle. In addition, degrees

influence global connectivity, with fully connected graphs, exhibiting high integration between vertices versus disconnected with isolated components, impacting applications such as logistics and modeling. An urban road network, for example, is a connected graph, in which the intersections are vertices and the connecting roads are edges, allowing anyone to reach any other. An archipelago dispersed in the sea, on the other hand, forms a disconnected graph, in which each island, a vertex, is detached from the others.

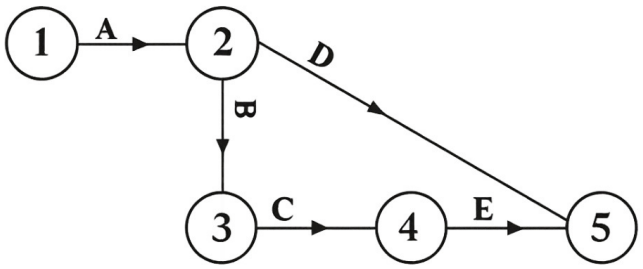
In contrast, digraphs represent one-way relationships, such as what happens on the Web. Each Web page is a point, and the links between the pages are oriented edges. When you move from page X to page Y through a link, there is not necessarily a link that ensures the return from page Y to page X, demonstrating the direction of the connections. In turn, weighted graphs (networks) arise when we assign numerical weights to edges, as happens in a complex public transport network. In these networks, the nodes are stations and the weights on the edges represent the estimated average travel time between the stations. This information is essential, for example, to calculate the fastest route between two stations. An example of a bipartite graph can be found in sophisticated movie recommendation systems. The nodes on one side represent the users, while the nodes on the other side represent the movies. Edges can indicate the ratings given by users to the movies, linking users to the movies they have reviewed. More specifically, we have that cyclic and acyclic graphs play important roles in several applications. A cyclic graph can be observed in a complex manufacturing chain, in which the production steps are the nodes and the edges indicate the order of the steps. Any loop in this graph can lead to infinite fabrication problems. On the other hand, an acyclic graph is commonly used to represent hierarchical relationships in sophisticated organizations, where the nodes are the employees and the edges indicate supervisory relationships (Stanley 1973).

There are three basic types of graphs: directed (oriented), undirected (unoriented), and mixed. In the first type, all the edges of the graph are directed. In the second, the edges are not directed. Finally, in mixed graphs, some edges are directed and others are undirected – in this work we are particularly interested in directed graphs. From now on, this type of graph will be referred to simply as (Diestel 2018) a digraph. Another pertinent aspect is to know that the computational efficiency of an algorithm to solve problems in graphs does not depend only on its intrinsic characteristics, but also, and a lot, on the data structures used to represent the graph (ways of storing and manipulating the data associated with the graph) and to store the intermediate results necessary for the algorithms. Generally, to represent a graph computationally, two types of information are needed, the topology (structure of the nodes and edges)

and the data (attributes such as time, costs, capacities, among others) associated with the nodes and edges. Normally, the schema used to store the topology of the graph will naturally suggest, without rules, a way to store the information associated with edges and nodes. The most common representations of a graph are achieved through the adjacency matrix and the incidence matrix, or alternatively, in homologous list structures, when the graphs are suitably disconnected.

2.1.1 The Adjacency Matrix

Let G be a directed graph, G = (N, A), containing N vertices (nodes) and A edges (edges), the adjacency matrix, or more precisely the node–node adjacency matrix, stores the graph, G in a matrix M={m_{ij}} square of order n, where each row and column corresponds to a vertex (node). Each element m_{ij} of the matrix takes one of the following values: 1 (if (i, j) ∈ A) or 0 (if (i, j) ∉ A). Having as an example the matrix M of Figure 1 and its corresponding (oriented) graph.



$$M=\begin{bmatrix}0&1&0&0&0\\0&0&1&0&1\\0&0&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}$$

Figure 1 – Oriented graph (digraph).

The simplicity of the graph representation through the adjacency matrix allows algorithms to be easily developed for adjacency detection, degree determination and path identification, since it is possible to determine the parameters associated with any edge (i, j), simply by taking the element (i, j) from the respective matrices, as well as easily obtaining the edges that come out of node i, examining line i. In addition, it makes it possible to obtain the edges that arrive at the node j by examining the column j - if the ith element of that column has the value one (1) then (i, j) is an edge of the graph.

2.1.2 The Incidence Matrices

Considering a digraph G = (N, A) with N vertices (nodes) and A edges, the incidence matrix B = {b_{ij}}, usually called the vertex-edge incidence matrix, stores the graph, G in a matrix consisting of i rows and j columns, where for each row a node is associated and each column j, a graph edge. Each element (i, j) takes one of the following values: 1 (if edge a leaves node i), −1 (if edge a reaches node i), or 0 (other cases).

$$B=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Figure 2 - Incidence Matrix Referring to the graph of Figure 1.

Consider matrix B presented in Figure 2. In undirected graphs, negative values (−1) are replaced by positive (+1), since every node is tail (source) and how head (sink) of a edge. However, this structure becomes quite inefficient, both in terms of search time and the memory space required for data storage, since the amount of elements reserved for the graph is (n x a) and this has only a element to be stored (or a x a if it is an undirected graph). On the other hand, the value +1 in a row is equal to the amount of edges that leave the corresponding node, whatever the type of graph. Similarly, in directed graphs, the value −1 in a row is equal to the quantity of edges that arrive at the corresponding node; consequently, the sum of (+1) and (−1), in a row, corresponds to the degree of the node associated with it.

2.1.3 Graphing graphs

In the graphical representation of graphs, two formal methods of structuring logical precedence are distinguished, namely the American and French methods. The latter stands out for its notoriety, largely due to its ability to explore identities during the precedence relationships. For a better understanding, it is pertinent to perform a careful analysis of the graphs associated with each of these methods. Based on the data presented in the Table 1, a comparative study will be conducted, exemplifying the distinctive features of both methods through a practical example. This approach will allow not only a visual

appreciation, but also a more in-depth evaluation of the implications of choosing between the American and the French method.

When examining the practical example of Figure 3, It is possible to see how the particularities of each method impact the organization and interpretation of precedence relationships. This analysis makes it easier to identify visual nuances, as well as provide consistency on how these differences can affect the overall understanding of the graph. It is important to emphasize that the choice between these methods is not limited to the visual issue because such a choice will directly influence the logical interpretation of the relationships present in the graph.

Table 1: Data for the construction of the graph example.

Ref.	**Predecessors**	**Successor**
A	—	E; B; G
B	A	C
C	B	G
D	E	—
E	A	D; F
F	E	G
G	A; C; F	—

The American method, also known as the "arrow method", is characterized by the singular representation of each entity through a single arrow (edge). This arrow extends and connects at specific events, identified as topological nodes (Figure 3). In the graph presented in this figure, in particular, the application of the principles of the American method is evidenced, in which the arrows diverge and converge at key moments, providing a visual representation that highlights the relationships and dynamics of the graph as outlined by this specific method.

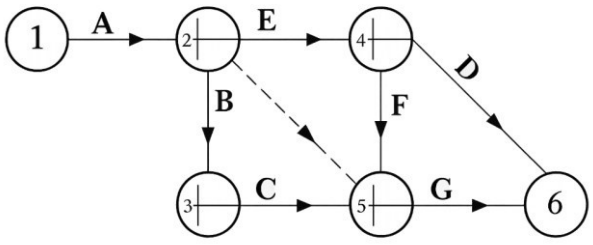


Figure 3 – Representation of a graph using the American method.

The complex structure of the graph presented in Figure 3, according to the American method, reveals an interlaced network of components with directed links. Component A, without predecessors, has as successors the components E, B and G (with inference from the fictitious component). B, in turn, has A as its predecessor and C as its successor. C, B's predecessor, has G as its successor. D, E's predecessor, has no successors. E, with A as its predecessor, it is linked to components D and F. F, the predecessor of E, is associated with component G. Finally, G, the predecessor of A, C and F, does not show successors. The American method clearly shows the meaning of the connections between the components, providing a visual representation that facilitates the analysis of dependencies and logical sequences in the network.

The French method, on the other hand, stands out as a graphic approach in which the elements are represented through blocks. The blocks represent the nodes, identified by labels, that refer to the components in question. In addition, a dynamic representation of the relationships between these components is established by means of arrows (Figure 4).

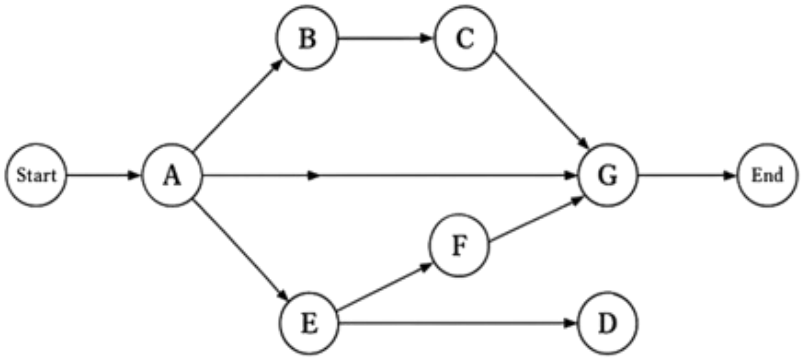


Figure 4 – Representation of a graph using the French method.

The structure by blocks and arrows directly reflects the entities and their relationships, simplifying the understanding of the flow of information. A significant advantage of this method is the elimination of the need to introduce recurring fictitious relationships, simplifying modeling and avoiding additional representations for specific situations. In addition, the visual clarity and simplicity inherent in the French method facilitate the understanding of complex relationships and interdependencies and align naturally with the graphically oriented nature of these databases. The transition between visual representation and the organization of data in graph format contributes to a cohesive integration between the graphical approach and the storage of information in graph-oriented systems. By choosing the French method for

the graphical representation of data in graphs, a proficient integration with graph-oriented databases is established. This choice also drives an integrated approach, improving understanding, accuracy, and communication in environments where proper relationship interpretation is required for the success of projects and processes, especially in systems that employ structures with intricate relationships.

French method it does not merely offer a clear and intuitive visualization of relationships but also opens the way to other analytical possibilities. Among these, the construction of Graph decomposition algorithms, as highlighted in academic works, in the studies of Eppstein and Vazirani (2019), that highlight the relevance of Clicks[6] in the representation and analysis of complex structures.

As illustrated in the Figure 5, this approach allows us to examine how specific configurations of elements are interrelated and subdivided, giving rise to more cohesive and semantically precise subsets of interconnections. By integrating the properties of clicks in this context, it becomes possible to identify dense and functionally significant substructures, which expand the ability to interpret the graph. These click-centric approaches thus provide a deeper understanding of the relationships between the entities involved and reinforce the potential of the French method as an advanced modelling tool.

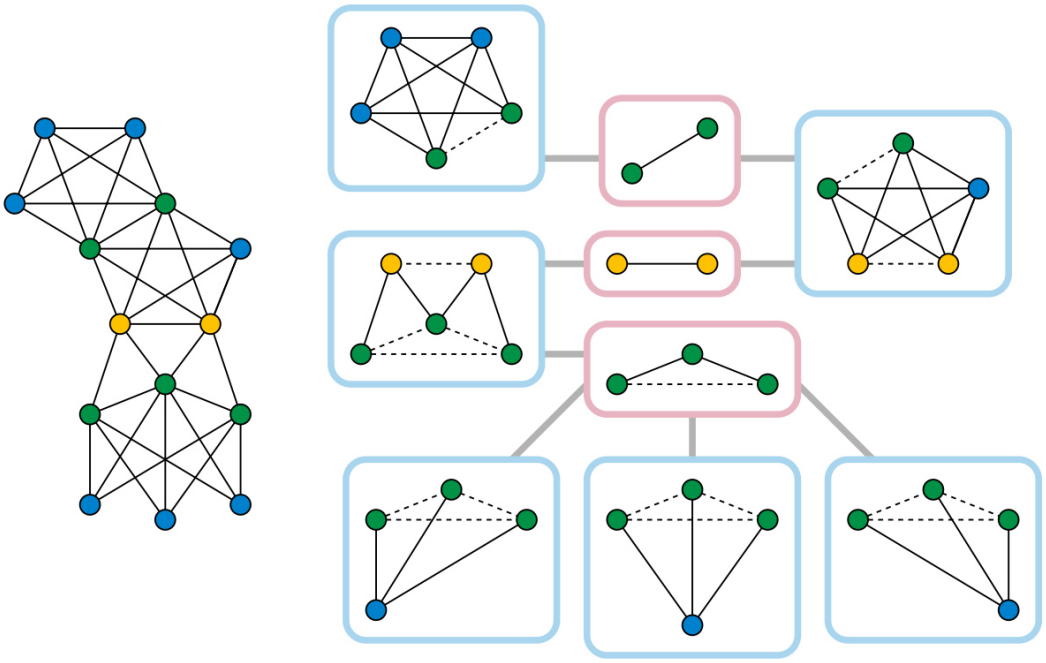


Figure 5 – Example of decomposition of a graph graphed by the French method - figure adapted from (Eppstein and Vazirani 2019).

[6] A group of vertices where each pair of vertices is directly connected by an edge.

Detailed analysis of specific element configurations provides an identification of interconnections, increasing the understanding of the relationships between entities. This choice, which goes beyond the enrichment of the approach adopted, plays a fundamental role in efficient and informed development, especially in environments that require interpretation of relationships for the success of projects and processes. By adopting the French method in the graphical representation of data in graphs, supported by relevant contributions from academic research, substantial gains are achieved in terms of comprehension, accuracy and communicational clarity. For this reason, this technical perspective will be fully integrated in the development of this thesis, serving as a methodological basis for the formalization and analysis of the proposed models.

2.1.4 Alternative Representation for digraphs

In recent years, the universe of project management and planning has become increasingly complex, with significant challenges related to schedule monitoring, resource allocation, and budget control. In a highly competitive corporate environment, the ability to make well-informed and strategic decisions is essential for the success of any venture. The research developed by Ribas (2015) on the mapping of directed graphs, acyclic digraphs, in the context of project management, allowed the development of an *abstract data type* (TDA), which was called holocode. This type of data is designed to represent and handle digraphs. This modeling, together with other techniques, serves to fulfill the initial objectives of the work presented in this thesis. Note that a TDA as defined by is a valuable tool for specifying the logical properties of a data type. Essentially, a data type is made up of a set of values and a set of operations that can be performed on those values. These components form a mathematical construct that can be implemented using a specific data structure. A TDA is usually divided into two parts: the definition of values and the definition of operations. The definition of values establishes the set of values that belong to the TDA and is composed of two parts: a specification clause that describes the values themselves, and a condition clause that establishes the limitations on those values. In this sequence, there is a brief reminder of the holocode concept that, in its essence, is configured through a single string, whose semantic structure is composed of five blocks Srivillibhuthur et al. (2018).

The general specification of the holocode is given by:

IDE; IDEP^Ee<Ff_If,i;Sf,s#Ii_Ii,j; Si,s>Ss

with the restrictions:

$$IDE \neq IDEP \neq Ee \neq Ff \neq Ii \neq SS$$

$$If,i \cup Ii,j = Ii \; e \; Sf,s \cup Si,s = Ss$$

Key elements:

IDE – Identifier of the current digraph. Represents the root node or reference unit of the current holocode. It is unique and unchanging;

IDEP – Parent digraph identifier. Indicates the connection to the higher hierarchical level, allowing the recursion of the structure. It is also unique and unchanging;

Ee – External digraph. Introduced by the ^ symbol. Refers to a subdigraph with no mandatory direct precedence. It functions as an external link;

Ff – Source digraph. Introduced by the < symbol. It represents a subdigraph from which dependencies emerge;

If,i – Set of interior digraphs associated with the source digraph Ff;

Sf,s – Set of sinking digraphs associated with the source digraph Ff. These sets are separated by the symbol; and they are distinguished with;

Ii,j – Subset of interior digraphs subordinate to Ii;

Sirs – Subset of sinking digraphs subordinate to Ii;

Ss – Sinking digraph. Introduced by the symbol >. It represents a terminal subdigraph, which does not give rise to additional precedence.

Control Symbols:

; – separator of elements within the same structure;

^ – indicates that an external digraph follows;

< – indicates that a source digraph is followed;

_ – separates identifiers from interior subgraphs and sinks;

– indicates that an inner digraph is followed;

> – indicates that a *disusing digraph is followed*.

The first block constitutes the support base and is labeled the root block, whose function is to support the remaining blocks, if they exist, and are sequenced as follows:

(Root block) IDE; IDEP

The subdigraphs in reference were created with the IDE record until it finds the symbol ";" and from there forward, until it finds some of the symbols (^) or (<) it will be related to the IDEP, which is, the record of the parent digraphs, each of which is unique and immutable in nature. Let's take a closer look at each of the blocks previously used in the holocode specification.

(1st Block) ^Ee - The symbol (^) indicates to the algorithms that the subdigraph designated by (Ee) is an external digraph, and that there will be no precedence by this element to be added to the digraphs.

(2nd Block) <Ff_If,i;Sf,s - The symbol (<) indicates to the algorithms that a sub-digraph is followed by (Ff) and that this is one of the source digraphs. The occurrence of the symbol (_) will indicate to the algorithms that the precedence of that source digraph is made up of inner digraphs, designated by If,i and by sinking digraphs, designated by Sf,s, successively separated by the symbol (;).

(3rd Block) #Ii_Ii,j; Si,s - The symbol (#) indicates to the algorithms that a subdigraph is followed by (Ii) and that this is one of the inner digraphs, followed by the appearance of the symbol (_), which will indicate that the precedence of that inner digraph is constituted by Inner digraphs designated by Ii,j and by digraphs sinks designated by Si,s, separated successively by the symbol (;).

(4th Block) >Ss - The symbol (>) indicates to the algorithms that a subdigraph is followed by (Ss) and which is one of the sinking digraphs, also indicating that there will be no precedence to add to the digraphs through this element.

It is a condition that only one of the following possible seven orderly combinations (Ci *) of those last four blocks can occur exclusively, in such a way that the integrity of the digraphs is validated, which are:

C1 * (1st) ^Ee

C2* (1°) ^Ee + (2°') <Ff_Sf,s + (4°) >Ss = ^Ee<Ff_Sf,s>Ss

C3* (1º) ^Ee + (2º) <Ff_If,i;Sf,s + (3º) #Ii_Si,s + (4º) >Ss =
^Ee<Ff_If,i;Sf,s#Ii_Si,s>Ss

C4* (1º) ^Ee + (2º) <Ff_If,i;Sf,s + (3º) #Ii_Ii,j; Si,s + (4º) >Ss =
^Ee<Ff_If,i;Sf,s#Ii_Ii,j; Si,s>Ss

C5* (2º') <Ff_Sf,s + (4º) >Ss = <Ff_Sf,s>Ss

C6* (2º) <Ff_If,i;Sf,s + (3º) #Ii_Si,s + (4º) >Ss =
<Ff_If,i;Sf,s#Ii_Si,s>Ss

C7* (2º) <Ff_If,i;Sf,s + (3º) #Ii_Ii,j; Si,s + (4º) >Ss =
<Ff_If,i;Sf,s#Ii_Ii,j; Si,s>Ss

For a more practical understanding of this TDA, the following illustrative example presents the application of a holocode to the representation of precedence networks in complex projects, enabling the identification of hierarchical dependencies between tasks, the structuring of subprojects in a modular manner, and the analysis of the critical path (Figure 6).

Example of Holocode:

[^134<2_3<22_23;86<39_40<58_59;75<97_98#3_4#4_5;108#5_6#6_7;114#7_9#9_10;116#10_11;122#11_12#12_13;124#13_14#14_15#15_16#16_17;130#17_18#18_19#19_20#20_21#23_24#24_25;109#25_26#26_27;115#27_28#28_29;117#29_30;123#30_31#31_32;125#32_33#33_34#34_35#35_36#36_37#37_38#40_41#41_42;110#42_43;112#43_44;131#44_45#45_46#46_47#47_48#48_49#49_50#50_51#51_52;128#52_53#53_54#54_55#55_56#56_57#59_60;61;111#60_43;112#61_62;113#62_63;132#63_64#64_65#65_66#66_67#67_68#68_69;121#69_70#70_71;129#71_72#72_73;74#73_38#75_76#76_77#77_78;133#78_79#79_80#80_81#81_82;120#82_83#83_84#84_85#85_38#86_87#87_88#88_89#89_90#90_91#91_92;119#92_93#93_94#94_95;127#95_96#96_38#98_99#99_100#100_101#101_102;118#102_103#103_104#104_105;126#105_106#106_107#107_37#108_25;109#109_98#110_60;61;111#111_76#112_62;113#113_77#114_27;115#115_100#116_29;117#117_102;118#118_92;119#119_82;120#120_69;121#121_52;128#122_30;123#123_103#124_32;125#125_105;126#126_95;127#127_85#128_71;129#129_73;74#130_36#131_63;132#132_78; 133#133_90>21>38>57>74]

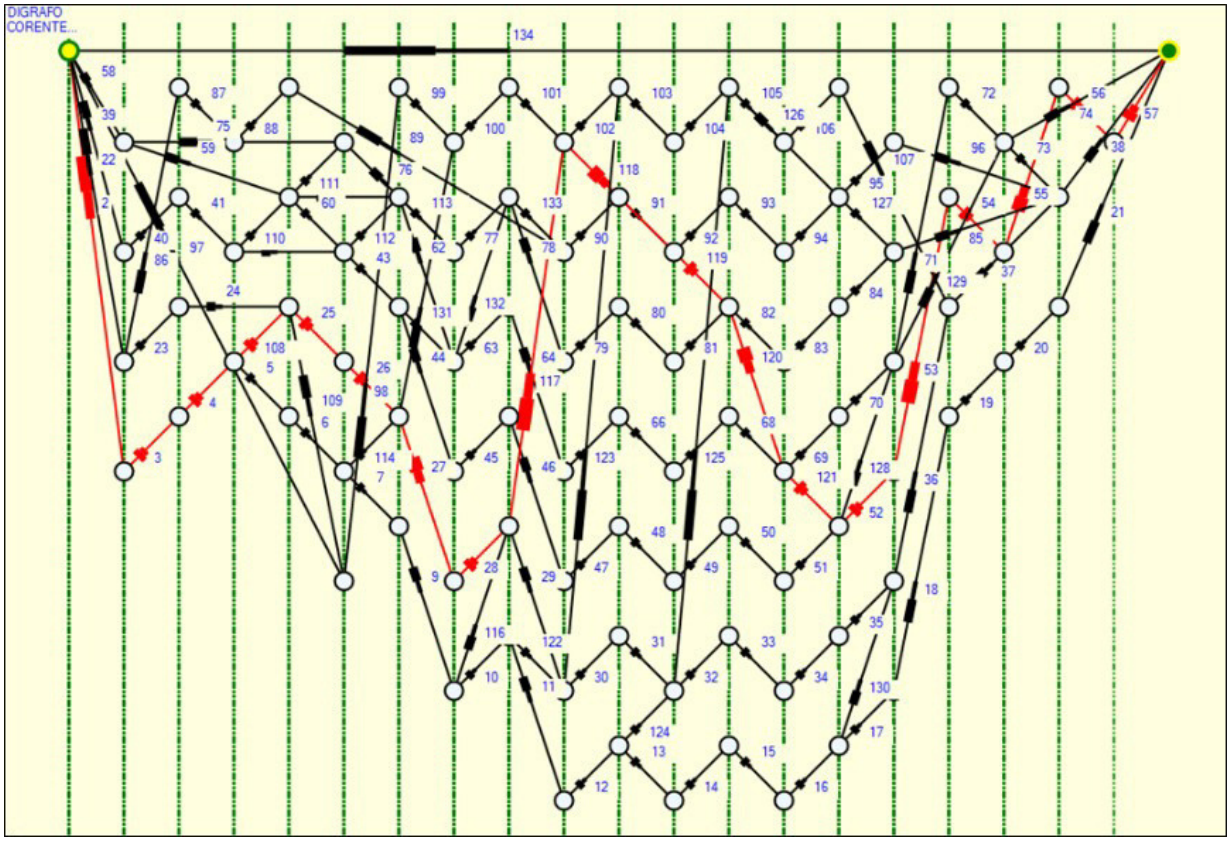


Figure 6 - Graphical representation of the digraphs example, according to the American method, adapted (Ribas 2015).

The holocode used in the example is an intricate representation of a digraph, containing detailed information about its structure and internal relationships. Each block and symbol play specific roles in the description of the digraph. The digraph begins with the symbol "[", followed by the symbol "^" which indicates the presence of an external subdigraph. The following are several blocks representing source digraphs ("<Ff_If,i;Sf,s") and interior digraphs ("#Ii_Ii,j; Si,s"), together with their respective sub-digraphs (">Ss"). Each block is separated by semicolon symbols ";". The numbers and symbols within the blocks (e.g., "2_3") indicate the connections between the vertices of the digraph, specifying where the edges are headed from. The holocode concludes with the sequence ">21>38>57>74]," which is related to identifiers, corresponding to the vertices of the graph.

This way of representing digraphs within digraphs is useful in situations where it is necessary to describe intricate systems, such as complex projects in which there are several tasks, subprojects, or components that maintain direct and indirect relationships with each other. The holocode allows you to

represent this complexity in an organized and legible way. This functionality is especially valuable in scenarios where there is an intricate hierarchical structure of relationships, such as in large projects where there are multiple layers of interconnected tasks, subprojects, or components. Holocode offers a structured method for representing this structure, ensuring that all entities are documented at any level of nesting. This means that the holocode can represent digraphs within digraphs, forming an elaborate hierarchy of associations. The holocode structure, as mentioned earlier, includes blocks thar represent external digraphs, source digraphs, internal digraphs, and collector digraphs, and these blocks can be nested within each other at different levels of abstraction. This provides a path to modeling complex relationships in systems that involve multiple levels of hierarchy and interconnections.

The *holocode*, initially developed as a representation scheme for digraphs in project planning, implicitly introduced a structuring principle that would later be formalized in the MLHG model: hierarchical semantic encapsulation. In essence, each block of the *holocode* (external, source, interior, or sink) is not limited to identifying the nature of the connection between vertices but also defines the relative position of that connection in a hierarchy of precedence. Such an approach allows a digraph to be described as a composition of interdependent sub-digraphs, in which the overall structure emerges from the organized interaction between the different blocks. This recursion gives the *holocode* a semantic dimension that goes beyond simple topological connectivity, adding a layer of hierarchical interpretation to it.

The MLHG appropriates this logic and expands it significantly. While *holocode* can encapsulate directed binary relations by structuring them at different levels of precedence, MLHG extends this principle to the universe of higher-order (n-ary) relations, in which each hyper-edge can involve multiple vertices simultaneously. In addition, MLHG allows such hyperedges to be encapsulated at different levels themselves, while maintaining semantic coherence through well-defined boundary mechanisms. In this way, the modular organization suggested by the *holocode* is transformed, in the MLHG, into a robust formal framework that supports the modeling of complex systems, in which multiple levels of abstraction, high-order interdependencies and the need for structural reversibility coexist. Thus, the holocode can be regarded as a conceptual precursor to the MLHG model: its block-based grammar paved the way for the definition of an encapsulation hierarchy that was later generalized and formalized within MLHG. This transition not only ensures continuity between the practical experience of representing projects by hierarchical digraphs and the broader theory of hypergraphs but also establishes a solid basis for the advancement of complex data modelling in domains where the multiplicity of levels and n-ary relationships is an unavoidable requirement.

2.2 Data structures

An entity represents something that exists as a complete and unique unit. A relationship is a property or predicate that establishes a connection between two or more entities. The data structures used to model entities and relationships in graph database models reflect the representation of entities and their relationships. One of the most distinctive features of a graph-based database model is a structure for representing the connectivity between entities, distinct from the attributes of the relational model, standard abstractions of semantic models, complex objects from object-oriented models, or compositional relations from semi-structured models. Graph-driven (Yang et al. 2022, Song et al. 2023) database models have as their formal foundation variations in the mathematical definition of a graph, for example, directed or undirected graphs, marked or unmarked edges and nodes, hypergraphs and hypernodes. The models have several features influenced by the semantic or object-oriented edges. The use of hypergraphs allows a concept to grow from a simple concept to one defined by multiple and complex relationships. Generally, the data of a graph is represented as sets of nodes and edges, in which the nodes denote entities and the edges establish connections between pairs of nodes, reflecting various relationships. Both nodes and edges can be characterized by labels and possess a set of properties represented as attributes, providing a rich way to describe entities and their interactions (Angles and Gutierrez 2008, Ghrab et al. 2016).

Representing a database as a graph with many nodes connected has no added advantage, because in practice it is not easy to present the information clearly. A database, which consists of a set of nested graphs, can provide each real-world object as a separate database entity with inherent support for more efficient data abstraction. Currently, many graph databases adopt the property graph model, which describes a multigraph that is simultaneously labeled and assigned. In this context, a multigraph refers to a graph that allows multiple edges between the same pairs of nodes, while labeled indicates the association of labels to nodes and edges, adding a layer of semantics to the model. Assigning properties to nodes and edges provides a flexible way to include additional information and metadata, contributing to a richer and more expressive representation (Poulovassilis and Levene 1994, Rodriguez and Neubauer 2010).

The formal definition of a property graph involves the detailed description of its essential characteristics, encompassing the multigraph structure, the labels attached to nodes and edges, and the properties assigned to these elements. This formalization process lays the foundation for a consistent understanding of the model, simplifying its implementation and analysis in a variety of contexts, ranging

from data science to complex information systems. According to the same author, (Ghrab et al. 2016) a property graph is a formal structure G = (V, E, L_v, F_v, L_e, F_e, A_v, H_v, A_e, H_e), in which V is a set of nodes, E $\subseteq V \times V$ is a set of edges, L_v is a set of node labels, and F_v: V → L_v is the function that assigns a label to each node, Le is the set of labels on the edges of the entity and F_e: E → L_e is the function that assigns a label to each edge, A_v = {a_1, a_2, ...} is a set of node attributes represented as key value pairs. Each node is associated with an attribute vector [a_1, a_2, ...], Hv is a set of n functions such that: For a node $v_j \in V$, the function $h_v(v_j)$ returns the value x of its it attributes and x ∈ gift (a_i), A_e = {b_1, b_2, ...} is a set of edge attributes represented as key value pairs. Each edge is associated with an attribute vector [b_1, b_2, ...], and H_e = {h_{1e},h_{2e}, ...} is a set of functions such that: E → dom(b_i). For an edge $e_j \in E$, the function returns the value x of its attribute i–th and x ∈ dom(b_i).

Database models-based graphs play a prominent role by representing entities at two distinct levels: instance and schema. The database schema definition covers entity types, relationship types, and schema constraints, while entity and relationship instances conform to specific database instances. The representation of entities and relationships in these models is an essential facet of the architecture of these systems, traditionally accomplished through a digraph labeled, which is a structure composed of three components (Figure 7):

- Primitive Labeled Nodes, which represent elementary or primitive information associated with specific entities in the domain of interest. For example, in a library management system, a node labeled "Title" might contain the title of a book as a primitive value. These nodes capture direct details about the entities.
- Labeled Nodes, which are used to represent the tangible entities of the real world, such as people, books, places, etc. Each node is identified with the entity type name or a unique identifier. For example, a node can be labeled "Book" and represent a specific book entity. These nodes provide an organizational structure for the entities in the system.
- Labeled Edges, which are used to represent relationships between nodes. Each edge is labeled with the name of the relationship that describes the connection between the start node and the end node connected by it. For example, an edge labeled "Written_By" might connect an "Author" node to a "Book" node, indicating the authorship relationship between them. These edges capture the interactions and connections between entities in the system.

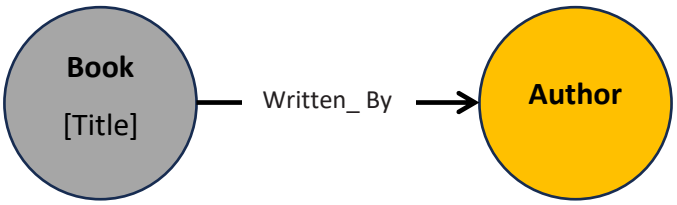


Figure 7 - The representation of entities and relationships - digraph lettered.

Representation by means of a labeled graph proves to be highly effective in modeling complex data and their interrelationships. This approach allows for hierarchical organization of data, offering insight into the connections between different entities. This structure is flexible and easily adaptable to accommodate changes in system requirements, consolidating graph database models as robust choices for various applications. One of the most prominent DBMS today is Neo4j, which is based on the Property Graph data model. However, it is essential to emphasize that, although Property Graphs provide a powerful representation framework for graph data, they do not constitute an integral database model. These frameworks are designed in a simplified manner, primarily geared toward storage and operational workloads. To meet the requirements of advanced graph modeling and analysis, it is necessary to develop a data model, oriented to graph analysis that respects the principles established by (Codd 1981). This model should overcome the basic structures, incorporating semantically richer features for the representation of complex graphs. Additionally, it is of paramount importance to integrate integrity constraints and specific manipulation operators, thus providing a basis for efficient analysis and exploration of these graph structures.

Edgar Codd was a pioneer in introducing the relational model in which he highlighted the concept of levels of abstraction (Codd 1970a) by introducing the idea of separation between the physical and logical levels, taking as an example of that concept, the well-known entity-relationship model of Chen. According to Codd, (P. P. S. Chen 1976) integrity constraints are statements and general rules that define the set of consistent states of the database, or state changes, or both. Integrity constraints are continuously studied in the various data models, in the case of graph database models, integrity constraints include (Codd 1981):

- **Instance and Schema Consistency**. The accuracy of the data in a graph database refers to the assurance that each element has information valid according to its unique nature. In other words, each node or link must meet the specific requirements of its type, ensuring that required attributes are present and contain appropriate values. For example, if a graph models the data

from a social network of people and their friendship connections, the accuracy of the data ensures that each node representing an individual includes plausible information — such as a reasonable age and other essential characteristics — according to the rules defined for nodes of type "Person." Model consistency, on the other hand, ensures that the graph conforms to a predetermined data pattern. This means that the structure of the graph faithfully follows an established schema, including the types of nodes allowed, the possible types of associations between those nodes, and the properties (attributes) that each type of entity must have. This link between instance and structure can be formally validated through homomorphisms between instance and schema, as demonstrated by . In summary, the graph needs to respect this structure provided for in the schema, avoiding, for example, the existence of nodes or connections beyond the defined schema or with attributes incompatible with the model (Bonifati et al. n.d.).

- **Identity and Referential Integrity Constraints**. Identity ensures that each node in a graph database has a unique identifier, capable of unambiguously distinguishing each element from all other nodes. This imperative avoids duplication of entities in the system by ensuring that each actual item in the modeled domain is represented by a single node, even though multiple relationships or attributes may be associated with it. This uniqueness is important to preserve the coherence of the representation and to prevent incongruities resulting from redundancy. Referential integrity constraints, on the other hand, state that every relationship present in the graph must reference valid and existing nodes, thus preserving network consistency. In practical terms, this means that if a node "A" maintains a connection with a node "B", then node "B" and its identity must exist prior to and remain valid throughout all changes to the database. This rule prevents, for example, the creation of "orphan" relationships or references to non-existent entities, ensuring that the connections in the graph maintain a valid semantic meaning within the context of the information model (Angles 2008).
- **Functional and Inclusion Dependencies**. Functional dependencies are rules that specify how values in a set of attributes are related. For example, in a graph that stores information about customers, there may be a functional dependency that states that the seniority of a customer depends solely on the customer's identification number. Include dependencies ensure that certain sets of values are contained within other sets. For example, there might be a constraint that states that all employees in a company must be included in the company's people list in the graph. Such constraints can be formalized in the context of graphs through *Graph Functional*

Dependencies (GFD), as proposed by Fan, Wu, and Xu (2016), which extend traditional functional dependencies to include topological relationships as well.

Behind any software application lies the complexity of a database's operations. The operations mentioned above refer to the basis used to make sure that data is being stored, retrieved, and managed in accordance with reliability. In other words, the reliability of the points previously presented. To this end, two different sets of principles have been developed and implemented, respecting the way in which the DBMS will come to work, referred to as ACID and BASE.

ACID (Atomicity, Consistency, Isolation, and Durability) is a set of properties that ensure that transactions in a database are handled accurately and securely. These principles ensure that, in any situation, the data in the database remains in a reliable state, even in the face of system failures, unforeseen outages, or concurrent situations involving multiple operations (Haerder and Reuter 1983). On the other hand, BASE, an acronym for BASE *(Basic Availability, Soft State, and Eventual Consistency)* represents a set of principles that prioritizes availability and performance in distributed systems. Rather than immediately looking for data consistency, BASE recognizes that in a highly distributed and ever-changing environment, achieving instant consistency can be difficult without compromising availability and performance. As we investigate the increasing complexity of contemporary SBD, it is important to understand these two sets of principles. ACID and BASE are not in opposition, but rather represent complementary approaches aimed at meeting the diverse demands of database applications, ranging from highly critical financial systems to highly scalable Web app (Pritchett 2008).

2.2.1 ACID

At the heart of DBMS, transactions and their ACID properties represent one of the most solid guarantees of reliability and integrity in the execution of complex operations. Proposed by Haerder and Reuter 1983, ACID properties emerged as a response to the need for formal consistency and strict control in multi-user and failure-prone environments. These properties ensure that each transaction is executed in a complete and isolated manner, preserving database consistency regardless of competition, technical failures, or interruptions. Its importance remains unquestionable, especially in mission-critical areas – such as banking, hospital or institutional registration systems – where any deviation in data integrity undermines the trust and operability of the system. In contemporary practice, including graph-driven databases such as Neo4j, ACID properties are maintained as the standard of transactional excellence.

- Atomicity, which ensures that a transaction should be treated as a unit. In other words, all operations in a transaction are executed successfully or if something goes wrong in any part of the transaction, all operations previously done are rolled back, which means that the database always remains in a good state.
- Consistency, which ensures that a transaction takes the database from one consistent state to another. In other words, after the transaction, all database integrity rules must be respected. If a transaction violates any of these rules, that transaction is rolled back, ensuring that the database never loses its consistency.
- Isolation, which regulates competition. This means that a running transaction is isolated from other transactions until it is completed. Therefore, transaction A cannot change if transaction B is working on the same baseline as transaction A.
- Durability, which indicates that after a transaction is finalized the record should be made permanent. Even if system failures or unexpected outages occur, the records remain intact and cannot be unredacted. To ensure this persistence, transactions are stored in logs, so that the system can regain its consistent state after any eventuality.

In this way, Neo4j, as a graph-oriented database management system, stands out for being solidly based on the ACID principles. This transactional basis gives it a high degree of robustness and reliability, making it particularly suitable for scenarios where data accuracy and integrity are critical.

2.2.2 BASE

With the advent of cloud computing, large-scale distributed architectures, and the real-time digital economy, new requirements have emerged that challenge the rigid assumptions of the ACID model. The BASE paradigm represents a change in perspective, favoring scalability, continuous availability and latency tolerance over immediate consistency. This approach is particularly relevant in NoSQL systems and largely geographically replicated data architectures, where seamless synchronization is often impractical. BASE does not deny consistency but makes it more flexible. It admits temporarily inconsistent states if convergence occurs in an eventual and predictable way. In environments such as social networks, streaming platforms or global e-commerce, BASE provides the means to ensure real-time response without sacrificing the stability of the system. It is, therefore, a paradigm of distributed robustness, contrasting but complementing the ACID model. Let's take a closer look at the BASE properties (Birds 2009):

- Basic Availability, which addresses the continuous availability of the system, even in situations of partial failure. This means that even if some parts of the system fail, it should still be able to respond to requests and remain up and running.
- Soft State, which recognizes that the system can change over time due to updates or failures, and that change doesn't have to be instantly reflected everywhere. In other words, the system may have transient states, which may not always be consistent.
- Eventually Consistent, which is based on the idea that in a distributed system data consistency may not be instantaneous but will eventually be achieved as updates propagate throughout the system. This enables higher availability and performance in distributed systems while sacrificing immediate consistency.

ACID and BASE, therefore, constitute two sets of principles that play fundamental roles in database management, intertwining in a significant way CRUD operations. This fundamental set of operations is essential in the management of data in information systems and databases. Each component of CRUD plays a specific role in the effective handling of information (Huseynli and Akcayol 2021).

2.2.3 CRUD Operations

Regardless of the complexity of the data models — whether relational, document-oriented, graph-oriented, or object-oriented — the fundamental operations that underpin interaction with any database are synthesized in the acronym CRUD (Create, Read, Update, and Delete). These four operations represent the functional core of persistent data manipulations, serving as the basis for SQL, RESTful APIs, and entity management mechanisms. Originally derived from software modeling, the CRUD model has gained cross-cutting recognition in transactional and analytical environments, in which clarity in the manipulation of entities and relationships is critical. In graph-driven databases, such as Neo4j, CRUD operations apply not only to entities (nodes), but also to relationships (edges) and their respective properties, requiring mechanisms that ensure the topological and semantic consistency of the network. CRUD is not just an operating convention, but a methodological building block for the design, evolution, and governance of data in any modern digital architecture (Cattell 2010, Stonebraker et al. 2010). Let's detail each of these operations:

- Create, which involves the ability to enter new data into a system or database. This can be anything from creating a new record in a relational database to adding a new object in an object-oriented application;
- Read, which refers to the ability to access and retrieve data already existing in a system or database. This includes querying specific records, searching for information, or reading data for display or analysis;
- Update, which deals with the ability to modify data already existing in a system or database. This can range from updating a specific field in a database record to modifying attributes in a software object;
- Delete, which is about removing data from a system or database. This may involve deleting entire records in a database or removing specific objects in an application.

2.3 Graph-Based Data Models

In the context of graph-oriented databases, the central question of its definition as a data model arises, that is, the identification of the set of characteristics that distinguishes it from traditional database models. Understanding this paradigm requires the analysis of its inherent properties, exploring its structural and

functional particularities that differentiate it from other approaches. This model is defined by attributes that differentiate it from traditional database models, such as specific data structures, specialized query languages, and tailored integrity constraints. When we analyze the most relevant graph database models, we find a variety of approaches, each with its own nuances and specific applications. Thus, it is important to understand the distinction between the schema and the instances of a graph database. While the schema defines the overall structure of the database, including node types, relationships, and properties, the instances represent the actual data that populates that structure, reflecting domain-specific objects, entities, or events.

The term "data model" has been widely studied and used in academia. For Codd, the conceptual tools that make up the database model must contemplate a structuring, description, maintenance of data and a way to retrieve or query the data (Codd 1981). Based on these characteristics, the main aspects found in the modeling are presented, which are: data structures, query languages and integrity constraints. This is followed by a brief description of the most relevant graph database models identified throughout the research. Such data models can be used to specify allowable data types, support a general database design methodology, deal with database evolution, develop high-level language families for querying and manipulating data, and be a vehicle for research on the behavioral properties of alternative data organization structures. According to the same author, there are three essential components to database models, namely: a set of data structure types, a set of operators or inference rules, and a set of integrity rules (Codd 1981).

Entity types and type checking are valuable data modeling tools, as well as integrity checking, as they allow database schemas to be represented and applied. However, some graph database models do not define a schema, but this premise is consistent with most models. To face the issue of the consistency of the instance of the schema in general, it is consensual that two guidelines should be followed. First, in which the instance must contain only concrete entities and relationships of entity types and relationships that were defined in the schema. The second, where an entity in the instance can only have these relationships or properties defined for its entity type. Knowing that, all node and edge labels that occur in the instance must also exist in the schema; But curiously the opposite is not always necessary. In this way, incomplete or non-existent information can be incorporated into the database.

The consistency check is also involved with the restructuring of queries and database updates, namely with the exclusion of entity types or relationships which, in the schema, implies the exclusion of real entities in the instance. The level of separation of the instance and the schema, or rather the degree

to which the schema and the instance are different objects in the database, are sometimes a difficult problem to solve. Representing unstructured entities means that the data does not respect a previously defined data structure. For this purpose, a data model can be classified as structured or unstructured, that is, whether (or not) it allows the definition of a schema to restrict the database instance to data. Data redundancy in an instance graph can contain many entity instances, artificial nodes, set nodes, join nodes, or composite value nodes, which leads to the database having redundancy in formation. One solution is to group entities based on some of their relationships. In this way, this abstraction creates a single entity for each duplicate entity equivalence class. Therefore, it enables the elimination of duplications.

Regarding object identity and referential integrity, set-oriented data models such as the relational model determine tuple identity through the values of key attributes. On the other hand, in object-oriented databases, the identity of the object is independent of the attribute values, being obtained by defining for each object in the database with a unique identifier, for example, a label. In the case of graph database models, it is verified with some redundancy that they implement both types of identification. The notion of integrity constraint in graph databases is implied in the creation of consistent instances and in the correct identification and reference of entities (Angles and Gutierrez 2008).

Graph-driven database models can be defined as those in which the data structures for the schema and instances are modeled by graphs or generalizations of them, and the data manipulation is expressed by graph-oriented operations and their constructors (Angles and Gutierrez 2008). Object-oriented database models, whose purpose is to represent data as a collection of objects, are organized into classes and are assigned to complex values. Graph database models appeared at the same time, along with the first object-oriented database models. The term "graph data model" in the literature is defined by three basic components: the (Kim 1990) data structures, the transformation language and the integrity constraints, and can be characterized by either the data or the schema in graphs, hypergraphs or other functional abstractions of graphs. There are several related aspects that are outside the scope of the work, but after reading they served as guiding complements to this research. Some of these topics are graph visualization, graph methods for databases, and the implementation of the graph SBD.

2.4 Graph-Based Database Models

In graph-based database models applied to real-world contexts, a common difficulty for beginners lies in distinguishing between the schema level and the instance level, since both can be represented using graphs (Robinson, Webber, and Eifrem 2015). During the initial learning phase associated with the development of a BDOG using Neo4j, the author encountered a significant challenge when modeling the support management of a swimming pool construction project (Figure 8 and Figure 9).

This experience proved to be particularly instructive, as it exposed a widespread confusion between the concepts of schema and instance, prompting a deeper conceptual and methodological refinement in both the modeling process and the use of the Cypher language in Neo4j.

Figure 8 presents an example of a graph instance, built on Neo4j, used to model the management support of a swimming pool construction site. The central vertex represents the main entity (the work), from which multiple relationships are established with other vertices that symbolize specific components of the execution, such as *Concrete tanks and beams*, *Support equipment*, *paints*, *Coatings* and *Prefabricated finishes*. The edges identified with the label **"**INTEGRATED**"** describe the integration relationships between the main work and its constituent elements, highlighting how the graph structure allows to represent dependencies and interactions explicitly.



Figure 8 - BDOG conceptual schema for the construction of a swimming pool.

This example illustrates how the graph-oriented approach facilitates the representation of real construction project management scenarios, by allowing both the schema (types of entities and relationships) and the instance (concrete data of a specific project) to be represented under the same conceptual structure.

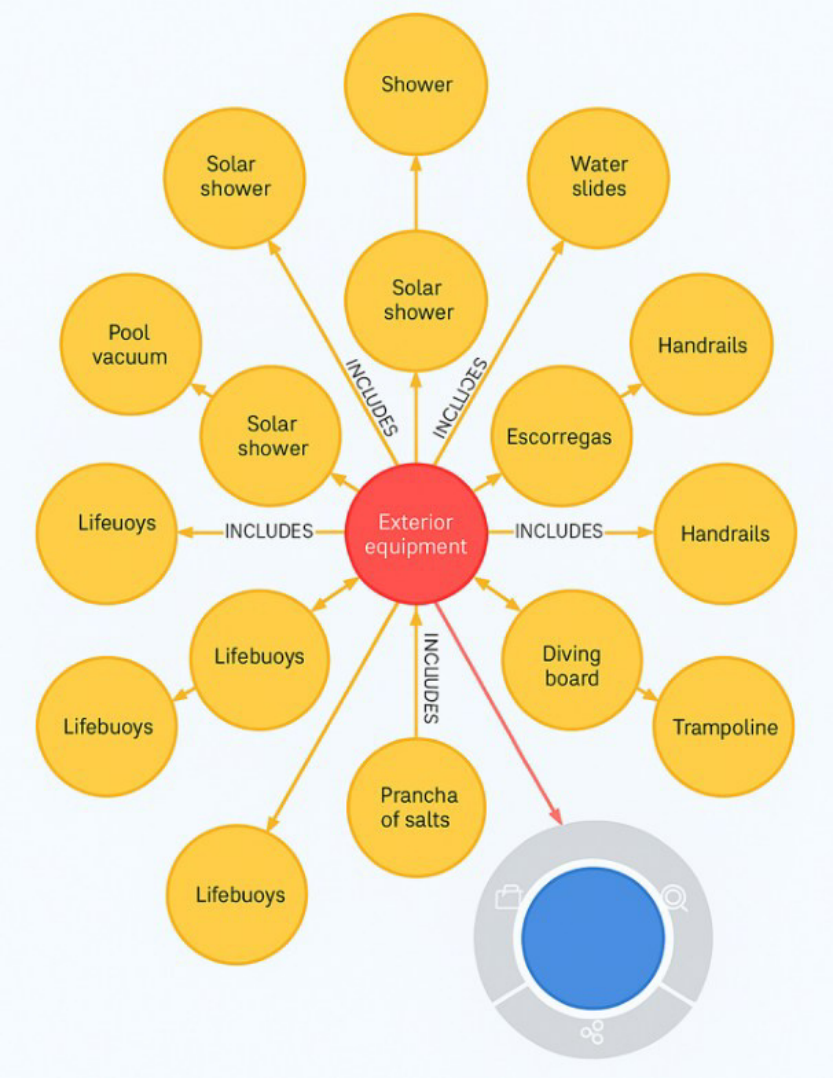


Figure 9 – Partial schema from the construction of a swimming pool model.

Through practical examples, such as modeling the construction of a swimming pool in a graph-oriented database, we can illustrate the challenges and complexities involved in properly defining the schema and creating the corresponding instances. Distinguishing schema and instances is essential for the development of graph-oriented databases, allowing for a more accurate and efficient representation of real-world data. A clear understanding of the distinction between schema and instances is essential for

the development of graph-oriented databases. Once this distinction was realized by the author, it became possible to create a tangible data structure to represent the construction of the pool in Neo4j. This learning process and methodological evolution is an example of how practice and experience are the cornerstones in the field of graph-oriented database modeling. A possible graph database model is therefore a model in which the data structures for the instances and schema are modeled as a directed graph (Figure 9), possibly labeled, or generalizations of the graph data structure, in which data manipulation is expressed in the form of graphs, allowing the definition of oriented operations, constructors, and appropriate integrity constraints on the structure itself (Angles and Gutierrez 2008).

In graph-driven databases, the adoption of BASE principles, in contrast to ACID properties, marks a conceptual transformation that favors more adaptable, scalable and resilient systems. Such guidance not only enables a more agile handling of interconnected data but also responds to the demands of modern applications operating in distributed and highly complex environments, while ensuring high availability and robust performance.

According to Gyssens (1990) data manipulation is expressed by graph transformations, or by operations whose primary primitives are in graph resources, some database models define a flexible collection of type constructors and operations, which are used to create and access graph data structures. In the first steps of modeling using graph-oriented databases, although the author has some mastery of the constituents of swimming pools and their construction process, little was shown to be favorable for the intended modeling (Angles and Gutierrez 2008). Later, although there were "INTEGRATED" relationships associated with the entities instantiated in that database, such a schema was revealed confusing and inappropriate for the required modeling, because its instantiation is easily confused with the other schema, which is normal, given the type of natural language incorporated in these BDOG. In practice, for the pool object, the semantic potential of the instantiated BDOG would not be enriched at all, as illustrated in the Figure 10.

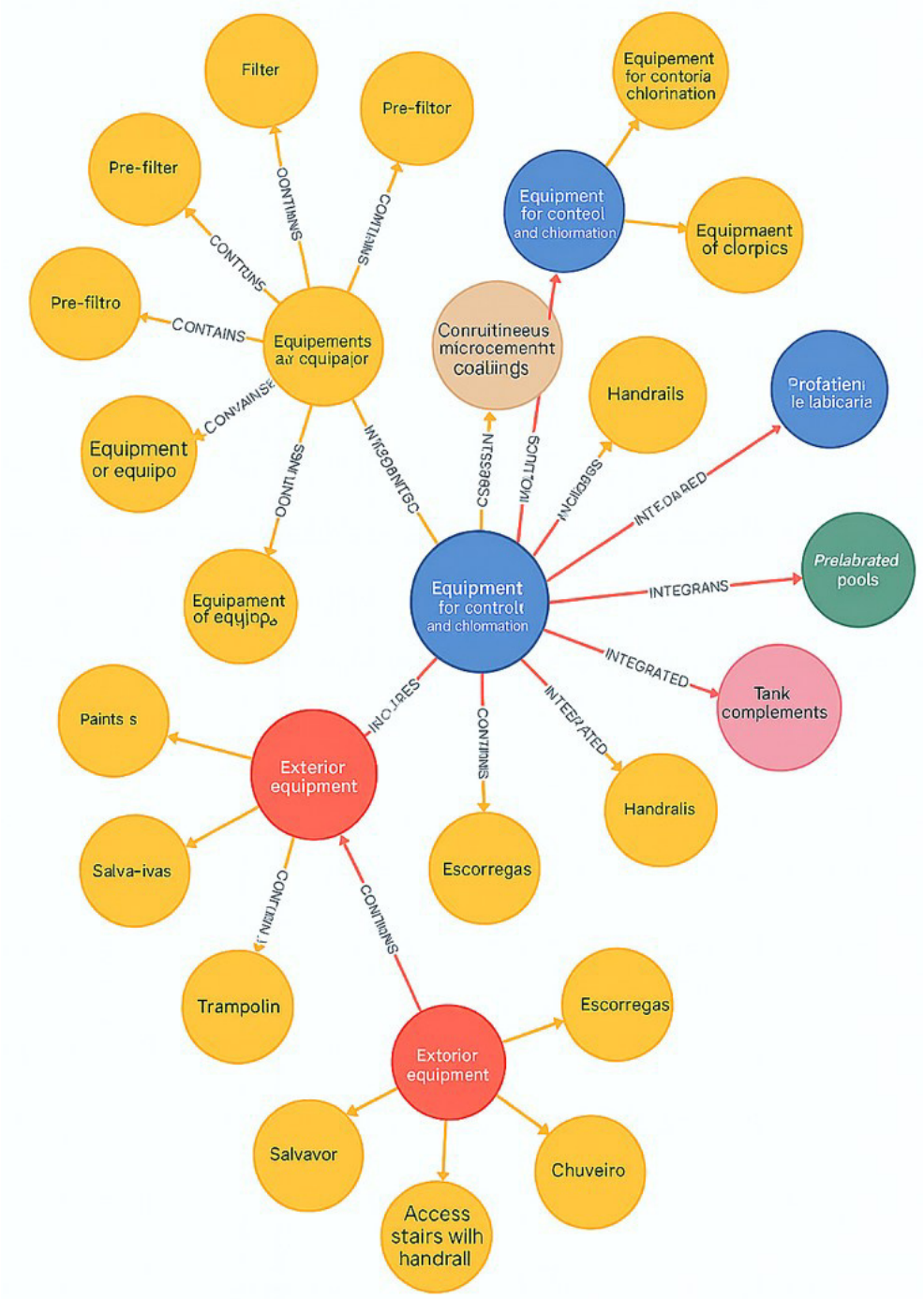


Figure 10 - BDOG graph of the pool.

Finding that, to the detriment of what was developed, it would be simpler to store that information in tables - see Table 2.

Table 2 - Entities considered in the BDOG for the construction of a swimming pool.

Entities
{"cost":,"description":P pre-made iscines}
{"cost":,"description":Pre-made finishes}
{"cost":,"description":Ceramic tiles}
{"cost":,"description":Switchboard}
{"cost":,"description":Debugging equipment}
{"cost":,"description":Pool electric pump}
{"cost":,"description":Filter}
{"cost":,"description":Selector valve}
{"cost":,"description":P re-filter}
{"cost":,"description":P water-to-water heat exchanger}
{"cost":,"description":Access stairs, with handrail}
{"cost":,"description":Handrail set}
{"cost":,"description":Handrail set}
{"cost":,"description":Slides}
{"cost":,"description":Shower}
{"cost":,"description":Solar shower}
{"cost":,"description":Cleans funds}
{"cost":,"description":High chair for rescuer}
{"cost":,"description":Lifeguard}
{"cost":,"description":Lifeguard}
{"cost":,"description":P jumping board}
{"quantity":,"cost":,"description":Push mouth}
{"quantity":,"cost":,"description":Suction mouth}
{"quantity":,"cost":,"description":Union passes walls}
{"brand":,"cost":,"description":P pre-made iscina}
{"quantity":,"cost":,"description":Pool edge}
{"quantity":,"cost":,"description":Pool edge drainage gutter}
{"quantity":,"cost":,"description":Pool tank liner with mosaic}
{"quantity":,"cost":,"description":Stoneware Tile Pool Tank Lining}
{"quantity":,"cost":,"description":P special ceramic tiles for pool finishes}
{"quantity":,"cost":,"description":Decorative continuous pool tank liner with microcement}
{"quantity":,"cost":,"description":Electrical panel for pool lighting}... And so on

As such, the initial attempt to implement, in Neo4j, a representation of the construction process of a swimming pool, whose objective was precisely to highlight the dependencies and stages of the "constructive process", proved fruitless. Such failure resulted, above all, from the difficulties inherent in the abstraction necessary to adequately structure the semantic relationships between entities, in the distinction between schema and instance. However, this experience proved valuable in showing that the resulting BDOG was, in fact, nothing more than an embryonic conceptual schema — a metamodel — of a BDOG yet to be populated. This structure took the form of a digraph with weak connectivity, representing not a functional instance, but rather a support topology that could be realized. This finding allowed us to identify a recurrent confusion between the concept of schema and the instantiated database itself, a distinction that becomes critical in the context of BDOG, in which schemas assume, as a rule, the form of directed and annotated digraphs. Once this initial difficulty was overcome, it was possible to develop a deeper understanding of the role of modeling in graph-oriented databases. It has become evident that the semantic expressiveness of data does not reside exclusively in isolated entities but arises intrinsically from the relationships established between them. This property — emergent relational semantics — is one of the main differentials of BDOG compared to traditional relational models, whose semantic expressiveness depends heavily on external keys, normalization, and explicit joins. It is important to note that this relational semantics may not be readily apprehended by inexperienced modelers. However, once understood, it is essential for the construction of meaningful consultations, especially in contexts in which it is intended to dynamically represent entities over time. Building on these findings and insights, it was possible to conceive, with greater precision and structural coherence, a new BDOG oriented to the modeling of the process of assembling a LEGO set, this time with a clear understanding of the articulation between the levels of abstraction, instantiation and inter-relational semantics. In the following section, we present the study and case modelling of the database to host the construction process of the LEGO set.

2.4.1 BDOG Model for LEGO Assembly

The construction of a graph-oriented database to represent the process of assembling LEGO sets, BDOG_LEGO, was a significant step in the development of semantic modelling with graphs, contributing to consolidate practices applicable to different domains. Starting from an official catalog of LEGO instructions, it was possible to extract an implicit structure in the assembly steps, which allowed the identification of relevant entities, relationships and attributes to be integrated into the structure of the

graph. Graph-oriented modelling proved to be particularly appropriate in this context, given that assembling a LEGO set involves a hierarchical sequence of actions, explicit topological dependencies between bricks and compositional relationships that would hardly be represented with equal expressiveness in a classical relational model. Initially, an elementary conceptual schema was defined, taking as reference the assembly instructions of a specific LEGO figure, from which the main nodes (such as "Doll", "Piece", "Component") and the relationships that connected them ("composed of", "includes", "requires", among others) were identified. This schema (Figure 11) served as a guiding basis for the population of the database, carried out on the Neo4j platform, through the Cypher query language. As an example, it was possible to associate the 6182787 ID piece to the "Citizen" doll, using the following query:

```
MATCH (I:Dummy ) WHERE I.Description ="Citizen" MATCH (p:Pieces) WHERE p.ID="6182787" CREATE (I)-[:Set_Steps_Dummy {NR:'1', Quantity :'1', Time:'15'}]->(p) RETURN I, p.
```

This query allowed us to create a directed edge between the puppet's representative node and the specific part, adding contextual attributes such as the step number, the number of pieces involved and the estimated time for its placement.

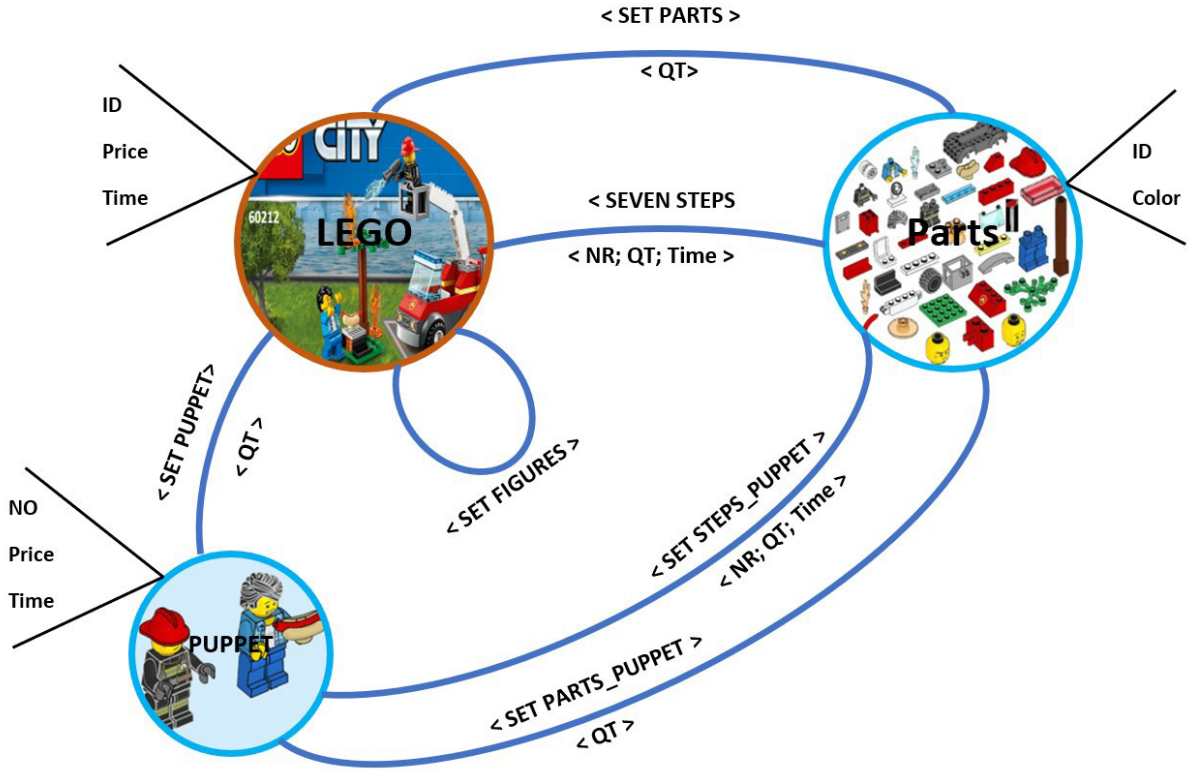


Figure 11 - LEGO BDOG schema.

It is important to note that assembly time, represented by the "Time" property, was an additional semantic dimension introduced during the modelling process, as it is not explicitly present in LEGO catalogues. This variable, estimated empirically, allows the model to be enriched with a useful time component for further analysis, such as assembly simulations or logistical optimizations in educational or industrial contexts. As the database was populated, the instantiated graph began to move structurally away from its original schema, reflecting the complexity and richness of interactions between the modeled entities. Such evolution is evidenced in the Figure 12, in which the complete graph of the "BDOG_LEGO" database is observed, consisting of multiple vertices and interconnected edges, with recurrent patterns and multiple paths between components. This model was particularly important for the PhD process, as it allowed us to clarify the distinction between the conceptual schema — an abstract structure that defines the logical model of the database — and the effectively instantiated graph, which materializes the concrete data, loaded according to the principles defined in the schema.

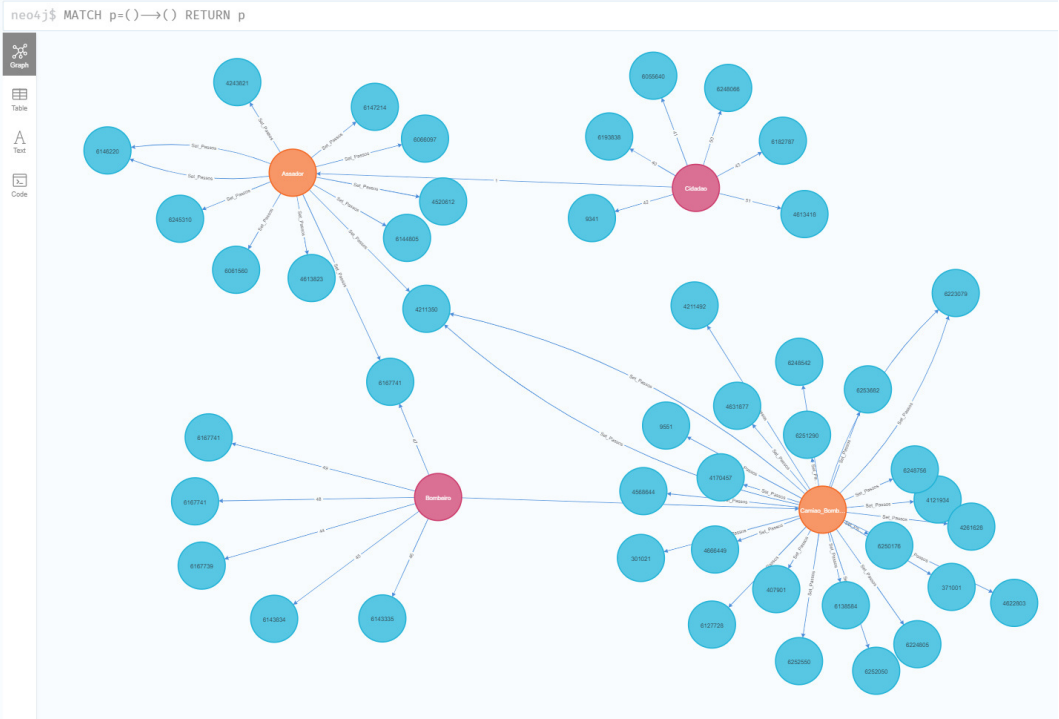


Figure 12 - Global graph of the BDOG_LEGO.

The BDOG model enables the execution of complex queries in an intuitive and efficient manner. Among the most expressive results, the identification of common parts between different assemblies, the detection of reusable subassemblies in the assembly of several objects, and the estimation of the total

time required for the construction of a composite entity stand out. These examples demonstrate the advantages of graph-based databases in modeling systems with high semantic connectivity, where relationships are often as informative as the attributes of the entities themselves.

More than a technical exercise, the modelling of the "BDOG_LEGO" database was a process of conceptual refinement. On the one hand, it reinforced the perception of the semantic value inherent to relationships in graphs. On the other hand, it consolidated the understanding that the semantic richness of a data model is not only in the amount of data stored, but above all in the way this data is interconnected, enabling inferences, efficient navigation and answers to complex questions with reduced computational cost. This experience, markedly different from the limitations faced in the previous attempt to model the construction process of a swimming pool, proved to be instrumental for the advancement of the research and directly contributed to the ability to apply the developed models (such as the MLHG) in real contexts.

2.4.2 A BDOG for a Reinforced Concrete Structure of a Building

Based on concrete practical situations in the domain of Civil Construction[7], a database ("BDOG_EBA") was created with the purpose of modeling the construction of the reinforced concrete structure of a small building. In Figure 13 we can see the structural schematic, in 3D, of the building, which was previously modeled in the software *CYPECAD*[8].



Figure 13 - Structural schema (3D) of the building - preliminary case study.

[7] The choice of Civil Construction as a field of application stems from the professional experience of the doctoral student and the relevance of this area as a critical sector of the economy, characterized by high structural complexity and abundance of relational data susceptible to representation in graphs.
[8] https://www.cype.com

For the realization of the "BDOG_EBA" an extensive amount of data was analyzed, as well as the most appropriate elements (entities) for the characterization of the database model. Next, a digraph was developed for the BDOG schema, correlated to the preliminary modeling of the concrete structure of the building.

In this modeling, the entities were considered as structural elements, covering elements such as footings, foundation beams, columns, beams, slabs and massifs, being related through three types of links: "Supports", "Links" and "Supports". The Figure 14 presents a simplified representation of the schema, without describing the properties of the entities and relationships.

This schema can be identified as a trivial MLHG, with a single level of abstraction, directed to the specific modeling required. However, more comprehensive aspects were not addressed, considering three main dimensions: integration, dimensional characterization, construction process and cost. The schema, although it offers the ability to model dimensional characteristics and some functional dependencies between entities, does not fully cover the construction process and its impact on the progressive cost of construction.

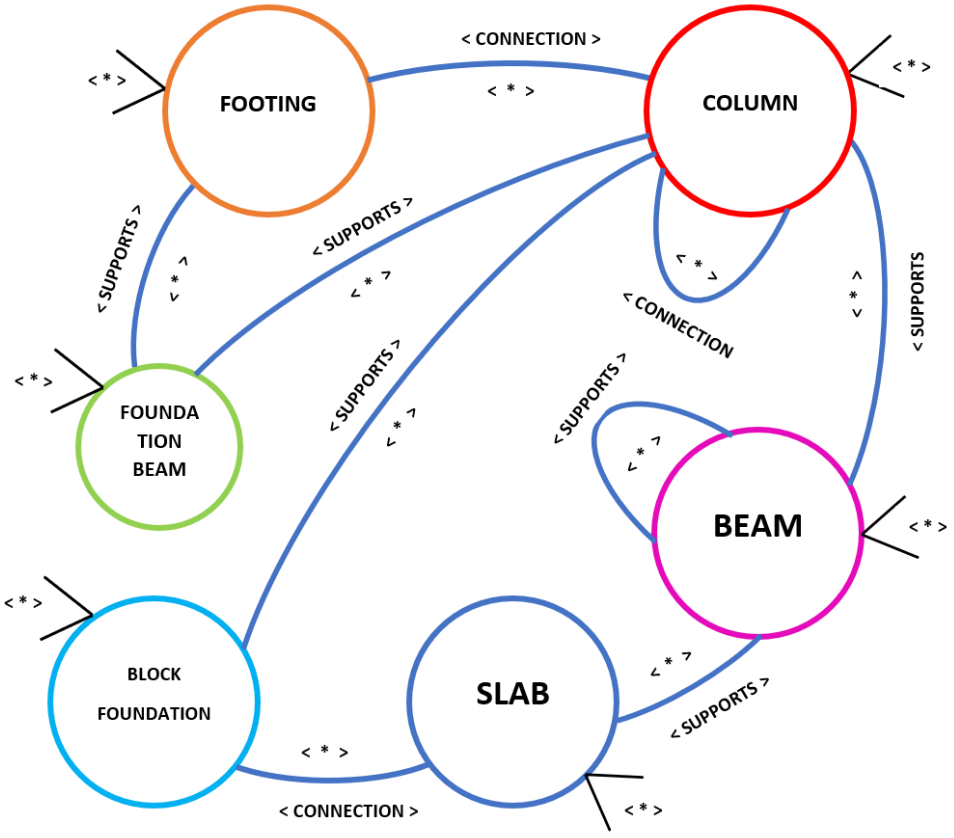


Figure 14 - Schematic of the BDOG_EBA

The inclusion of the construction process in the modeling can represent a significant addition, since this aspect strongly depends on available resources, planning of the work, execution and technical capacity of the builder, among other factors. Additionally, the time required for the execution of the construction process, combined with the dimensional characterization, will influence the progressive cost of construction, aspects that are not fully modellable with the schema presented in the Figure 14. However, in the field of the problem of structural calculation, there are several parametric possibilities, which can include, for example, the types of forces, the types of loads, the characteristics of materials, regulatory technical considerations, etc.

Following the population of the respective database, the data and results provided by CYPECAD's structural modelling were considered, and, after processing and cleaning the data in Excel, all the entities in the example were loaded into the case study. See the following example:

```
MATCH (p:Pilar)
where p.NRP="1"
match (s:Footing)
where s.NRS ="1"
create (p)-[r:Link{NRV:'1', Tipo:'Exterior', Mx:'215',My:'205', N:'1220'}]->(s)
return p, s
```

As an example, for the entity "Footing", some of its properties uploaded to BDOG are also listed, which as illustrated in the Figure 15 Characterized by:

```
“MATCH (n:Footing) RETURN n “
{
  "Height_cm": 70,
  "D16_kg": 11.11,
  "Concrete_Type": "C16/20",
  "WidthX_cm": 80,
  "WidthY_cm": 150,
  "Column_Ref": "P1",
  "Steel_Grade": "A400",
```

```
  "NRS": 1,
  "D6_kg": 0.84,
  "Foundation_Type": "Isolated",
  "Reinforcement_X": "X: 5Ø12 @18",
  "D20_kg": 11.95,
  "Concrete_m3": 0.84,
  "Reinforcement_Y": "Y: 5Ø12 @18"
}
```

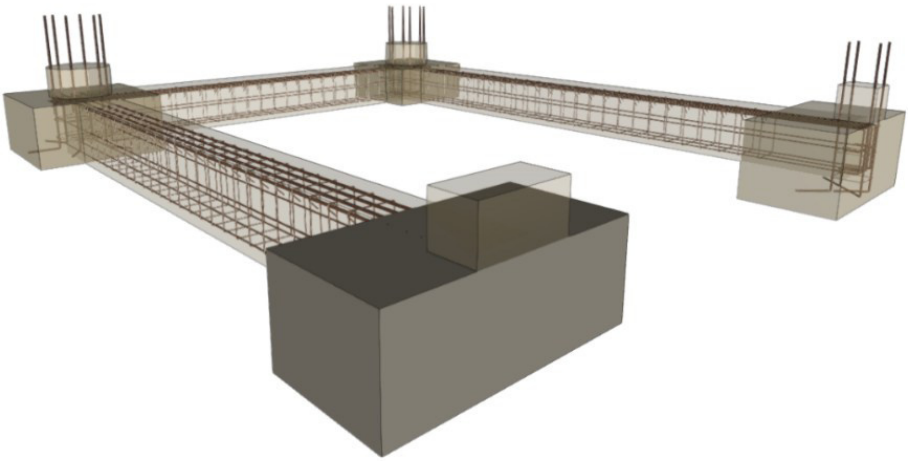



Figure 15 - Case study representation of the “Footing” entity.

Thus, it is possible to verify the wide variety of data that can be represented in a database oriented to property graphs, whose structure allows the trivial registration and efficient manipulation of this information, including in the context of the structural calculation associated with the entities (nodes). For the "Beam" entity, you can apply identical relationships to other types of nodes. As an example of this type of relationship, we have the "Support" relationship, which serves to relate four types of nodes, two by two, namely, [Foundation Beam – Footing], [Beam – Column] and [Solid – Pillar], whose application is exemplified below:


```
"MATCH p=()-[r:Supports]->() RETURN p"
[
  {
```

```
  "Stirrups_Reinforcement": "1xØ8 @20",
  "Height_cm": 50,
  "D12_kg": 30.98,
  "Concrete_Type": "C16/20",
  "Web_Reinforcement": "1x2Ø12",
  "Bottom_Reinforcement": "3Ø12",
  "Column_Ref": ["P1", "P2"],
  "Steel_Grade": "A400",
  "Concrete_m3": 0.86,
  "NRVF": 3,
  "Width_cm": 40,
  "Top_Reinforcement": "4Ø16",
  "Element_Type": "Beam VC.T-1.1",
  "D20_kg": 46.59,
  "D8_kg": 15.88
},
{
  "NRA": 6,
  "Element_Type": "Balance"
},
{
  "Height_cm": 70,
  "D16_kg": 11.11,
  "Concrete_Type": "C16/20",
  "WidthX_cm": 80,
  "WidthY_cm": 150,
  "Column_Ref": "P1",
  "Steel_Grade": "A400",
  "NRS": 1,
  "D6_kg": 0.84,
  "Foundation_Type": "Isolated Footing",
```

```
  "Reinforcement_X": "X: 5Ø12 @18",
  "D20_kg": 11.95,
  "Concrete_m3": 0.84,
  "Reinforcement_Y": "Y: 5Ø12 @18"
 }
]
```


In Figure 16 one can see the BDOG graph created for the case study. Although it is the structure of a modest building, its strong nodal connectivity to the data model referring to the database support graph is notorious.

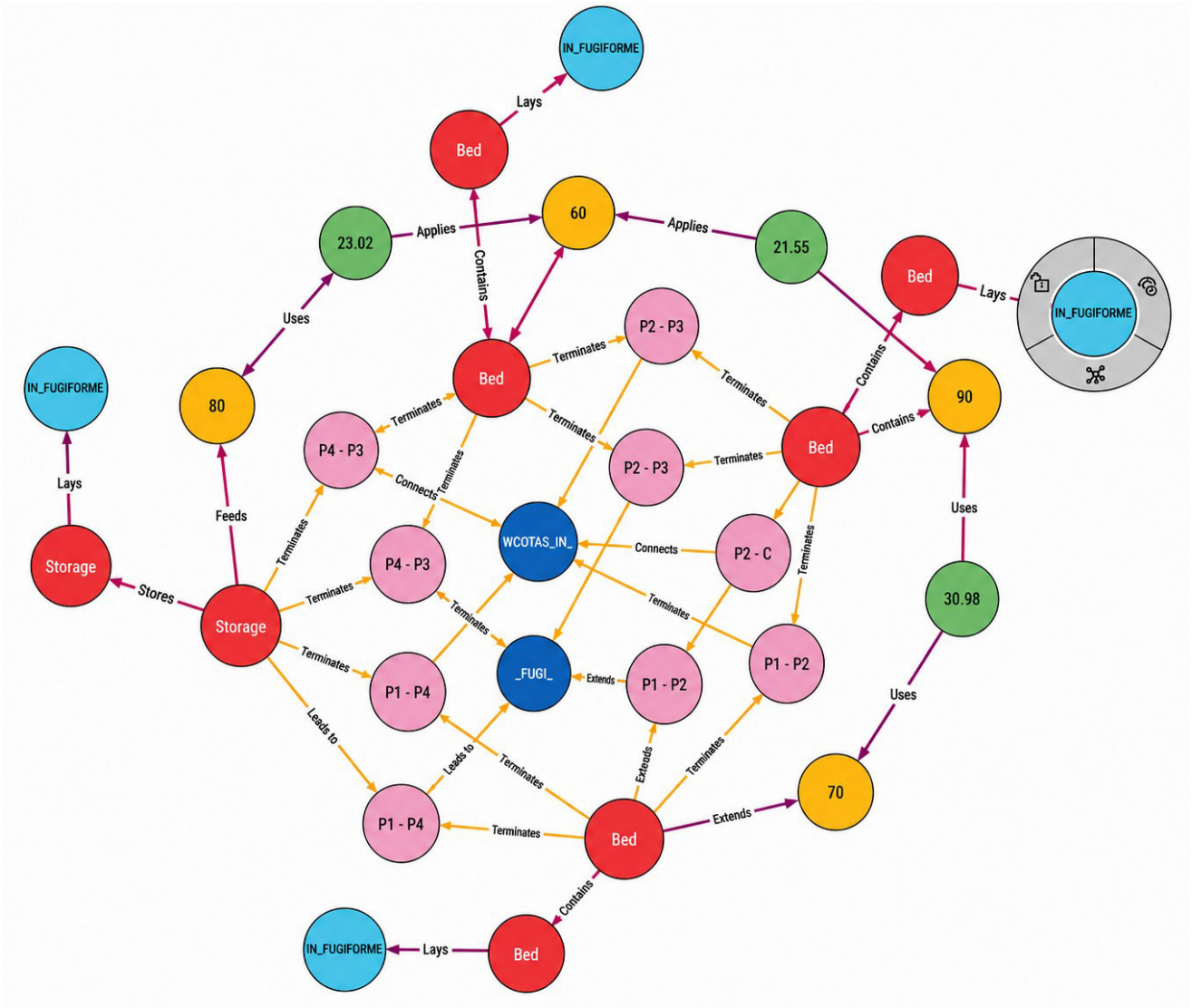


Figure 16 – The complete graph of BDOG_EBA.

Certainly, graph-oriented database models promise to play a key role in civil engineering soon, considering the still untapped potential of this technology in this area. This opportunity for change stems from the intrinsic ability of BDOG to model the complex relationships between elements in an intuitive and natural way, more faithfully replicating the reality of the various processes involved. The Figure 17 illustrates some queries carried out in the implemented base. In this figure, it is noted that both the vertices and their connections contain dynamic attributes, which eliminates the need for predefined rigid structures. Instead, connections between entities are established through external indexes, providing greater flexibility for different data analysis and request scenarios. Thus, it is concluded that BDOG offers a representation model that is more adaptable to Civil Engineering processes, bringing the digital world closer to the complex reality of the elements and relationships inherent to this area of knowledge.

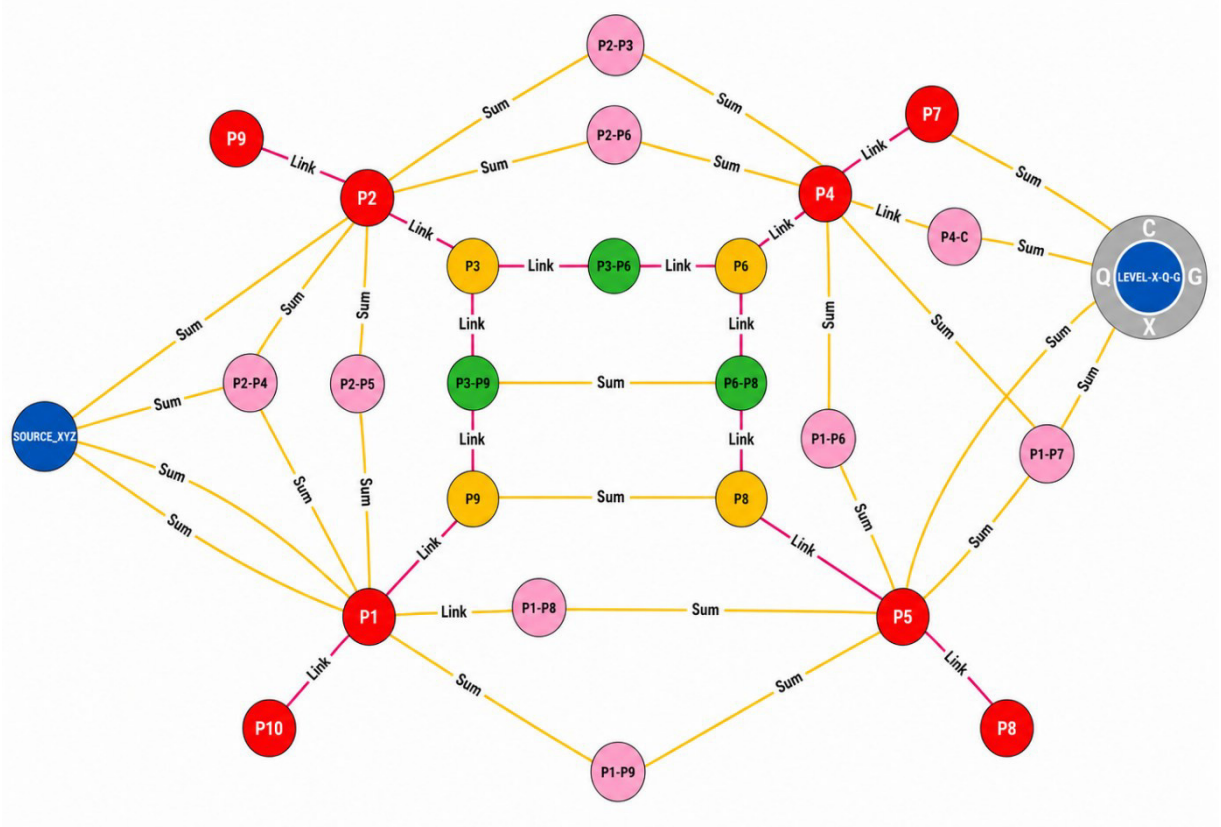


Figure 17 - Consultations at the BDOG_EBA of the previous case study.

Having identified the unequivocal potential of BDOG for the modelling required in the scope of Civil Engineering, in general, and Civil Construction, in particular, and bearing in mind a new context for the development of the case study, the work proceeded with a broader purpose, with the modelling of the

same building, now considering three aspects for modelling through a hypergraph: the dimensional characterization, the construction process and the cost.

Dimensional characterization is one of the axes of analysis already contemplated in BDOG, presenting a strong correspondence with the work previously developed. In contrast, the construction process introduces an additional dimension to modeling, whose central attribute is "time", strongly conditioned by the available resources, the planning of the work, the mode of execution and the technical capacity of the builder. In a dynamic way, this dimension, when articulated with the dimensional characterization, determines the progressive cost of construction, revealing the interdependence between both axes of analysis.

For the study in question, it was necessary to assign to the entities an additional property capable of providing, to each instance, a global identifier that would establish a hierarchy. This hierarchy would allow the graph nodes to be correlated with the construction process and, consequently, with the associated cost. The identifier designed for this function was the "Group" property — already aligned with the concept of hypergraph — as illustrated in Figure 18 and in the Figure 19. This property not only establishes the level of each constituent relative to the corresponding floor but also references the structurally connected elements on the lower levels. Thus, in Group 1 and 2 slabs from floor is included, as well as the respective massifs and four pillars, while in the Group 0 foundations are represented by the entities "Footing" and "Foundation Beams". For example, for Group 2, the ownership of "group" was considered in the entities instantiated. This can be seen in the instructions that were implemented to create it:

```
CREATE (:Column {
  NRP: '5',
  Group: '2',
  Column_Ref: 'P1',
  Material: 'Concrete',
  Section: 'Rectangular',
  Width_cm: 30,
  Height_cm: 30,
  Steel_Grade: 'A400',
  Longitudinal_Reinforcement: '4Ø16',
```

```
  Stirrups_Reinforcement: 'Ø6 @20',
  Concrete_Type: 'C16/20',
  Concrete_m3: 0.27
})
CREATE (:Beam {
  NRV: '7',
  Group: '2',
  Gantry: '2',
  Column_Ref: 'P4–P3',
  Length_cm: 630,
  Width_cm: 100,
  Height_cm: 34,
  Steel_Grade: 'A400',
  Top_Reinforcement: '5Ø10',
  Bottom_Reinforcement: '4Ø20',
  Web_Reinforcement: '0',
  Stirrups_Reinforcement: '1xØ8 @20',
  Concrete_Type: 'C16/20'
})
CREATE (:Slab {
  NRL: '2',
  Slab_Type: 'SOLID_FUGIFORM',
  Slab_Ref: 'LF_01',
  Group: '2',
  Thickness_cm: 34,
  Concrete_Type: 'C25/30'
})
CREATE (:SolidBlock {
  NRM: '2',
  Block_Type: 'INFILL_FUGIFORM',
  Group: '2',
```

```
  Thickness_cm: 34,
  Width_X_cm: 2.62,
  Width_Y_cm: 1.6,
  Top_Reinforcement: '6Ø12',
  Bottom_Reinforcement: '6Ø12',
  Concrete_Type: 'C25/30'
})
```

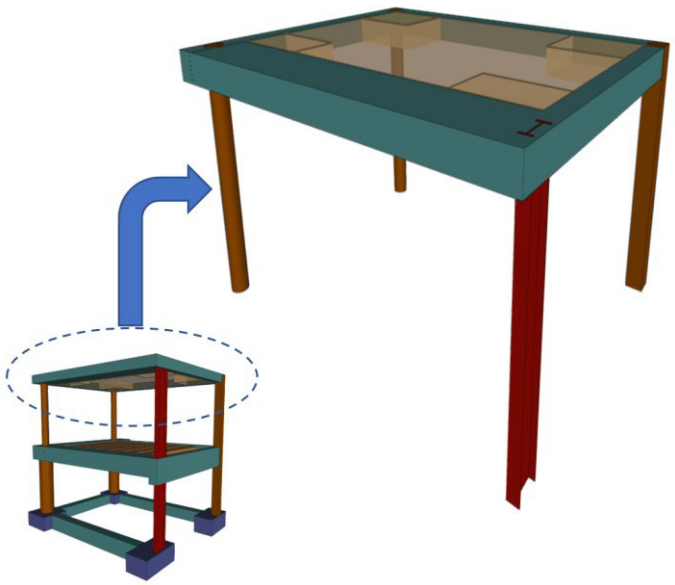


Figure 18 - Group 2 (previous case study).

For the Group 1 property, example:

```
CREATE (:Column {
  NRP: '4',
  Group: '1',
  Column_Ref: 'P4',
  Material: 'Steel',
  Section: 'IPE400',
  Width_cm: 0,
  Height_cm: 0,
  Steel_Grade: 'Fe360',
  Reinforcement: '0',
```

```
  Stirrups_Reinforcement: '0',
  Concrete_Type: '0',
  Concrete_m3: 0
})
CREATE (:Beam {
  NRV: '1',
  Group: '1',
  Portal_Frame: '1',
  Column_Ref: 'P1–P2',
  Length_cm: 655,
  Width_cm: 30,
  Height_cm: 80,
  Steel_Grade: 'A400',
  Top_Reinforcement: '2Ø10',
  Bottom_Reinforcement: '2Ø16',
  Web_Reinforcement: '2x2Ø10',
  Stirrups_Reinforcement: '1xØ8 @20',
  Concrete_Type: 'C16/20'
})
CREATE (:Slab {
  NRL: '1',
  Slab_Type: 'VIGOTAS_IN_SITU',
  Slab_Ref: 'LA_01',
  Group: '1',
  Thickness_cm: 35,
  Concrete_Type: 'C25/30'
})
```

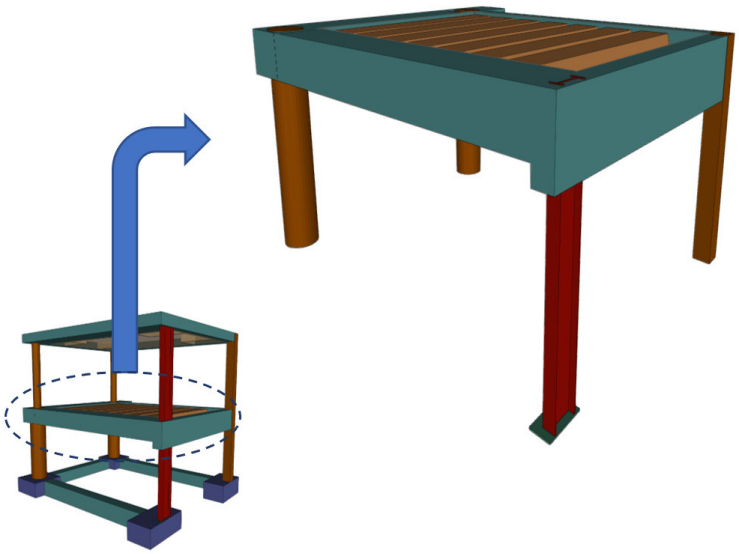


Figure 19 - Group 1 (previous case study).

The "Group" property takes on the role of allowing a selection combined with other properties, serving not only for the geometric positioning of the construction elements, but also for the definition of a global hierarchy of the construction process, organized according to the temporal order of execution and directly associated with the determination of costs. In associated with Figure 20 and Figure 21 Examples of queries performed on nodes with the property "Group" = "1" and "Group" = "2" are presented, where other complementary properties are evidenced for the connected nodes, the concept of which will be detailed later.

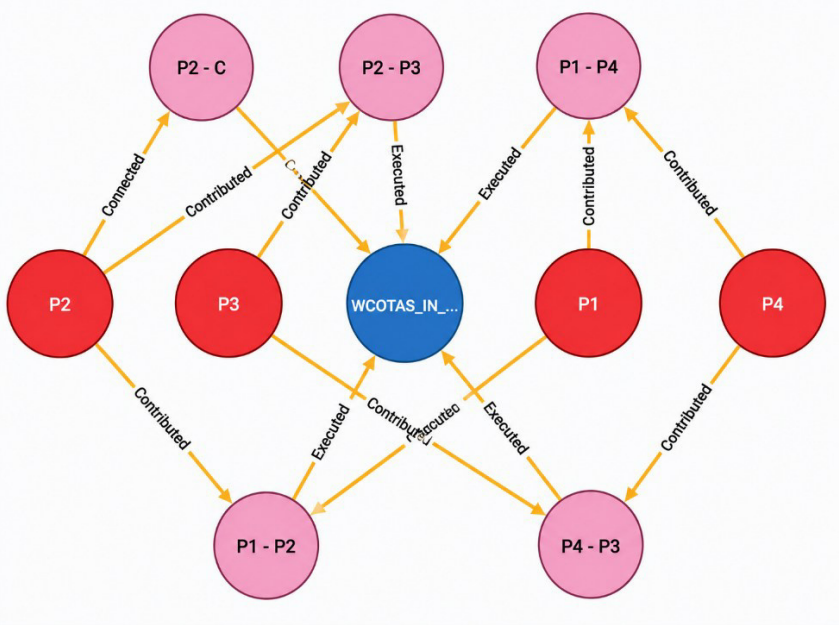


Figure 20 - Queries of nodes with the property of Group = "1".

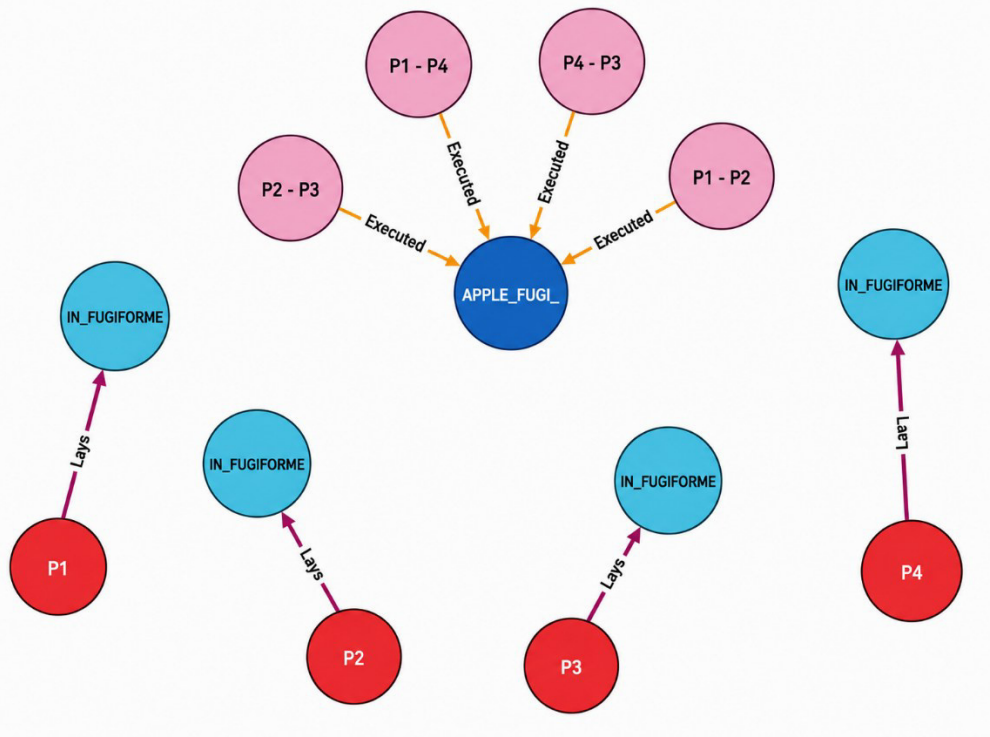


Figure 21 - Queries of nodes with the property of Group = "2".

To ensure control of the entities during the construction process, it was necessary to equip the nodes with cross-properties, to be possible to reference, for each constituent, its morphological connection with other adjacent elements. For example, for the pillar 'P1' (Figure 22) it is possible to intrinsically reference all the elements that interfere with it, thus promoting an execution map for the elements compromised by this pillar. For the VIGA_DE_FUNDACAO node, there is a reference property [P1-P2], in which the label translates the existence of a foundation beam that supports column 'P1' to column 'P2'. For the 'SAPATA' node, it was considered a property, reference to the pillar it supports — in this case, the "P1" pillar. For the support beams of the slabs, a property was also defined that indicates the column "P1" as the support element. Finally, the "P1" pillar itself received a reference to itself, ensuring its identification in different groups and enabling its reuse in the construction hierarchy.

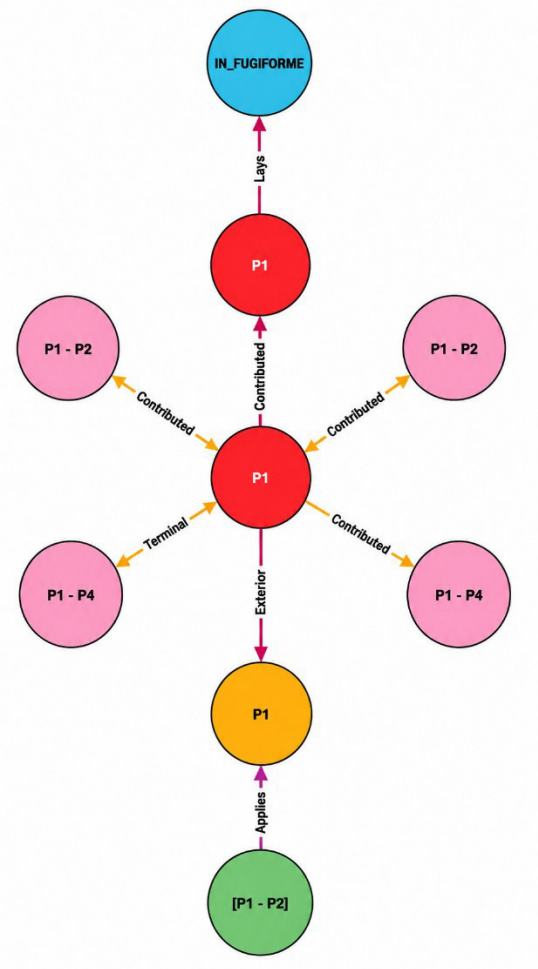


Figure 22 - Reference intrinsic to all elements that interfere with the P1 pillar

2.5 The Adoption of a Graph-Based Data Model

In real-world data manipulation scenarios, data tend to be located at the same level of abstraction, where the topology of the information plays a central role. This topological dimension is particularly relevant for the success of modelling in contexts that use graph-oriented database models. Graphs are appreciable to the user and allow a natural way of dealing with application data, having the advantage of being able to keep all the information about an entity in a single node and show the related information by edges connected to it (Paredaens, Peelman, and Tanca 1995). Graph databases provide specialized storage structures and algorithms optimized for the execution of specific operations (Gueting 1994) . Associated with graphs are graph-specific operations in query language algebra. It is not important to require full knowledge of the structure to express meaningful queries. Queries can refer directly to the structure of a graph, which makes the procedure easier (Abiteboul et al. 1997) operations on graphs allow users to express a query at a high level of abstraction. To some extent, this is the opposite of manipulating graphs

in deductive databases, where often, quite complex rules need to be written, which may be (Gueting 1994) convenient to forget about the schema for navigation purposes (Buneman, Davidson, and Watters 1991). To highlight the advantages of graph-oriented database models, a comparison with other widely recognized database models is carried out. This analysis, presented in the Table 3, focuses on three fundamental dimensions: (i) the level of abstraction achieved, (ii) the underlying data structure, and (iii) the types of information objects privileged by each model.

Table 3 - Comparison between other most popular database models (Angles and Gutierrez 2008)

Database model	**Abstraction Level**	**Database structure**	**Information focus**
Graph	Logical or User	graph	Data + Relationships
Network	physics	Pointers + Registers	Records
Relational	logical	Relations	Data + Attributes
Semi-structured	logical	tree	Data + Components
Semantics	User	graph	Schema + Relationships
Object-Oriented	Physical or logical	Objects	Object + Methods

The differences between graph-oriented database models and the relational model are significant. While the relational model focuses on simple record-type data, whose structure is previously defined, its rigid schema makes it difficult to structurally expand and integrate different schema.

Semi-structured database models (Buneman, Davidson, and Watters 1991) emerged motivated by the exponential growth of unstructured data, resulting from navigation processes and the intense exchange of information in social networks and large-scale applications. This type of data adopts, as a rule, a tree-shaped organization. However, the possibility of cycles between nodes often leads to graph-like structures, bringing them closer to graph-oriented database models.

Semantic database models[9] make it possible to incorporate a set of semantics into the database, from the user's point of view, using high-level abstraction concepts, such as aggregation, classification and instantiation, sub- and super-classification, attribute inheritance, and hierarchies (Navathe 1992). The object-oriented database model appeared when the database community realized that the relational model was unsuitable for large-scale data domains (Kim 1990). These models focus on the dynamics of

[9] Semantic data supports information that has additional meaning, allowing it to be understood in context by humans and machines, facilitating advanced interpretations.

the object, its values and methods. Graph database models focus instead on interconnection, while maintaining the structural and semantic complexity of the data.

There are several models and formulations for the grouping of vertices in graphs, covering a wide range of partitioning algorithms. Among these, the formulations of cutting and partitioning of graphs stand out, as well as the mathematical formulation for maximizing the sum of intra-cluster edges (Selvakkumaran and Karypis 2005) . Graph partitioning problems in k clusters can have different nomenclatures when it is desired to partition the graph into two (bi-partitioning) or more groups (k-partitioning). Formally, the k-partitioning problem based on graph cut formulations (e.g., Minimum Cut Problem; Cutting Radius Minimization Problem; Normalized Cut Minimization Problem; Min-max Cutting Minimization Problem) aims to eliminate branches from a given graph to produce k connected components (Moretti et al. n.d.) . Generally, the goal of formulations for this problem is to provide well-separated clusters with high intra-cluster connectivity. To this end, most of these criteria determine different alternatives for choosing the set of branches to be eliminated. The model, whose formulation is Mathematics for the Maximization of the Sum of the Intra-Cluster branches, seeks to find a possible partition in a graph, to maximize the number of shared branches within the clusters.

One of the (fashionable) algorithms that have been proposed to solve this problem is multilevel algorithms. As an example of these graph algorithms. The authors of this work argue that multilevel algorithms are fast, when compared, for example, to spectral algorithms (Karypis and Kumar 2005) [10], as they can partition graphs with thousands of nodes and branches in a few seconds and are parallelizable algorithms. In exploring graph partitioning algorithms, high-profile considerations have emerged for the following works. First, as noted, when a partitioning algorithm can't find two subsets that minimize the edges crossing between them, it terminates its execution. This is especially true when the initial partition is already of high quality, not allowing for significant gains in terms of optimization. The application of graph partitioning techniques to data classification naturally raises uncertainties, since several possibilities of viable outcomes may arise within the same overall graph structure. This uncertainty highlights the lack of consensus on the use of these techniques for data qualification. A pertinent suggestion is the need to adopt a less flexible approach to graph construction. Instead of restructuring the graph to fit the desired classification of data, the inherent proposal in this thesis work is to create the initial graph following a model that can be manipulated in terms of structure, but that maintains clear

[10] The use of spectral theory in clustering problems in graphs is usually found in the form of spectral relaxation of graph partitioning problems.

semantics. This would allow the expansion of logical structures in a systematic way and reduce uncertainty regarding classification results.

2.6 Multilevel digraphs

Previously, in section 2.1.4, holocode were addressed. These are a structured textual alternative representation of a digraphs encoding detailed information about their Connections and Hierarchies. From this structured representation, and in the general purposes, leading to the creation of data models for DBOD, a variation of the structure for the DBOG was first developed. Acyclic digraphs, according to the author's sketch, the free hand, presented in the Figure 23, called MLD. This functional abstraction, in addition to being able to encode detailed information about their connections and hierarchies, allows us to represent the nesting of digraphs acyclic at various levels of abstraction, starting with a digraph initial (parent) and, from any vertex of this digraph, you can create other digraphs parent(s). This process can be repeated successively at different levels of abstraction. The semantics inherent to the "holocode" presented above, can represent all entities after the nesting of nodes at any level of abstraction. This means that, regardless of the level of nesting, the "holocode" has the ability to leverage and represent all relevant topological information about the digraphs and their connections, being a notation that allows the representation of digraphs (directed graphs) nested at various levels of abstraction. (J. R. G. Ribas 2015).



Figure 23 - Representative Freehand Sketch of Concept of Digraphs Multilevel, MLD.

In this first draft of a MLD, the idea of recreating a structure of acyclic digraphs within acyclic digraphs in multilevel of abstraction is visible, enhancing in each vertex a higher-order encapsulation sequence, which translates, for each vertex, into the need to have exactly one degree of input (source) and one degree of output (sink) for each capsulable vertex. That is, for each vertex, at a given level of abstraction n, the possibility, in itself, of being a shrunken digraph represented at a lower level is considered. The proposed MLD aims to play a role in defining and achieving the objectives of this research work, providing a structured approach to address complex challenges in project management.

We will delve into the main objectives of MLD and its specific motivations in the context of project planning and management, focusing on deadline control, resource allocation, and cost management, and considering methods such as Critical Path. Let's look at each of them in particular:

- Deadline Management. MLD seeks to optimize the management of deadlines in projects, allowing a detailed analysis of decisions that affect the schedule, which is motivated by the need to meet deadlines and avoid delays that could jeopardize the success of the project;
- Resource Allocation. MLD seeks to optimize resource allocation, considering capacity constraints, costs, and timelines. This is motivated by the need to use resources effectively and avoid waste;
- Cost Control. The MLD aims to provide control of project costs, allowing a detailed analysis of decisions that affect expenses. This is necessary to keep the project within budget;
- Critical Path Identification. MLD integrates the CPM, allowing the identification of the most critical tasks for the project schedule. This is essential to prioritize activities and ensure compliance with deadlines.

All these MLD objectives, in project planning and management, aim to address the growing challenges in this field by ensuring that projects are delivered on time, within budget and with the efficient use of resources. With the support of MLD, construction companies can make more informed and strategic decisions, increasing their chances of success in a highly competitive business environment.

In this way, the MLD can be understood as a hierarchical extension of the traditional digraph model, enabling the representation and manipulation of information across multiple levels of abstraction, a capability that is fundamental to project management. Both the (Ribas 2015) MLD framework and the B-Hypergraph approach (Chapter 3) share fundamental similarities in the way they deal with the

representation of complex systems. First, both emphasize hierarchy as a central element in their approaches. MLD achieves this goal through the nesting of digraphs at different levels of abstraction, while B-Hypergraphs establish hierarchies using hyperedges that connect groups of vertices.

The holocode and the MLD were the first key components of this work, which aimed to meet the initial objectives proposed for the representation and manipulation of digraphs as the purpose of creating a digraph-oriented database model. The critical attitude of human thought is fundamental to achieving success in problem solving, challenging the pursuit of comprehensive control, which cannot be fully achieved through the fragmented use of the computational tools available today. These applications, whether individually or complementarily, constantly seek a "success" that often discourages the necessary synchronized aggregation of all the governmental parameters involved in the real world.

As noted by Simon, human information processing is limited, often leading to the adoption of satisfactory rather than optimal solutions (March, J.G. and Simon 1958). One may therefore consider a data structure, namely graphs, capable of scalable expansion, independent of rigid predefined schemas, and naturally suited for incremental creation and manipulation. Rooted in mathematics, Graph Theory provides a powerful framework for modeling problems that often resist trivial solutions or cannot be effectively addressed using classical analytical approaches. The ability to assimilate a simple idea or productive process to a digraph stands out as a distinctive feature of this theory. By assigning weights to the constituent elements of the digraph, it becomes a weighted graph, often referred to as a lattice. This graph-based modeling proves to be effective in addressing complex problems, providing an intuitive and structured visual representation of the relationships between different entities or elements. In a relational database, the administrator can adjust the structure while maintaining logical data independence. This process includes manipulating entities (tables), modifying attributes, cardinalities, and reconfiguring primary, foreign keys, and indexes. However, in a BDOG the flexibility for changing the structure is limited, since the need to manipulate part of the data structure or make changes to the relationships between entities compromises the logical independence of data. This impairment is evidenced in a resulting graph (Figure 24), in which the elimination of entities and relationships makes it impossible to guarantee relationships between other entities that originally existed. This limitation stems from the hierarchical and tightly connected nature of graphs, where interrelationships play a key role in data representation and integrity. This limitation is further illustrated in Figure 24, which highlights how structural changes in a BDOG may compromise logical data independence.

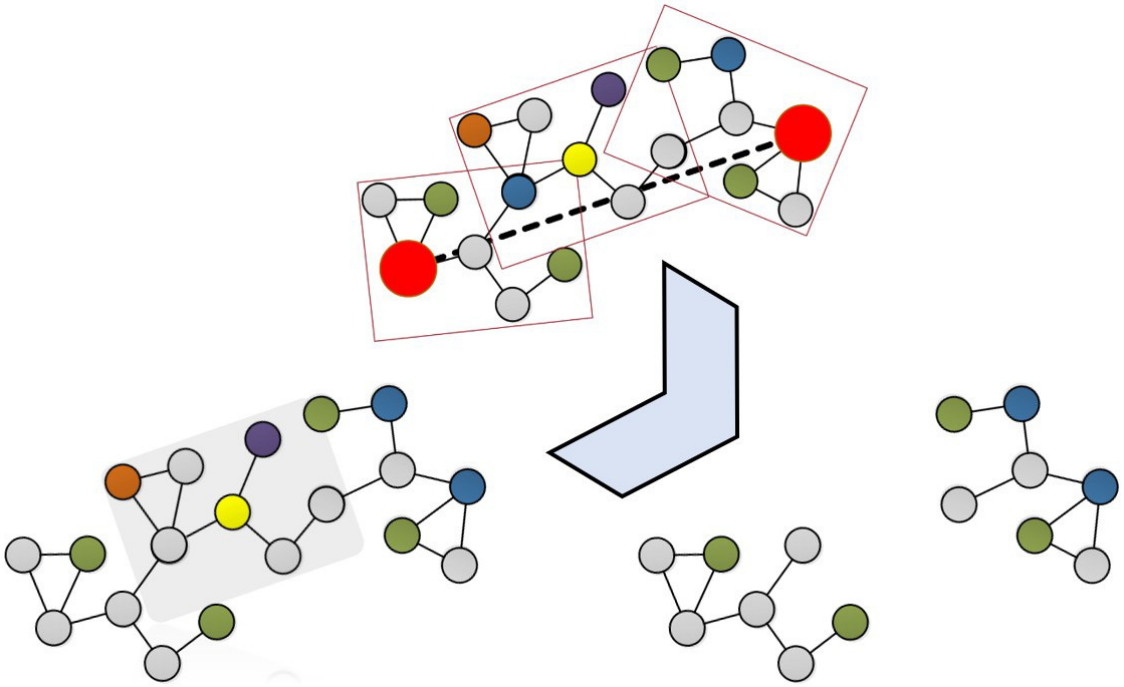


Figure 24 - Compromise of logical data independence in a BDOG.

Although it is advantageous to have an initial schema, whether conceptual or logical, graph data modeling most of the time proves to be poorly structured. With the rapid growth of the database, any well-designed schema can become inefficient when confronting challenges such as aggregating or incorporating data from other databases. In BDOG there is an inherent possibility of "confusion" between the conceptual and logical schema and the physical schema. In fact, when we consider a graph as a data model, we observe that its structure has a propensity to expand easily with the addition of nodes and edges, and thus, the graph's own graphical representation can quickly become unreadable. The complexity increases, and it may even become impossible to perform interventions without compromising the integrity of the BDOGs own support graph. This is evident, for example, when there is a need to modify relationships between nodes or eliminate some component, be it a node or an edge. This complexity can even result in the total dysfunction of the structure of the support graph, in such a way that control of the structure of the database itself is lost. This phenomenon is attributed to the dynamic and fluid nature of graphs, in which relationships arise during the data loading process, challenging the rigidity of predefined schemas. Requiring adaptive and flexible approaches in data management and integration. In this context, we propose the use of MLD as a data model for the schema of a multilevel digraph-oriented database, BD_MLD.

From a practical implementation perspective, it is critical to ensure the ability to create BDOG, in this case, BD_MLDs, in a systemic way. This not only allows adjustments to the data values at nodes or edges but also changes to the data structure of the resulting graph. This flexibility must be executed in order to avoid internal disturbances in the nodes or surrounding edges, even in the face of structural modifications. By addressing the systematic creation of BD_MLDs, we seek to gain full control over the data and the resulting structure after the aggregation of the digraphs. Not only does this make it possible to unambiguously manipulate the structure as a whole or specific parts, but it also greatly simplifies queries to BD_MLD by eliminating the unique need for complex traversals in the graph during query operations and information retrieval.

The systemic creation of BD_MLD is detached from traditional approaches based on rigid schemas or static graphical representations of graphs, as a direct consequence and aligned with the specific purpose of establishing representation in MLD. This representation in digraphs is accomplished through the application of the TDA as established in (Ribas, 2015) to acyclic digraphs, and its realization through the holocode, as discussed earlier in this chapter. Among the various characteristics present in the holocode, the memory savings consumed in the storage and processing of information stand out for this work. This is because digraphs, given their nature as connected graphs with generally dispersed structure, often result in large adjacency or incidence matrices loaded with zeros. This savings provided by the holocode is extremely relevant, contributing to the efficiency of data storage and processing in a digraph context. On the other hand, the systematic approach in the creation of BD_MLD transcends mere graphical representation, aiming to overcome challenges associated with logical data independence. By enabling the unambiguous manipulation of the structure as a whole or of specific parts, the methodology employed seeks to mitigate the internal disturbances in the nodes or surrounding edges, often present during structural changes. This approach not only preserves but also strengthens logical independence, allowing adjustments to the graph structure without compromising connectivity and semantic potential in queries.

Regarding queries, considered to be of high level and currently supported by several algorithms dedicated to graphs, we propose a contribution to obtain a strategy to facilitate these consultative tasks. In Figure 25 a digraph resulting from the aggregation of others MLD, which are also digraphs, at various levels of abstraction. This example visualizes the complexity of the aggregation process and highlights the need for advanced techniques to maintain structural and semantic integrity when dealing with multiple levels of abstraction in BDOG.

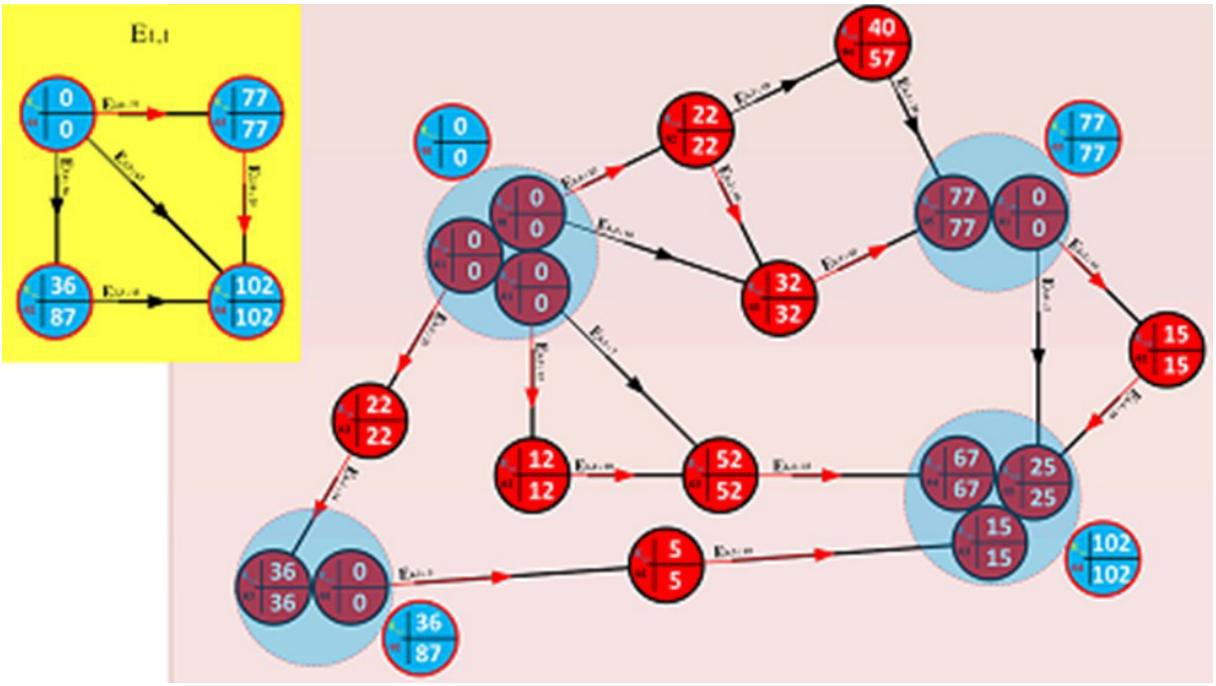


Figure 25 – Digraphs (MLD) resulting from the process of aggregation of other MLD (Ribas 2015).

2.6.1 Superior Structure in MLD

Later in chapter 3, we will see that directed hypergraphs, which substantiate flow hypergraphs, also appear as generalizations of digraphs and find application in the representation of binary relations between subsets of a given set. These relationships are observed in several areas of Computer Science, especially in SBD. To create hypergraphs, partitioning algorithms commonly perform the "partitioning" of an existing graph, followed by a refinement of the partitions, and then an aggregation of the parts, often accompanied by further refinement, which can lead to eventual loss of information. Faced with a real scenario, in which we can often be faced with the presence of billions of entities in a single graph, it becomes imperative to design an improved formulation for a superior structure composed of parts structured in (Luiz, Guedes, and Markenzon 2004, G. Ausiello, D'Atri, and Sacca 1986) MLD, where it can offer not only the ability to record the pertinent data and its metadata, but also to instantiate, uniquely, the structure of the graph itself. As an example, still in the context of civil construction, production processes are intrinsically interconnected, often depending on other productive and non-productive processes. These processes are incorporated into others and, over time, undergo constant and recurrent mutations, influenced by new configurations of the environment, which are recreated by the expanding

open system itself. In this sense, when creating a BDOG to support the governance of a given problem, it is essential that this support is systemic and as direct as possible.

The starting point for instantiation of BD_MLD is an empty DB_0 (Figure 26), level 0, which can also be called root hypergraph (expanding). That is, the first *Databases* (DB) instance created will necessarily be at level 1. However, it is admissible that other BD are created at level 1. For each DB, the level of abstraction is established by the cardinality (+1) of the level where its parent DB is located. There is no upper limit to the cardinality of the abstraction levels on the part of the proposed data model. The number of levels is only limited to computational issues of memories and the like. The creation process is completed when no BD_n exists pending to be defined or validated. At this moment, also an algorithm dedicated to aggregation identifies the total of levels, n corresponding to the DB created, and in this sequence, begins at each level, from the highest level to the level, n=0, the aggregation process to All the Created comics. Aggregation traverses all databases created. At the limit of processing, this procedure culminates in the constitution of a global digraph to support the DB, whose final structure corresponds to an elementary directed graph that occupies level 0. This global digraph allows, in a conventional way, the performance of queries to the graph structures, since, at the end of the processing, all the relationships between edges and vertices (V_{en}) of all databases created in multilevel are explained. In the case of graph-oriented databases, this task, although possible, becomes quite delicate, or even complicated even for a small database, because the graph (the DB) is easily configured in thousands of entities, whose visibility of the associations is impractical to the database manager. For example, let's suppose a digraph, the result of a small database, from which we simply want to eliminate one of its identities or relation, (or even a fragment of the graph''), it is found that it is not possible to observe in the global graph, all its precedence paths, and it is also obvious that by eliminating any target entity, we will be compromising all its dependencies that orbit in deeper data layers in itself, that in addition, all dependencies of global depth, which were pre-established in the primitive graph, will be forcibly broken, unless the entire graph, i.e., the database, is created again. In this context, a methodology was developed, capable of enhancing the creation and manipulation of the data structure for a graph-oriented database (digraphs), capable of ensuring the possibility of objectively manipulating part or all of the data structure, through the programming of a necessary computer tool that is capable of making the systemic creation of the structure of a graph-oriented database and transmutation[11] of the data structure.

[11] It refers to the ability to make changes to the graph structure in a dynamic and adaptive way, enabling a more flexible management of data in the graph-oriented database.

To define this possible structure of a BD_MLD, in the context of constructive processes, let us see it as a universal identity of abstraction, Figure 26, may be established. This universal identity is taken into account a number j (mutable), which is the index of certain levels of abstraction N_j, where for each of these levels N_j is incorporated a set of $E_{j,i}$ elements, elements that commutatively represent resources or products, according to the direction of sweeping between levels is made, denoting as a "creative" sense, when the sweep is made from level N_1 to level N_k. The starting elements are considered products, and in the opposite direction, the "aggregator" sense, when the sweeping is done from level N_k to level N_0 and then the source elements are seen as resources. Excluding from that switching principle, the elements of the set referring to the first level N_1, which are to be considered only as commodities, and the elements of the set referring to the last level N_k, which represent atomic resources and products.

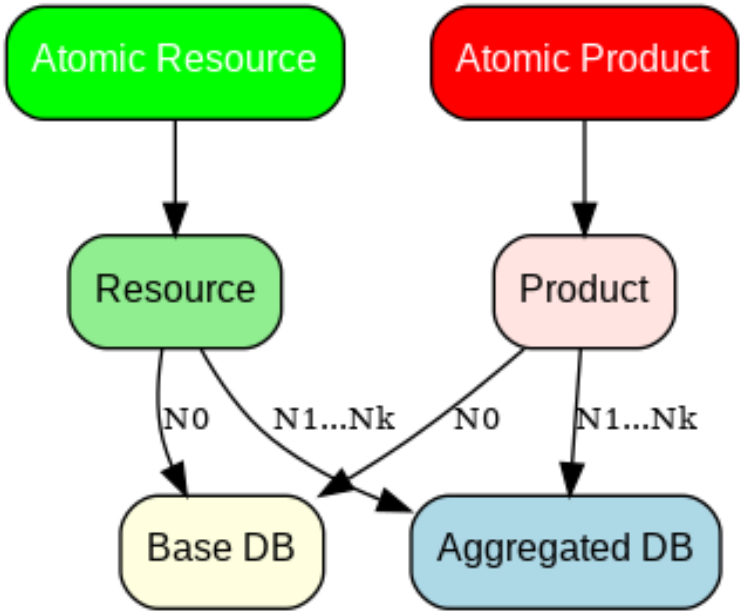


Figure 26 – The universal identity of abstraction.

For a given project of creating a BD_MLD, which aims to dominate its data analytical structure, each element $E_{j,i}$ of the set of dominant elements at the N_j level, (j> 0) can be defined and labeled in a list, as well as, for each one, lists of parameters of interest and their dependencies among the other elements of the lower level will eventually be created, N_{j+1}. The definition of all the elements $E_{1,i}$ of the set of elements corresponding to the base level N_1 will give sustainability to the first draft of the desired DB,

for which the first "logical paths" of the relational flows between its elements are defined, taking into account factors, determined by the Manager's experience, creating at that level, for this purpose, the proper precedence links of that base database.

Similarly, in the other levels N_j, (K>j>1) the procedure can be recreated for each $E_{j,i}$, (K> j> 1), considering the $E_{j+1,i}$, up to j<K. That is, in each level of abstraction N_j, (K>j>0), and for each $E_{j,\ i}$, we will be facing a digraph, $G_{j,i}$ ($V_{j+1,n}$, $E_{j+1,r}$), in which V_{j+1},n, are the vertices and $E_{j+1,r}$ the edges (some of the elements of the N_{j+1} level), an element that, taking into account the direction of aggregator sweeping and before $G_{j,i}$, is switched from product to resource. In this process of creating the $G_{j,i}$, there will naturally be the need to increase the $E_{j+1,r}$, successively up to $E_{k,r}$. In this sequence, when it is a question of the set of the last level, N_k of the universe considered for the BD, its elements $E_{k,i}$ are called atomic resources or products.

Also in this sequence, that the number of levels of abstraction for the creation of a *Digraph-Oriented Databases* (BDOD) is verified, as well as all the N_j, (k-1>j> 0) levels, that after this recontextualization it can be clarified, that this approach reflects the knowledge of their respective sets of elements $E_{j,i}$, and that the obtaining of the BDOD structure can go through the following phases:

- F_1- In the first phase of this procedure, taking into account the mastery of the BDOD, the first draft of the base BD should be prepared, BD_0 at level, N_0, using $E_{1,i}$, considering for this purpose a digraph $G_{0,i}$ ($V_{0,n}$, $E_{1,i}$);
- F_2- In a second phase, the BDOD, in the "creator" sense, will scan from the N1 level to the N_k level, which results in the meticulous construction, in each N_j, (j>0) of all the digraphs, $G_{j,i}$ ($V_{j+1,n}$, $E_{j+1,r}$), where Vj+1,n are the vertices, $E_{j+1,\ r}$, the edges (some of the elements of the j+1 level). As an unequivocal result of the construction of each digraph, $G_{j,i}$ ($V_{j+1,n}$, $E_{j+1,r}$), its respective defining matrices will be obtained, namely the adjacency matrix and the incidence matrix by reading the intrinsic code (holocode) of each subsystem created;
- F_3 – In this last phase, in the aggregating sense of the sweep of the level N_k up to N_1, in which for each level N_j (0<j<k) visited, and for each of the elements $E_{j,i}$, their corresponding digraphs, are rewritten on the basis of atomic resources and products, giving rise to $G'_{j,i}$ ($V_{j,\ m}$, atomic RP), at each aggregator level, and that at the limit of this procedure, when j=0, $G'_{0,i}$ is obtained, which is the digraph that translates the desired $BDOD_0$, that is, it will be possible in this base to obtain the respective adjacency, incidence and weighting matrices in that base through the reading and processing of the respective holocode now aggregated.

The rewriting process will only be controlled if there is the availability of a functional algorithm, possible to be applied to generic digraphs of the type G(VM,AN), consisting of only one initial vertex, f called source, and only one final vertex, s, this one called sink, in which it is possible to obtain a digraph resulting G'(V'$_{M^*}$, A'$_{N^*}$), by application {G(V_M,A_N),AT(u,v),G*(V*P,A*Q)} → G'(V'$_{M^*}$,A'$_{N^*}$), where G*(V*P,A*Q) is also a digraph of a single initial vertex, f* (source) and a single final vertex, s* (sink), such that G'(V'$_{M^*}$,A'$_{N^*}$), is constructed by the direct exchange of any A_T edge of vertices (u, v) belonging to G, where you and v, respectively are the vertex "tail" and the vertex "head" of that edge, both belonging to G (Ribas 2015).

That direct exchange involves matching the vertices u and v, of AT, belonging to G, respectively (u →f*) and (v →s*), leading to a total integration of the vertices V*P, and the edges A*Q, belonging to G* in G, whose renumbering remains topological, and that are made, without loss of relative position in these sets, and that they are remade in an integrated way in M* and N*, giving rise to the definition of the resulting digraph G'(V'$_{M^*}$,A'$_{N^*}$) intended. The programming of that functional algorithm is one of the basic objectives to be attempted throughout the development of the thesis.

2.6.2 MLD Data Structure

By analyzing a digraph specific acyclic value, we observe that each element belonging to the set of reference values intrinsically constitutes a digraph. As outlined in the formal characterization presented by (Ribas 2015), a digraph can be characterized by three fundamental parameters: the integrity state, the blocking state, and the operational state, as illustrated in Figure 27. In context of the "State", two distinct possibilities arise: the subdigraphs can be considered complete, indicating the existence of a validated holocode for that subdigraphs, or, conversely, it can be classified as incomplete, denoting the absence of valid coding. Regarding the "block" parameter, two options are also defined: the subdigraphs it can be free or fixed, a distinction that manifests itself only during the process of manipulation. The "state" parameter of the subdigraphs, determined by their active or inactive status.



Figure 27 - Formal characterization of a given subgraph (Ribas 2015)

About the sequence of operations on these values, considered as a mathematical construction that can be implemented by means of a specific data structure, it is relevant to mention their fundamental transmutation operations. According to Ribas, the procedure to perform a specific transmutation in each digraph requires the existence of basic functional operators. By manipulating these operators within a dedicated application, they can be interpreted as tools. In this context, six basic operators proposed for development have been designed, namely:

- Creating or Canceling Events. In the context of a specific digraph, the creation or cancellation of events can be requested by the database manager. This action is subject to the verification of the absence of dependencies in any digraphs, in the case of events;
- Activation or Deactivation of a Complete or Incomplete digraph. The activation or deactivation of a complete or incomplete digraph requires the application of dedicated algorithms to be developed to ensure effectiveness in the process;
- Conversion of an Elementary digraph to an Incomplete digraph. The automatic conversion of an elementary subdigraph into an incomplete digraph must be conducted after due validation by the database manager;
- Reconversion of an Incomplete digraph into an Elementary digraph and vice versa. The reconversion of an incomplete digraph into an atomic subdigraph follows a procedure analogous to that of neutralization. In this case, the outer subdigraphs will have the category of atomic, instead of connectors;
- Creation of an Instance in digraphs. Instantiation occurs in a guided manner, always demanding command and decision from the database manager. It consists of the creation of a new digraph

with the same structural typification of an existing complete or incomplete digraph, ensuring flexibility and adaptability to the system.

To establish more detailed internal structural rules, this proposal incorporates two specific types of singular subgraphs: atomic and neutralizing (or fictitious). Figure 28 illustrates potentially singular subgraphs, referred to as atomic subgraphs, which are composed of atomic, connector, or fictitious elements. These subgraphs are termed atomic due to their unique composition of atomic, connector, or fictitious components

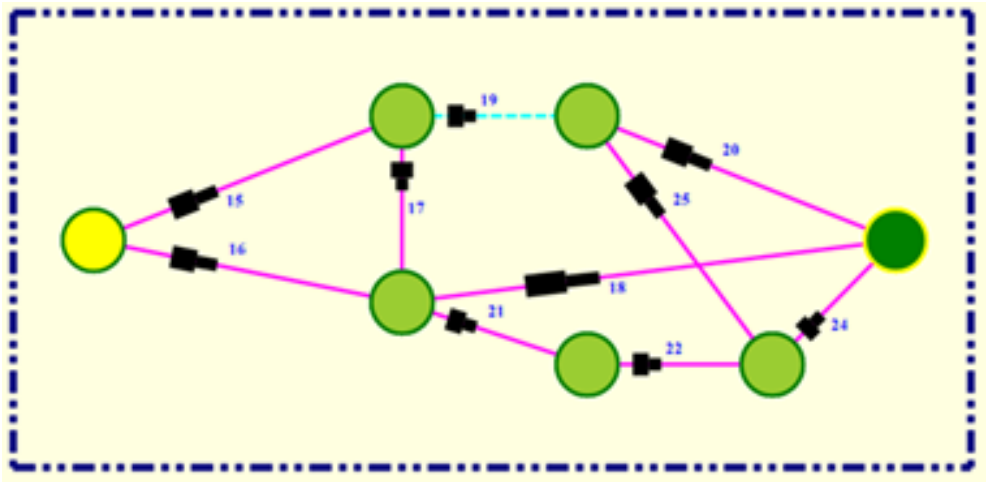


Figure 28 -Subdigraphs atomic (Ribas 2015).

In the context of atomic subdigraphs, each element plays a key role in internal structuring. Atomic subgraphs, as fundamental building blocks, form the structural basis of the broader atomic subdigraphs. Connectors facilitate integration and connection between elements, while dummy elements have a neutralizing role in certain operations. This diversity of elements gives the atomic subdigraphs a unique versatility, allowing their participation in different transmutation operations. A more detailed analysis of these elements and their interactions provide a deeper understanding of the internal structure of the subgraph, providing a basis for the associated mathematical operations and for the practical implementation of these structures in graph-oriented database management systems. The creation of a given subdigraph is therefore substantiated by the construction of a digraph, which results in a valid encoding (TDA).

Due to the necessary combination between the verification commanded by the absence of errors and the typification of its constituent elements, it was necessary to enhance a tool, according to Figure 29, capable of generating the intended geometric "creation" in an environment of strong iteration with the Database Manager, capable of providing the possibility of a dynamic and communicative handling of the creation of the digraphs in which abundant information is provided on whether or not there are errors in the data model, namely:

- Existence of events without the possibility of flow;
- Existence of loose, unconnected events;
- Existence of internal cycles, implying errors in the logical construction adopted by precedence;
- Uncontrolled existence of the various types of elements, (external, fountains, interiors and sinks);
- The sequencing in tables of the successor and predecessor edges;
- The possibility of validation of the current subdigraphs, when a valid holocode is recognized and displayed.

The existence of a valid TDA for a given sub-digraph, which is in the process of being completed or transmuted, should be achieved by its own validation, which should be done recursively by verifying the non-existence of the errors mentioned above, seeking synchronization with the development of the graph, which the DB is processing.

In terms of programming, its dedicated module presupposes a constant calibration of errors, which intermediates the existence or not of a possible holocode. That is, if there is a holocode, then it will be valid. Almost always, thoughts are shaped over time, as well as the reality of the projects that, at a given moment, is interpreted differently by the individual. To create greater modulation flexibility, a function capable of structuring the conversion of a complete subdigraph into an incomplete subdigraph was also implemented. This procedure was carried out in anticipation of the transmutation process, required by the BD.

In this methodology, digraphs do not have a fixed dimensional structure, that is, what exists is procedural behavior. The mutation of any sub-digraph does not occur independently of the structure of the entire database (of the digraph).

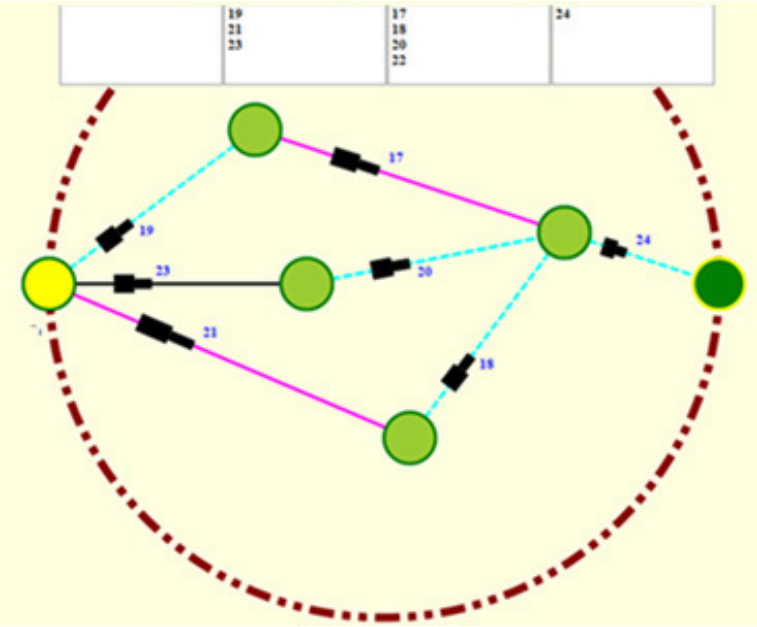


Figure 29 – Window management tool MLD.

Also in this follow-up, the possibility of instantiation of sub-digraphs was also implemented in order to simplify the procedures during the creator scan. Creating an instance in digraphs is the same as replicating (copying) a subdigraph, it gives the possibility of an even higher level of abstraction. An instance is, so to speak, a replica of an existing sub-digraph, which can be complete or incomplete, that is, it may be possible to take advantage of a small part of the data structure of a certain sub-digraph to be intentionally applied in another part of the DB structure, which is already clarified in some aspects, to keep in mind in your schedule:

- Create the structural instance option, with the typification of the element to be created;
- First, to go through the aggregating sense, so that it is possible to build the structural order;
- Choose the element (your IDE) to be instantiated;
- Reserve a new IDE for the future registration (of the new digraphs) of the instance;
- Insert the same DTA structure into the new instantiated digraphs, creating or copying new elements for all elements, for all their lower levels (deconstruction by ordinance, replicating records).

2.6.3 Aggregation algorithm for MLD

To functionally materialize the MLD, and following the existing coding of the holocode formulation, a function was implemented, called "Aggregation", substantiated in an algorithm created for this purpose, which we will describe below. However, to better describe how digraphs (*'system':* parent) and subdigraphs (*'subsystem':* children) are labeled and how the connections between them are managed within the context of the "holocode", here is a small excerpt from Ribas work:

"*... The hydrological terms mentioned in the design of the 'holocode... In this context, systems and subsystems are labeled as 'Streams,' 'Rivers,' and 'Seas.' The basis for the creation of a possible aggregation algorithm may lie in logic that mirrors the dynamics of nature. The concept of 'subsystem' is associated with the terms 'stream' and 'river,' just as that of 'system' is related to the term 'sea.' This is because streams aggregate or flow into rivers, and in turn, rivers join the seas. Therefore, when a stream connects to a river, that river is called a 'pseudo-sea' and remains a river if another stream tries to connect to it. When there are no more streams to connect, the rivers take on the role of streams, and the seas aggregate as if they were rivers. This cycle continues until all the last streams have been absorbed, at which point the last river assumes the condition of 'Sea.'"* (Ribas, 2015, p. 173).

The algorithm of aggregation of elements in an acyclic digraph is a technique created in this work, which materializes the MLD and which is intended to be fruitful in the universe of computation and information processing. The process begins with the thorough identification of the elements present in the input code (in the holocode). These elements are classified into five main categories: external elements, font elements, precedence elements, interior elements, and sink elements, identified by special characters in the code. Once classified, the algorithm proceeds to determine the overall structure of the digraphs, calculating the total number of elements and establishing their relationships. This results in the creation of an internal representation of the digraphs, making visible how these elements are connected. The organization starts with the external elements, which are those that have connections with elements external to the digraphs. This initial organization creates a basis for the following steps. Next, the algorithm addresses the source elements, checking if they have precedents and, if necessary, connecting them appropriately. This ensures that these elements are positioned correctly in the digraph. The interior elements are treated with equal importance. Like source elements, the algorithm analyzes their relationships and positions according to their dependencies.

Next are the sink elements, those that have no digraph output connections. The algorithm also takes precedence elements into account, determining their roots and how they relate to other elements in the digraph. The algorithm maintains, throughout the process, a hierarchical organization of the elements, ensuring their logical arrangement in accordance with the existing dependencies. The final stage consists of the construction of the resulting digraph, which is presented as a clear and intelligible representation of the elements, with all connections and relations of dependence duly consolidated. In the reorganization stage, the elements are grouped into distinct subsets according to their categories, which makes it even easier to understand the digraphs. The establishment of connections between the elements is desired, as they can depend on each other, ensuring the accuracy of the representation. The elements of precedence are treated with special attention, and their roots are determined to maintain the correct hierarchy in the digraphs. With all these steps completed, the algorithm finalizes the construction of the final digraph, an organized representation of the elements. The benefits of aggregation are numerous, since such an operation simplifies the analysis of complex data, improves the efficiency of information processing and facilitates the understanding of the relationships between the elements. The algorithm has practical applications in workflow systems, databases, and project management systems, making it essential for organizing and interpreting information in various areas.

MLD play an important role, as they enhance the creation of BDOD models. This empowers practitioners to represent and analyze complex systems, allowing for a deep understanding of the relationships between the elements. The introduction of the aggregation algorithm for MLD marks a advance in the development of Chapter 5, promoting convergence between BDOG and MLD. This merger reinforces the ability to represent complex data, providing a more complete and contextualized view. By incorporating MLD decision modeling, BDOD acquires a more robust analytical structure, allowing dynamic relationship management and flexible adaptation to changing project management requirements through CRUD operations.

2.6.4 CRUD Operations in MLD

The integration of CRUD operations in MLD, guided by holocode supported by the aggregation function, as we will see in Chapter 5, plays a key role in the efficient manipulation of data in MLD. In the creation operation, the developed aggregation algorithm ensures the cohesive insertion of new elements into the existing digraph. The read operation takes advantage of the organization provided by the aggregation

function, optimizing the search and interpretation of information. When addressing the update, the aggregation function preserves consistency after modifying elements, while on deletion, it rearranges MLD to reflect the changes and ensure structural integrity. This integrated approach illustrates the importance of the aggregation function at all stages of the data lifecycle, contributing to the effectiveness and coherence in complex information management. To systematically highlight the relevance of the aggregation function in the life cycle of MLD data, the detailed analysis of each of the CRUD operations is presented below:

- *Create*. Creation is an operation that adds new elements to the digraph by applying holocode as a means of identification and integration. The course of the creation process is as follows. First, the instances are defined according to the required additional elements. First, the elements that are identified in the holocode are assigned to four initial categories based on the special characters in the code. Second, the aggregation algorithm is activated, assuming that newly introduced elements are incorporated into the pre-existing digraphs. This means the arrangement of these elements based on their dependencies and connections to the elements already present in MLD. Thus, the aggregation function is to create a logical environment for the digraphs, ensuring that new elements are incorporated into it in a coherent way.
- Read. The read operation in MLD involves retrieving specific information from the digraph using holocode as a guide. In this process, holocode is used to identify the desired elements in the MLD. Although the aggregation function is not directly involved in the read operation, it is necessary for the organization and simplification of the MLD structure. This makes the consultation and interpretation of information more efficient, since the digraphs are organized in a clear and logical way.
- Update. The update operation in MLD involves modifying existing elements identified by holocode. Here, the aggregation function plays a vital role. After the desired elements have been identified and modified, the aggregation function comes into play to ensure that dependencies and relationships between elements are maintained. Reorganizing MLD is essential after the update, and the aggregation function helps adjust the affected elements, ensuring they remain consistent and coherent with the new configuration.
- Delete. The delete operation in MLD involves removing specific elements identified by the holocode. Here's how the aggregation function is indispensable. After the elements are deleted,

the aggregation function is used to reorganize the MLD, ensuring that the structure is adjusted to reflect the changes resulting from the removal. Dependent elements are re-evaluated, connections are adjusted, and the integrity of the MLD is maintained.

In summary, the aggregation function plays a critical role in all CRUD operations in the context of MLD, ensuring that the structure of the digraph remains consistent and coherent, as well as simplifying the organization of complex information and facilitating the analysis of relationships between elements, making MLD management precise. It is an essential tool for the maintenance and evolution of systems that use MLD in their data structure.

3 Hypergraphs

The theory of hypergraphs has caught the attention of many researchers over the years. The concept of hypergraphs was first presented in 1973 by Claude Berge, in the book "Théorie des graphes et ses applications" (Bergé, 1973). From that time on, hypergraphs were used in a wide variety of machine learning and optimization applications, network analysis, and many more. Hypergraphs offer a complete way to describe relationships between elements, which is best suited for modeling more complex interactions in various scientific areas. This approach is particularly useful when the connection between entities is not binary, because the relationships occur in higher-order aggregates. Hypergraphs are particularly useful for modeling systems that involve cluster or level relationships. The main fields of application of hypergraph include, for example, complete and broken networks, scientific co-authorship, social networks, practical engineering projects, and machine learning problems, among others. Throughout this chapter, a framework for understanding hypergraphs will be established and discussed.

3.1 Hypergraph Modelling

Modelling dynamical systems using hypergraphs is an emerging field of research. Understanding how the relationships and interactions between the various elements of a hypergraph evolve over time is the focus for the analysis of dynamic networks, such as social networks or biological networks. According to Dai and Gao (2023), hypergraph modeling aims to establish a high-quality hypergraph structure to accurately formulate high-order correlation between data. Different modeling methods can be used to construct hypergraphs. These methods, according to the same authors, can be organized into two broad categories, depending on whether these correlations are explicit or implicit (Figure 29).

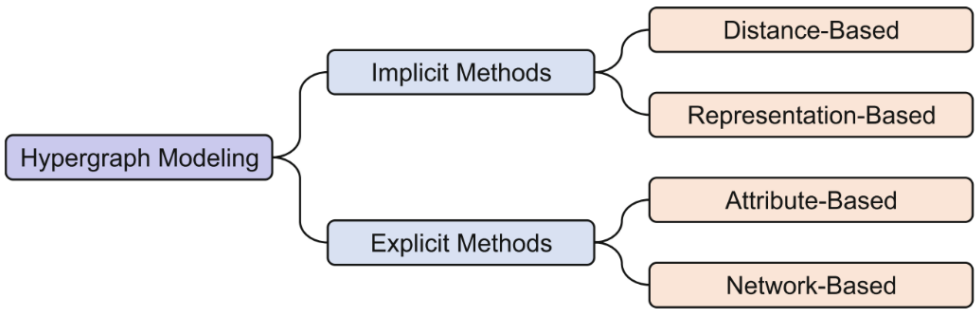


Figure 30 - Different categories of hypergraph modelling methods - adapted from (Dai and Gao 2023).

The process of network segmentation in favor of a given classification method underlying the modeling of real situations is a key challenge for computing, so multiple layers of network segmentation are considered, so called due to the various steps that structure the entire process. Network partitioning is therefore an essential technique in computing, providing a structured approach to the analysis and categorization of complex data represented by a network. In this context, the network is visualized as a graphical representation of the connections between entities, in which the nodes represent elements and the edges indicate connections or interactions. Before partitioning, we can take several types of approaches to building the network. An elementary strategy could be the creation of the network through the controlled aggregation of small structures. This procedure would involve the gradual combination of smaller components or subnets to form a broader, more meaningful structure. Controlled aggregation can enable the formation of more cohesive clusters or groups within the graph, representing similar entities or patterns. This approach lays a foundation for the subsequent partitioning process. Graph partitioning is then performed to divide the structure into distinct sections or communities. Such communities can represent more homogeneous datasets or significant groupings of elements. This division facilitates the analysis, understanding, and subsequently the process of classifying data. By partitioning a graph, we effectively organize the elements into more manageable groups, simplifying the task of classification. This strategic process allows for the application of specific classification algorithms in each community, better adapting to the local characteristics of these smaller datasets. Another important challenge is the generalization of well-established concepts and theorems from graph theory to hypergraphs. Extending metrics, algorithms, and properties from traditional graphs to hypergraphs requires careful study and a deep understanding of the structural differences between these two structures. Each type of hypergraph has its own advantages and disadvantages. Choosing the most suitable hypergraph type depends specifically on the area of your application (Tian et al. 2022).

Hypergraphs are undoubtedly a promising tool for analyzing complex data, but their practical application still presents several challenges. In Data Science, defining centrality measures and routing in hypergraphs are open problems. But there is already some work done: they proposed a measure of hyperedge centrality to measure the importance of hyperedges in the dissemination of information or influence through a hypergraph. In engineering, the main difficulties are related to computational complexity and scalability of algorithms for data analysis and visualization and the lack of specific tools to work with hypergraphs. Aware of these difficulties, several label propagation algorithms have been developed for hypergraphs. See, for example, the *Label Propagation Algorithm* (LPA) (Xie et al. 2023),

which is a great contribution to the detection of communities in hypergraphs and has opened paths for the resolution of routing problems. Going in, with reference to the LPA, although it is a classic method, it has the problems of randomness and instability (Li et al. 2021).

Hypergraphs are particularly versatile structures, whose morphology can be adapted to different contexts of representation and analysis. Among its most relevant classifications are hierarchical hypergraphs, in which vertices and hyperedges are organized according to different levels or layers, which allows a structured modeling of interactions at multiple scales. Based on this conceptualization, the following Chapter 4 proposes the introduction of a new class called MLHG, which extends the traditional hierarchical paradigm by incorporating multiple dimensions of abstraction and interdependent connectivity. This new structure gives increased flexibility to the modeling of complex systems, by allowing the simultaneous representation of vertical (between different levels) and horizontal (within the same level) relationships, which favors the description of phenomena with interconnected layers of information or governance. MLHG are a promising tool for the representation, analysis and manipulation of highly complex systems, with potential application in domains such as Data Science, Systems Engineering, Organizational Networks or Cognitive Structures. The introduction of MLHG as a natural extension of multilevel partitioning algorithms represents a significant conceptual and practical advance in the modeling of complex systems. While traditional algorithms deal with binary graphs – that is, with edges that connect pairs of vertices – hypergraphs, by their very definition, allow a hyperedge to connect multiple vertices simultaneously, offering a richer and more expressive representation of n-ary relationships, such as functional groups, interdependent components, or concurrent tasks in collaborative environments. The adaptation of the multilevel partition paradigm to hypergraphs implies, first, rethinking the concept of partition boundary, since the shared belonging of vertices to multiple hyperedges can induce cross-interactions between clusters in a non-trivial way. This requires algorithms capable of maintaining the balance between granularity, internal cohesion and minimization of connections between partitions, now in the presence of higher-order interdependencies.

Multilevel graph partitioning algorithms are so named because of the various phases that make up the entire partitioning process. Initially, a graph G is reduced to a few hundred vertices, and then this reduced graph is partitioned, and finally this partition is projected back to the original graph. During the return to the original graph, refinements are performed on the partition, with the aim of reducing the number of vertices on the boundary between domains. In (LaSalle and Karypis 2015), authors provide an illustration of the various phases of this type of algorithm (Figure 31).

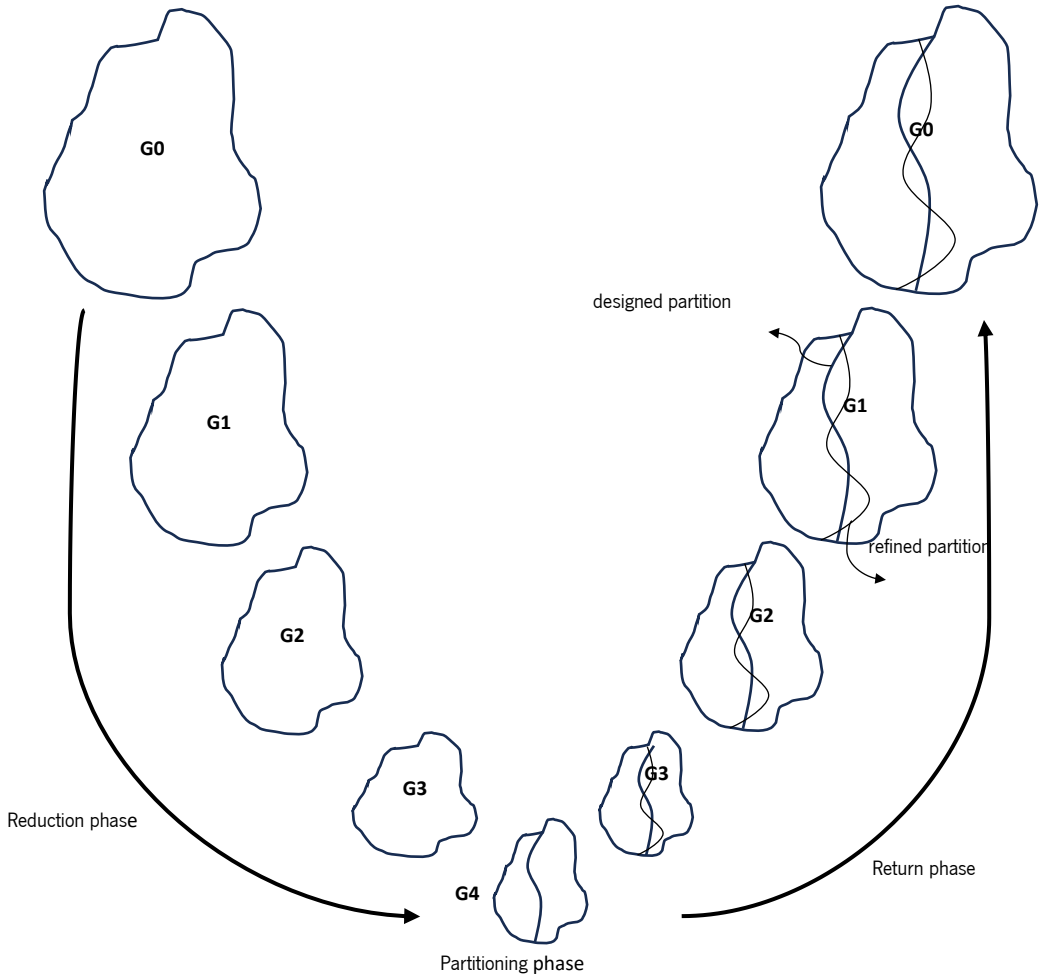


Figure 31 - Graphical representation of the various phases of the multilevel algorithm (Karypis et al. 1999).

In the reduction process, a sequence of smaller graphs $G_m = (V_m, E_m)$ is constructed from the original graph $G_0 = (V_0, E_0)$, such that $| V_m | > | V_{m+1} |$. The graph G_{m+1} is constructed from G_m and the maximum selection of branches Mm is found $\subseteq$ In G_m and collapsing the vertices that are incident to each branch of the selection. The maximum selection consists of choosing several sets of adjacent branches, which are not incident at the same vertex. Vertices that are not incident to any branch of the selection are simply copied to the graph G_{m+1}. In Figure 32 The different forms of branch selection in the process of reduction of the same graph are presented. It is visible that the nodal indices and the indices in the branches indicate, respectively, the number of nodes and the number of branches that collapsed together.

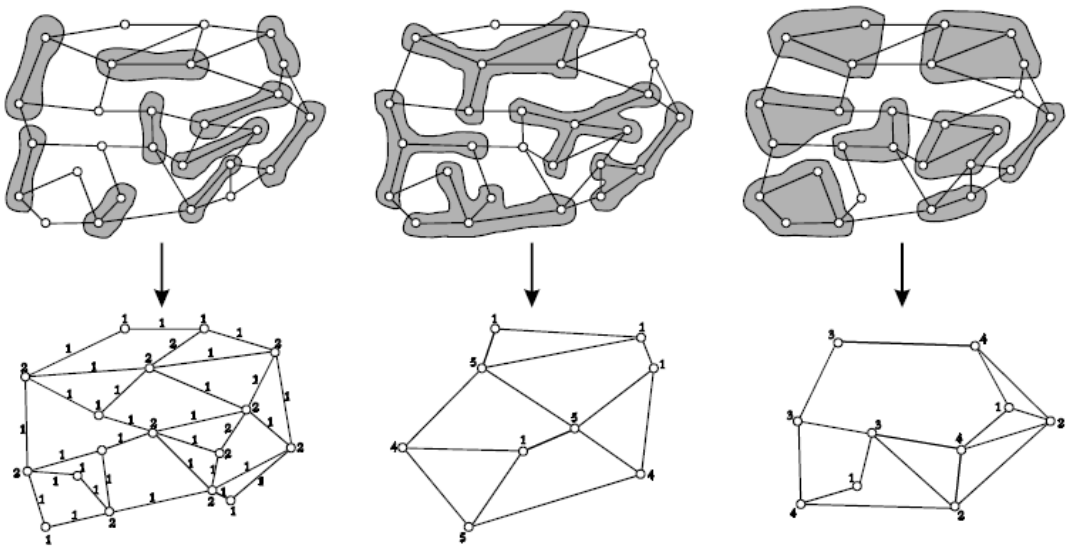


Figure 32 - Different ways of selecting branches and reducing the same graph (Moretti et al. n.d.)

In the opinion of the author, those different forms of selection of branches are a restriction to the reduction process itself, as it makes the algorithms in question that are implemented with something deterministic. This, of course, will depend on the ordinance of the entities (vertices or edges) in each research organizational structure. Thus, compromising the desired two-way correspondence, which ideally should be as complete as possible.

When vertices v, u, $\in V_m$ are collapsed to form the vertex w $\in V_{m+1}$, the weight of the vertex w is defined as the sum of the weights of the vertices v and u, and the branches incident to w are the branches incident to v and u, except for the branch (v, u). If there are two branches starting from the same vertex that are incident at v and u, respectively, then the weight of the resulting branch after the collapse of vertices v and u is equal to the sum of the weights of those two branches. Therefore, during the graph reduction phase, the weight of both vertices and branches is increased. The maximum selection can be made in different ways. The method used to find the maximum selection affects both the quality of the partition and the time required for the phase of returning to the original graph.

The second phase of a multilevel partitioning algorithm consists of determining a bisection of the graph $G_k = (V_k, E_k)$ such that each part contains approximately half the weights of the vertices of the original graph. Since, during the reduction, the weights of the vertices and branches were determined to reflect the weights of the original graph, G_k contains enough information to make a balanced partition with the minimum boundary between the parts. G_k partitioning can be done using various algorithms, such as

spectral bisection[12]. During the return phase, the partition of the reduced graph G_k is projected back to the original graph through the graphs G_{k-1}, G_{k-2}, ..., G_0. Since each vertex $u \in V_m$ contains a distinct subset of the vertices U of V_{m-1}, designing the partition from Gm to $Gm-1$ consists simply of associating the vertices in U with the same partition to which the vertex u belongs. However, even if the Gm partition has the minimum boundary between the parts, this does not guarantee that the projected G_{m-1} partition also has the minimum boundary, Since the G_{m-1} graph has more degrees of freedom, the partition can be optimized and consequently reach a smaller boundary than the one obtained by the Gm boundary projection. For this reason, after partition design, a partition refinement algorithm is used. The basic purpose of the partition refinement algorithm is to select two subsets of vertices, one from each part, such that when exchanged between the parts, the resulting partition has the minimum boundary. More formally, if A and B are two parts of a partition, a refinement algorithm selects $A' \subset A$ and $B' \subset B$ such that $A - A' \cup B'$ and $B - B' \cup A'$ results in a bisection with a minimal boundary[13].

A class of algorithms that tends to produce good results are those in which given an initial partition, in each iteration, the algorithm finds subsets A' and B' with the properties described above. If such subsets exist, then they are exchanged between the parts, and this new partition is used as the initial partition of the next iteration. The algorithm continues to repeat the entire process (Moretti et al. n.d.). If it is not possible to find those two subsets, then the algorithm terminates, since the partition reached is the one with the minimum boundary and new optimizations cannot be made by the algorithm. Since the projected partition, used as the initial partition, is already a good partition, this type of algorithm usually finds the minimum boundary with a small number of iterations.

Multilevel graph partitioning algorithms have been shown to be effective in splitting complex graphs into smaller, more manageable subgraphs. However, to deal with even more intricate problems and more accurate representations, it is possible to enhance the use of hypergraphs in this context. Hypergraphs, unlike traditional graphs, allow edges to connect more than two vertices simultaneously. This means that they are best suited for representing complex and multifaceted relationships between the elements of a system. By incorporating hypergraphs into multilevel graph partitioning algorithms, we can improve the ability to model and analyze complex systems.

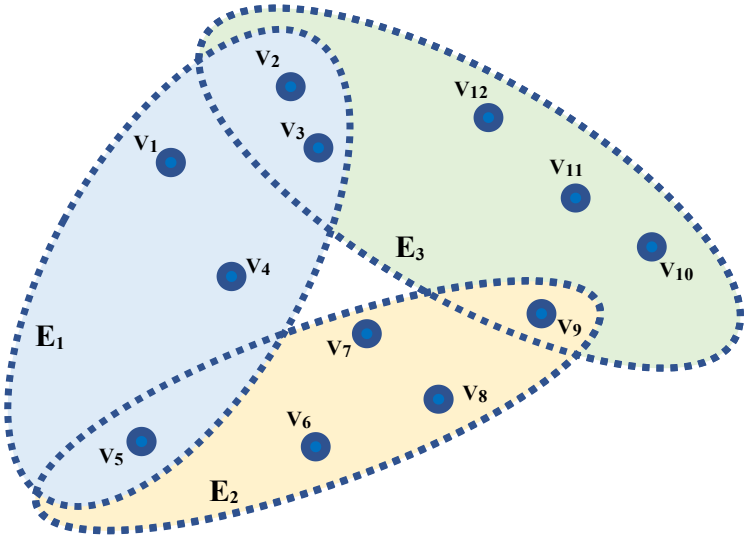
[12] A method that uses eigenvectors of the Laplacian matrix of a graph to divide the graph into two parts with balanced weights and a minimum boundary.
[13] A way to divide a graph into two parts so that the number of edges connecting vertices of different parts is minimized.

3.2 Definition and Characteristics of Hypergraphs

Given the potential of hypergraphs in the representation of complex relations, it is necessary to understand not only their general structure, but also the formal foundations that support their mathematical definition. After introducing the applied and theoretical context of hypergraphs as a natural extension of conventional graphs, we now delve into the structural elements that formally characterize them. This approach aims to provide an overview for understanding the mechanisms that distinguish hypergraphs from binary relational structures, highlighting the aspects that make them particularly apt to model phenomena with higher-order interactions. In this sequence, we move on to the formal definition and the different modes of representation of hypergraphs, which are fundamental for computational implementation and for the analysis of their structural properties. These modes of representation will be explored in the following sections.

A basic definition of a hypergraph consists of a set of vertices and a set of hyperedges, where a hyperedge is a subset of vertices. Unlike graphs, which are made up of binary edges, hypergraphs allow a hyperedge to have any number of vertices, from a single element to the inclusion of all vertices present. Formally, a hypergraph H is an ordered triple (V, E, I), in which V is a set of nodes (vertices), E is a set of hyperedges, and I is an incidence function that associates each hyperedge with a subset of V. The incidence function I is defined as $I: E \rightarrow P(V)$, where P(V) is the set of all subsets of V. through algebraic, incidence, adjacency matrices and their respective lists, as well as using set-based approaches. Each of these representations must be carefully chosen, considering the intended modeling, as each of them has specific advantages and applications (Ouvrard 2020).

Another basic issue, referenced by Ouvrard (2020), is the visualization of hypergraphs, in which he states that Mäkinen classifies visualizations of hypergraphs into two categories called "subset pattern" and "edge pattern". The subset pattern, the most common, reflects that hyperedges are subsets of the set of vertices (Euler diagram) – the vertices of a hyperedge are drawn as points and the hyperedges as closed lines. In Figure 33 We can see the illustration of a hypergraph. The design of its vertices and the three hyperedges are represented, as well as their respective incidence matrix. This is the most common matrix representation of hypergraphs, because it clearly expresses the set of vertices for each hyperedge.



	E_1	E_2	E_3
V_1	1	0	0
V_2	1	0	1
V_3	1	0	1
V_4	1	0	0
V_5	1	1	0
V_6	0	1	0
V_7	0	1	0
V_8	0	1	0
V_9	0	1	1
V_{10}	0	0	1
V_{11}	0	0	1
V_{12}	0	0	1

Figure 33 - Example of a hypergraph and its incidence matrix.

In the work of Ouvrard (2020), revised for this work, some visualization forms of hypergraphs were identified, morphologically derived from Euler's diagram. However, the same author still referred to a primitive representation, namely that of Zykov (1974), which incorporates extra nodes and connections to represent hyperedges, whose computational implementation is quite expensive, as it implies the creation of a global graph with high structural redundancy.

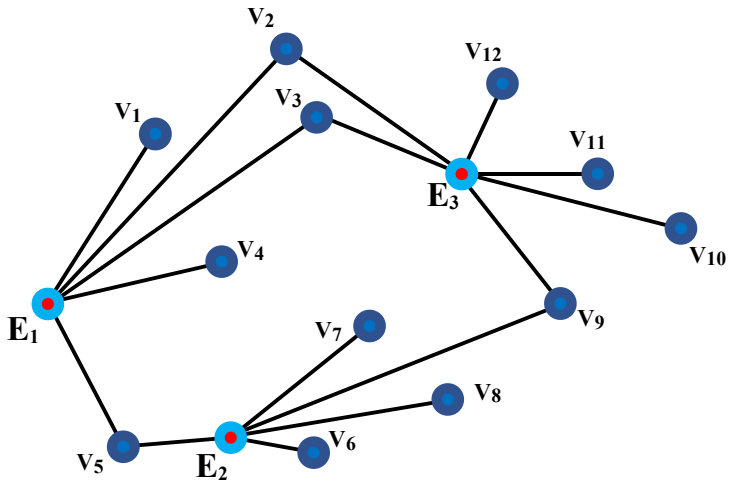


Figure 34 - Hypergraph representation using extra nodes.

Let us now analyze generalized hypergraphs. These are hypergraphs that allow a hyperedge to contain any number of nodes. This includes uniform hypergraphs but also allows for hyperedges with different sizes. These hypergraphs are often used to model complex relationships between objects. For

example, in a hypergraph that models scientific collaboration processes, each node can represent a scientist and each hypergraph a paper that was written by a group of scientists. Science is fueled by innovative collaborations and discoveries. By looking at the example presented in Figure 35, referring to the interconnected research universe, we identified twelve scientists - Alice, Bob, Claire, David, Emily, Frank, Grace, Henry, Irene, Jack, Kate and Luke - who share their scientific findings in three articles.

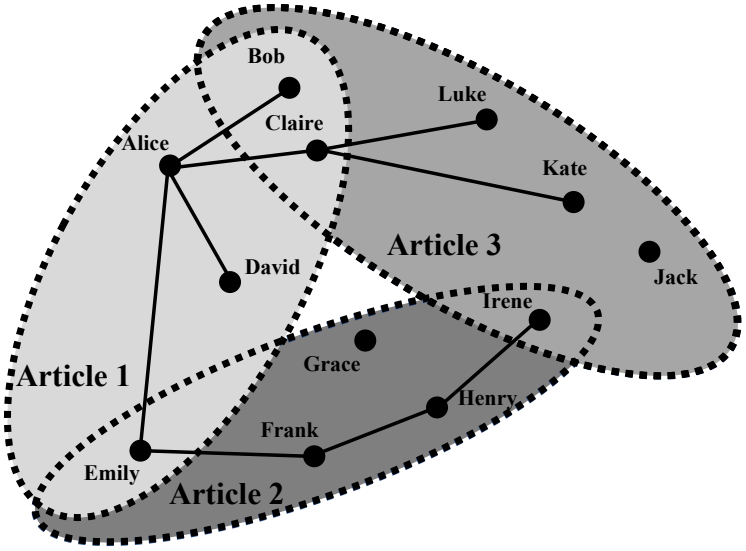


Figure 35 - Hypergraph modeling scientific collaboration

The first article, "Article 1", is born from the union of five of these scientists: Alice, Bob, Claire, Emily and David. Researchers cooperate in the analysis of a specific domain, integrating complementary contributions that lead to more robust results and a significant advance in scientific knowledge. Article 2, written by Emily, Frank, Grace, Henry, and Irene, focuses on a specific scientific field, distinct from the others. In turn, Article 3 brings together Bob, Claire, Irene, Jack, Kate and Luke, configuring a broader interdisciplinary collaboration, which brings together different areas of knowledge.

Hypergraph analysis reveals relevant patterns of collaboration among authors. It is noted, for example, that Bob and Claire act together in both Article 1 and Article 3, indicating a recurring partnership. Although they did not participate in the same studies, Henry and Irene demonstrate an indirect association, suggesting academic links within the network. Alice and David, on the other hand, appear exclusively linked to Article 1, signaling more targeted and specialized contributions in a particular domain. From this perspective, a hypergraph allows the exploration of connections and relationships. The intricate network of scientific collaborations is therefore a rich source of possibilities, representing the union of

scientists in progress towards new discoveries. One of the main general applications of a hypergraph is the analysis of complex networks such as genes, proteins, and social interactions. According to the study, hypergraphs have been commonly used in the modeling of biological and social networks, allowing the modeling of various solutions for pattern detection Feng et al.(2020).

3.2.1 Partitioning Graphs Across Hypergraphs

The singular purpose of graph partitioning (Figure 36) is to divide a graph G = (V, E), into smaller subgraphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, ..., $G_k = (V_k, E_k)$ in such a way that certain desired properties are satisfied.

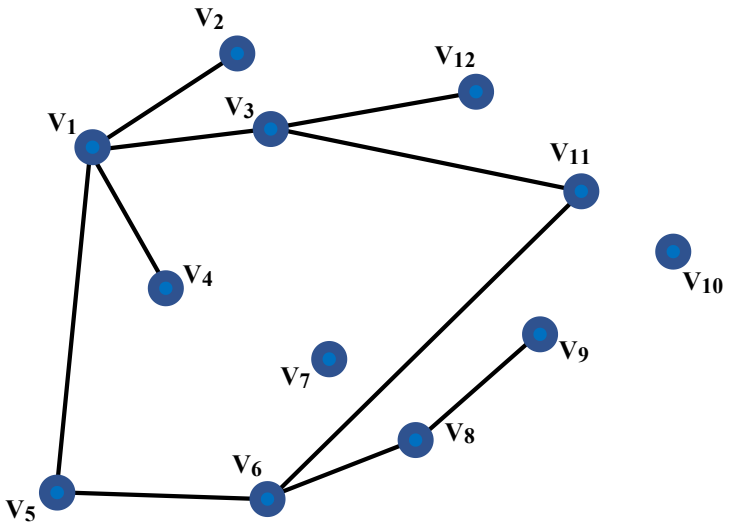


Figure 36 - Example of a graph G=(V, E).

Graph partitioning is a widely used technique in various areas, such as Computer Science, Mathematics, Engineering or Physics. Graph partitioning allows the development of applications in combinatorial optimization problems, data clustering[14], social network analysis, routing in communication networks, among others. One approach to graph partitioning is to use hypergraphs. For this purpose, numerous algorithms have been developed in Engineering, in various domains, such as, for example, the refinement of the finite element mesh and the optimization of computational resources (Sluzalec et al. 2022, Çatalyürek et al. 2023).

[14] An exploratory data analysis technique that aims to group similar objects or data points into distinct groups, called clusters.

The partitioning of a graph G (Figure 37), previously presented in the Figure 36, is carried out on the hyperedges and, taking into account, as far as possible, The connections between vertices contained in every hyperedge. In this example, each hyperedge E_i, corresponds to a subgraph G_i, implying the controlled loss of connections between the nodes that are contained in the resulting subgraphs and between the initial nodes of graph G. Such partitioning leads to the effective loss of the connection between (V_6 ; V_{11}). As such, this connection is not maintained in any resulting subgraph.

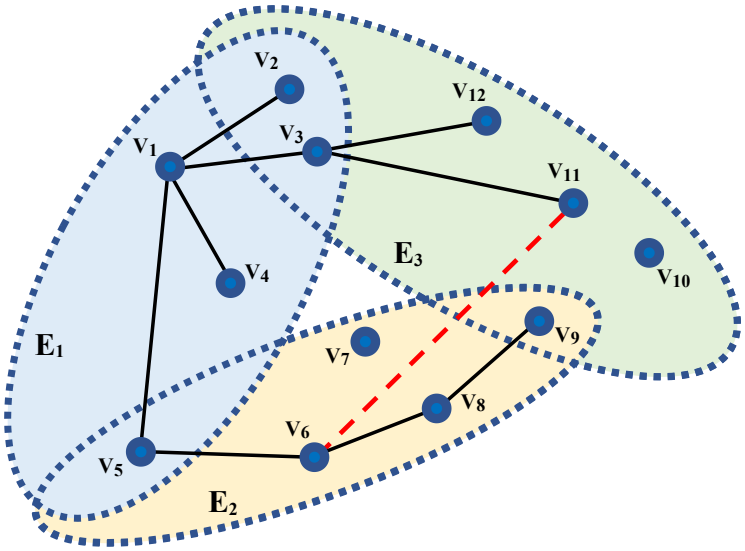


Figure 37 - Example of partitioning the graph G.

The situation modeled on the Figure 37, of the effective loss of connection between nodes in the partitioning process, is common, and constitutes one of the requirements to be achieved in graph partitioning: minimizing the number of edges connecting vertices of different subgraphs, i.e. the reduction of the number of crossed edges between subgraphs . Several authors have contributed significantly to the study of graph and hypergraph partitioning. There are different hypergraph partitioning techniques, each with its advantages and disadvantages, obviously. Among them, two-part partitioning, balanced partitioning, and spectral partitioning stand out. In a two-part partitioning, the goal is to find a division of the set of vertices V, into two subsets V (Sluzalec et al. 2022), and V_2, that minimizes the number of hyperedges that contain vertices of both subsets. This technique is simple but effective when used in cases where a division into just two parts is sufficient (Sluzalec et al. 2022) .

Balanced partitioning aims to divide the set of vertices V of the hypergraph into a fixed number of parts, usually k parts, of approximately equal size. This is useful when you want to achieve load balancing

in distributed systems, or when you need to break down a problem into sub-problems of similar size. The idea is also to minimize the number of hyperedges that cross different parts. The spectral partitioning technique, on the other hand, uses the analysis of the eigenvalues and eigenvectors of the hypergraph incidence matrix to perform the partitioning. In this technique, the goal is to find a division that minimizes the spectral cutoff ratio, which is defined as the ratio between the number of hyperedges that cross different parts and the number of hyperedges in each part (Chehreghani, et al. 2008, Cole and Zhu 2020).

3.2.2 Data classification through hypergraphs

Data classification by means of hypergraphs is a mathematically based technique, widely applied in the fields of machine learning and data analysis. This approach involves representing the data in hypergraph format, in which the data objects are represented as vertices and the relationships between these objects are represented by hyperedges. A key step in the classification of data by means of hypergraph is the proper construction of hypergraph from the input data. This can be done by assigning a hyperedge to each combination of data objects that have a relevant relationship. These relationships can be based on similarity[15], proximity,[16] or other measures relevant to the problem at hand. Careful choice of these hyperedges is critical to ensure proper representation of the data. The construction of hypergraphs plays an important role in subsequent tasks that are based on the structure of the hypergraph (Gao et al. 2022). Once the hypergraph is built, one can run hypergraph classification algorithms producing the final classification. This subtype of algorithms classifies hypergraph considering the hyperedges and relationships between them. From this, it is identified the need to design a way to assign labels to the vertices of the hypergraph, with the objective of making the classification in the best way. A common approach to doing hypergraph classification is label propagation. The principle of label propagation is to fix the similar labels of hypergraph. This method is formulated as an optimization problem, in which the labels to be propagated are obtained by minimizing the energy function, depending on the label of the vertices. Some optimization algorithms of Linear Programming or Quadratic Programming can be applied to solve the problem. These methods make it possible to obtain optimal or approximate solutions, depending on the characteristics of the data and the constraints involved. For more complex problems, approaches based on deep learning — especially convolutional neural networks and generative networks

[15] It is a measure that quantifies how similar or related two objects or data sets are.
[16] Spatial relationship between objects. The closer they are, the closer they are.

— stand out, which have been shown to be effective in the task of classification in hypergraph structures (Yang et al. 2025).

One of the main challenges in this domain is the selection of the appropriate hyperedges. That is, define the number of hyperedges and their size. It is important to mention that hypergraph classification algorithms are, by nature, computationally intensive. Despite this, hypergraph classification is considered one of the most viable and promising approaches in hypergraph classification, allowing us to detect patterns and form clusters in complicated datasets, representing complex relevant relationships between related entities.

3.3 Hypergraph Models

Throughout the exposition of the various models and forms of representation of hypergraphs, a brief contextualization of the respective fields of application will be included. This contextualization allows us to show that different types of hypergraphs require specific approaches, both in terms of representation and analysis. In this way, a solid basis is established for the transition to the next chapter, in which the particularities associated with hypergraphs will be deepened.

3.3.1 Uniform hypergraphs

Uniform hypergraphs are a common type of hypergraph, in which the hyperedges are the same size. Second, a uniform hypergraph is a hypergraph such that V is a finite set and E is a set of subsets of V, all the same size, i.e., |e| = k. Thus, if |and| = k for all and ∈ E, so H is a uniform hypergraph k-uniform, which is often used in applications in which all hyperedges represent a group of k elements, as formally defined by Berge (1989). In computational complexity theory, k-uniform hypergraphs are often used in optimization problems in which each solution is a subset of size k of a larger set of elements. An example of this is the partitioning problem, which seeks to find a maximum independent set in a graph that can be formulated as an optimization problem, whose solutions correspond to a subset of size k of its vertices.

3.3.2 Bi-split hypergraphs

The use of bipartite hypergraphs is a common solution, which has been studied in several areas, including recommendation systems, data extraction, and social network analysis. Bipartite hypergraphs have long been a popular topic due to their ability to model complex relationships between two different classes of

objects. The nature of these hypergraphs allows for the clear and concise representation of user requests and item preferences in recommendation systems. This type of approach is especially useful for large volumes of data, where it can be difficult for simple recommendations, as data overload makes it increasingly easier to provide expert recommendations based on user interest and story.

A recent study introduced the concept of hypergraph in recommendation systems, according to the information from the historical evaluation. In that study, the authors explored the power of bipartite hypergraphs in representing user preferences and items that showed the effectiveness of these hypergraphs in improving the quality of recommendations. Furthermore, as mentioned in, "recommendation systems emerge in digital commerce as important promotion tools to help customers find objects of interest." Therefore, hypergraphs can be useful in different contexts of recommender systems (Wang et al. 2022, Mao et al. 2019). Through the application of techniques related to graph analysis, such as Graph Convolutional Networks, it is possible to exploit information from bipartite hypergraphs to generate accurate and personalized recommendations. This is particularly relevant in a society governed by the prevalence of data, where personalization is an expectation of users. A bipartite hypergraph is thus a generalization of a bipartite graph in which a set of edges can connect more than two vertices. Then, it is explored that: bipartite hypergraphs have an extension that is called k-partite hypergraphs (Wu et al. 2021).

3.3.3 K-Split Hypergraphs

K-party hypergraphs are a generalization of bipartite hypergraphs, in which the set of vertices is divided into k disjoint sets, ensuring that each hypergraph contains at least one vertex from each set. This class of hypergraphs have been used in studies of partitioning models and methods, in which they have been successfully used for different purposes in a wide range of areas, such as parallel scientific computing (Oguz Selvitopi, Turk and Aykanat, 2012), layout design of large-scale integration circuits, parallel information recovery, rendering of parallel volumes, and SBD. proposed a K-Partite hypergraph-based approach to road network expansion planning in a growing urban area. With this model, it was possible to identify a set of routes for the construction of new roads, considering the interconnections between existing streets and points of interest in the region. This approach has demonstrated its effectiveness in improving the understanding of the complex interactions between project components, allowing the identification of constraints (Demir and Aykanat, 2010).

The representation of complex systems by means of hypergraphs has allowed advances in the modeling of real-world problems. In this approach, distinct vertices symbolize specific components, while hyperedges denote the multiple interactions between them. Such a model becomes particularly useful when it makes explicit relations that would not otherwise be explored. The search for interval hypergraphs, for example, allows the incorporation of notions of temporality and uncertainty, aspects that are often central to dynamic processes. The analysis from such tools may bring new perceptions about plural systems and their complex interdependencies.

3.3.4 Interval Hypergraphs

Interval hypergraphs are also of great importance in several areas of computing, such as in the modeling of highly complex systems and in the detailed analysis of large data sets. Interval hypergraphs extend traditional hypergraphs even further, since each hypergraph is represented by an interval on a real line. Formally, an interval hypergraph is defined as a pair (V, I), where V is a set of vertices and I is a set of real intervals, such that each hypergraph is a subset of V that intersects each interval at I. The representation of an interval hypergraph can be constructed through a correspondence matrix, which is a binary matrix in which each row represents a vertex and each column represents a time interval. If a vertex i is contained in the hyperedge j, then the element (i,j) of the matrix is equal to 1. Otherwise, it equals 0. This type of graph, with emphasis on temporal intervals, becomes important to analyze dependencies in the context of complex projects and dynamic systems, being a key structure to model various relationships, as observed by Let's see, in the context of civil construction, tasks constitute the elementary units of work or events within the analyzed system. Each task has a unique name or code that distinguishes it. In addition, the work is linked to a period delimited by a beginning and end. This characterization is essential to understand the temporal sequence of activities and how they relate over time. In graphs of this type, vertices represent tasks while hyperedges describe precedence between them. These connections can join more than two vertices, allowing you to model complex dependencies. Functional dependencies establish, in this context, how an activity depends on the completion of other activities. These dependencies can be classified into different types, including (Bulgaru and Voloshin 1997, Ribas 2015):

- Start-Finish (IT) dependency, which indicates that an activity (A_1) can only start after the completion of another activity (A_2);
- Start-Start Dependency (II), which requires activity (A_1) to start simultaneously with another activity (A_2);
- Finish-Start (TI) dependency, which determines that an activity (A_1) must finish before the start of another activity (A_2);
- Finish-Finish Dependency (TT), which states that an activity (A_1) must finish at the same time as another activity (A_2).

Interval hypergraph analysis allows the identification of critical paths, which consist of sequences of activities that define the minimum duration of the project. Recognizing these paths is also vitally important for project planning and management, as it directs resources efficiently and ensures that the project is completed on time. Interval hypergraph analysis and optimization can benefit from the use of specialized algorithms, such as CPM, which plays a key role in determining the start and end times of each activity, as well as identifying those activities that are critical to the success of the project. Following, the example of the management and control of the construction of a 2-storey building, with a structure in Reinforced Concrete, in which some values of time intervals for the execution of activities are supported. In this example, we're managing the build with a set of build activities and their corresponding execution time intervals. Let us characterize the case under analysis:

Set of Vertices (V):

V_1: "Foundation Excavation" - Break: 01/01/2023 to 15/01/2023;

V_2: "First Floor Concreting" - Break: 16/01/2023 to 31/01/2023;

V_3: "Building Walls" - Range: 01/02/2023 to 15/02/2023;

V_4: "Second Floor Concreting" - Break: 16/02/2023 to 28/02/2023;

V_5: "Installation of Pipes" - Range: 01/03/2023 to 15/03/2023;

V_6: "Roof Structure" - Range: 16/03/2023 to 31/03/2023;

V_7: "Electrical Installation" - Range: 01/04/2023 to 15/04/2023;

V_8: "Interior Finishes" - Range: 16/04/2023 to 30/04/2023;

V_9: "Exterior Painting" - Interval: 01/05/2023 to 15/05/2023;

V_{10}: "Final Cleanings" - Interval: 16/05/2023 to 31/05/2023;

V_{11}: "Final Inspection" - Range: 01/06/2023 to 15/06/2023.

Set of Intervals (I):

I_1: 01/01/2023 to 15/01/2023;

I_2: 16/01/2023 to 31/01/2023;

I_3: 01/02/2023 to 15/02/2023;

I_4: 16/02/2023 to 28/02/2023;

I_5: 01/03/2023 to 15/03/2023;

I_6: 16/03/2023 to 31/03/2023;

I_7: 01/04/2023 to 15/04/2023;

I_8: 16/04/2023 to 30/04/2023;

I_9: 01/05/2023 a 15/05/2023;

I_{10}: 16/05/2023 a 31/05/2023;

I_{11}: 01/06/2023 a 15/06/2023;

The binary incidence matrix of the interval hypergraphs of the example (Table 4) records the relationships between activities and time intervals for construction.

Table 4 - Binary incidence matrix of the interval hypergraph.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}
Foundation Excavation	1	0	0	0	0	0	0	0	0	0	0
First Floor Concreting	1	1	0	0	0	0	0	0	0	0	0
Wall Construction	0	1	1	0	0	0	0	0	0	0	0
Concrete of the Second Floor	0	0	1	1	0	0	0	0	0	0	0
Plumbing Installation	0	0	0	1	1	0	0	1	0	0	0
Roof Structure	0	0	0	0	1	1	0	0	0	0	0
Wiring	0	0	0	0	0	1	1	1	0	0	0
Interior Finishes	0	0	0	0	0	0	1	1	0	0	0
Exterior Paint	0	0	0	0	0	0	0	1	1	1	0
Final Cleanups	0	0	0	0	0	0	0	0	1	1	1
Final Inspection	0	0	0	1	0	0	0	0	0	1	1

The project management team can track in more detail the dependencies between the construction stages and ensure that the two-story structure is built within the deadlines established for each stage. This dependency analysis can show you how each activity is linked to previous and next events. This allows the project management team to clearly identify critical dependencies and plan the workflow more efficiently, ensuring that the project is completed according to the established schedule. Let us now make a brief analysis of the dependencies between the activities (vertices):

Foundation Excavation (V_1):

It has no functional dependencies.

Must occur during Interval 1 (01/01/2023 to 01/15/2023).

First Floor Concreting (V_2):

Functional Dependencies: Dependent on the completion of "Foundation Excavation."

Must occur during Interval 2 (16/01/2023 to 31/01/2023).

Wall Construction (V_3):

Functional dependencies: Dependent on the completion of the "First Floor Concreting."

Must occur during Interval 3 (01/02/2023 to 15/02/2023).

Concreting of the second floor (V_4):

Functional Dependencies: Dependent on the completion of the "First Floor Concreting" and "Wall Construction."

Must occur during Interval 4 (02/16/2023 to 02/28/2023).

Plumbing Installation (V_5):

Functional dependencies: Dependent on the completion of the "Second Floor Concreting."

Must occur during Interval 5 (03/01/2023 to 03/15/2023).

Roof Structure (V_6):

Functional dependencies: Dependent on the completion of the "Plumbing Installation."

Must occur during Interval 6 (03/16/2023 to 03/31/2023).

Electrical Installation (V_7):

Functional Dependencies: Dependent on the completion of the "Roof Structure."

Must occur during Interval 7 (01/04/2023 to 15/04/2023).

Interior Finishes (V_8):

Functional dependencies: Dependent on the completion of the "Electrical Installation."

Must occur during Interval 8 (04/16/2023 to 04/30/2023).

Exterior Paint (V_9):

Functional Dependencies: Dependent on the completion of the "Interior Finishes."

Must occur during Interval 9 (05/01/2023 to 05/15/2023).

Final Cleanups (V_{10}):

Functional Dependencies: Dependent on the completion of "Exterior Painting."

Must occur during Interval 10 (05/16/2023 to 05/31/2023).

Final Inspection (V_{11}):

Functional Dependencies: Dependent on the completion of the "Second Floor Concreting" and the "Final Cleanings."

Must occur during Interval 11 (01/06/2023 to 15/06/2023).

In this case, hyperedges are used to model the dependencies between construction activities, being associated with the relevant intervals. For example, the "Second Floor Concreting" activity depends on the completion of the "Wall Construction" and "First Floor Concreting" activities. Let us complement the analysis with the following:

- H_1: "Foundation Excavation" - > "First Floor Concreting" (Dependence from the end of activity 1 to the start of activity 2) - Break: 01/01/2023 to 31/01/2023;
- H_2: "First Floor Concreting" - > "Wall Construction" (Dependence from the end of activity 2 to the beginning of activity 3) - Break: 16/01/2023 to 15/02/2023;
- H_3: "First Floor Concreting" - > "Second Floor Concreting" (Dependence on the end of activity 2 for the beginning of activity 4) - Break: 16/01/2023 to 28/02/2023;
- H_4: "Concreting of the Second Floor" - > "Installation of Pipes" (Dependence from the end of activity 4 to the beginning of activity 5) - Break: 16/02/2023 to 15/03/2023;
- H_5: "Plumbing Installation" - > "Roof Structure" (Dependence from the end of activity 5 to the start of activity 6) - Range: 01/03/2023 to 31/03/2023;
- H_6: "Roof Structure" - > "Electrical Installation" (Dependence from the end of activity 6 to the start of activity 7) - Break: 16/03/2023 to 15/04/2023;
- H_7: "Electrical Installation" - > "Internal Finishes" (Dependence on the end of activity 7 for the start of activity 8) - Break: 01/04/2023 to 30/04/2023;

- H_8: "Interior Finishes" - > "Exterior Painting" (Dependence on the end of activity 8 for the start of activity 9) - Break: 16/04/2023 to 15/05/2023;
- H_9: "Exterior Painting" - > "Final Cleanings" (Dependence on the end of activity 9 for the start of activity 10) - Break: 01/05/2023 to 31/05/2023;
- H_{10}: "Second Floor Concreting" - > "Final Inspection" (Dependence from the end of activity 4 to the start of activity 11) - Break: 01/06/2023 to 15/06/2023.

This mapping intuitively shows how the interval hypergraph helps to visualize not only the activities and their execution intervals, but also the dependency relationships that organize the progress of the project. With this perspective, it becomes simple to identify critical moments of coordination and manage resources optimally, increasing predictability and efficiency throughout the construction.

The WBS,[17] widely used in project management literature (Project Management Institute, (2017), delineates the project in terms of its constituent parts, providing management with a clear understanding of the relationships and interconnections between activities. As illustrated in Figure 38, each level of the WBS details specific project deliverables, making it easy to assign responsibilities, estimate required resources, and track progress.

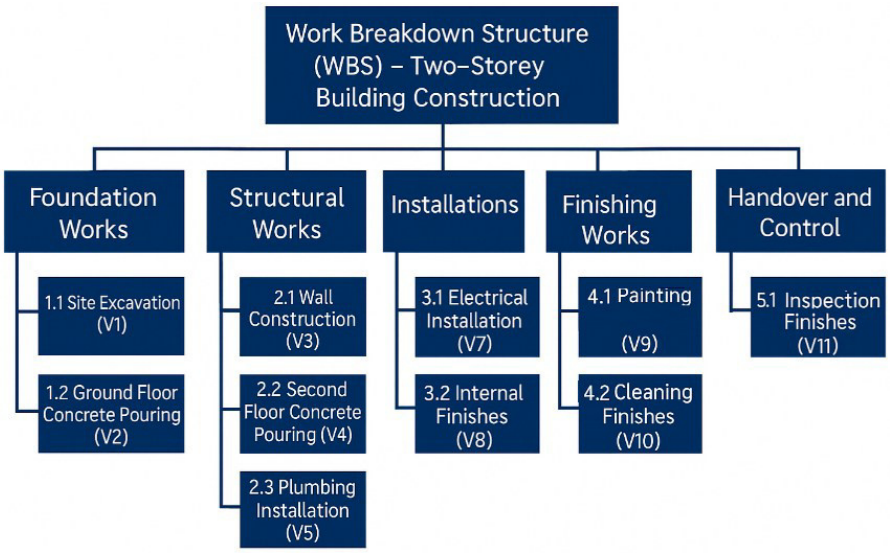


Figure 38 - A hierarchical WBS for planning a two-storey building construction project.

[17] The Work Breakdown Structure is a cornerstone in project management, playing a vital role in disaggregating and organizing the activities required to achieve project objectives. It provides a hierarchical and systematic representation of all tasks, deliverables, and project components, simplifying planning, execution, and control.

In the context of the interval hypergraph presented in Table 4, WBS plays a key role in the hierarchical representation of activities and the identification of their functional dependencies. This assists the management team in understanding the temporal dynamics of the project, ensuring that activities are executed efficiently and that the project is completed within the stipulated deadlines. Thus, WBS proves to be a valuable tool in managing complex projects, as exemplified, where time dependencies play a critical role in the overall success of the project.

To determine the construction project deadline, it is essential to follow a well-structured methodology. First, it is necessary to identify all the activities involved in the project and their functional dependencies. Next, you need to create a network diagram, a precedence network diagram, to visually represent the activities and their relationships. Subsequently, the earliest start and end dates will be calculated, as well as the latest start and end dates for each activity, considering the dependencies and in this sequence the critical path is required. Finally, you must ensure that the time interval between the latest start date of the last activity on the critical path and the earliest start date of the first activity does not exceed the desired deadline for the project. This will ensure that the project is completed within the specified timeframe. The use of project management software (for example, *MS Project*) can significantly simplify this process, helping in the analysis and monitoring of dates.

In summary, the interval hypergraph appears as a tool to model and analyze complex temporal dependencies between activities in projects and systems. Its detailed analytical structure, combined with the identification of functional dependencies and critical paths, plays an essential role in project planning and management, ensuring the efficient execution of activities within the stipulated deadlines. The inclusion of CPM further expands the ability to manage projects efficiently, ensuring their completion (Odedairo 2024) .

3.3.5 Weighted hypergraphs

Weighted hypergraphs allow for a more accurate representation of interconnections in complex networks, in which each link between elements has an associated weight according to its importance or frequency. This flexible framework can be applied in a variety of domains, such as modeling scientific relationships between researchers or mapping a city's public transportation system. According to this, weighted hypergraphs have been useful for exploring properties in complex systems by capturing subtleties in the connections between components. More recently, they created an algorithm capable of grouping

elements in a weighted hypergraph, considering multiple perspectives and using the weights as a guide to improve the accuracy of the clusters formed. Tests indicated that the method outperformed previous approaches. They also used this model to identify communities in social networks, proposing a strategy based on weighted hypergraphs and evaluating their effectiveness in varied data sets. Their analyses point to the potential for improved understanding of these complex networks, as reported in recent studies (Landry and Restrepo, 2022; Liu et al., 2021; Zhen and Wang, 2021).

Weighted hypergraph analysis has promising applications in natural language processing. Some researchers have explored this approach to map topics in texts, considering the relative importance of each term. The proposed model represents topics as groups of words related through a hypergraph, in which each lexeme is a node and each document a weighted edge. Initially, the framework is built by connecting shared terms between documents. Subsequently, weighting techniques such as TF-IDF[18] analyze the frequency of words to determine their relevance in contexts. This semantic mapping is important for the automatic detection of latent themes, carried out via algorithms on the hypergraph. By highlighting sets of keywords, these methods reveal the underlying subjects of the texts. To make the results more noticeable, they can be visualized graphically, evidencing the semantic links between lexemes and corpora. In addition, the detection of feelings can be coupled with analysis, attributing affective polarities to topics and words. This capture of the emotional aspect deepens the understanding of the content. In short, the mapping of lexical networks through hypergraphs is a deep and structured strategy to explore written documents. By mapping the relationships between terms, it enables the automated extraction of latent meanings in natural language communications.

3.3.6 Targeted hypergraphs

According to the pioneering definitions Giorgio Ausiello and Laura (2017) a directed hypergraph, H = (V,E) is a double, where V is a set of vertices and E is a set of hyperarcs, a hyper-arc EClique ou toque aqui para introduzir texto.i € E it's a pair (X,Y), where X and Y are disjoint subsets of V and in case X and Y are unit sets (x,y), then we're looking at a simple digraph. For a better formal understanding of this abstract structure, in the Figure 39 An example of a directed hypergraph is presented, H = (V,E), where V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and E = {E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8}.

[18] A technique that evaluates the importance of words in documents. It calculates a value based on the word frequency in the document (TF) and the rarity of the word in the entire document collection (IDF). This helps to highlight meaningful keywords in text analytics.

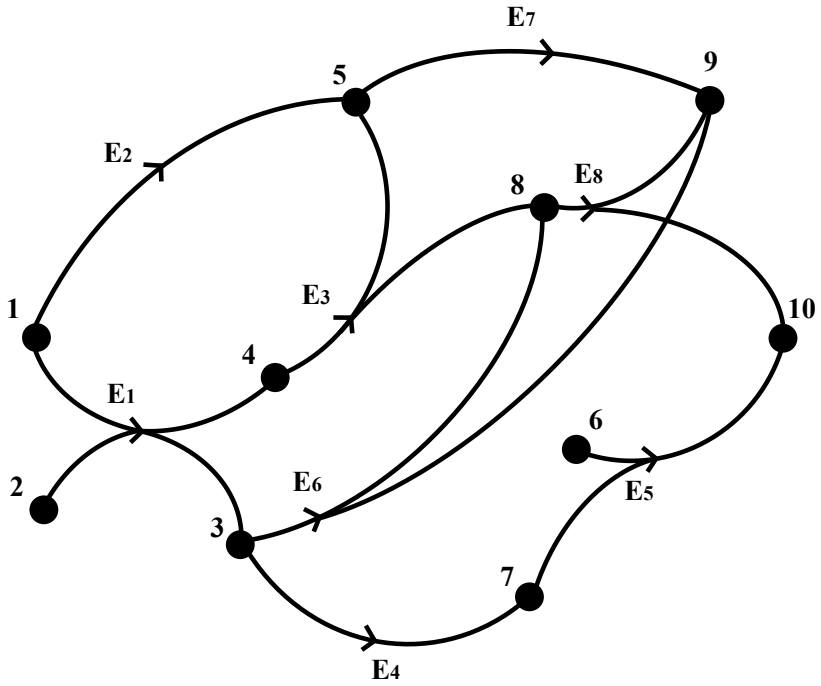


Figure 39 – An example of a directed hypergraph H = (V,E).

Where E: E_1 = ({1, 2}, {3, 4}), E_2 = ({1}, {5}), E_3 = ({4}, {5, 8}), E_4 = ({3}, {7}), E_5 = ({6, 7}, {10}), E_6 = ({3}, {8, 9}), E_7 = ({5}, {9}), E_8 = ({8}, {9, 10}), are the hyper-arcs defined in accordance with the incidence matrix in the terms proposed by the same author presented in Figure 40.

	E1	E2	E3	E4	E5	E6	E7	E8
V1	-1	-1	0	0	0	0	0	0
V2	-1	0	0	0	0	0	0	0
V3	1	0	0	-1	0	-1	0	0
V4	1	0	-1	0	0	0	0	0
V5	0	1	1	0	0	0	-1	0
V6	0	0	0	0	-1	0	0	0
V7	0	0	0	1	-1	0	0	0
V8	0	0	1	0	0	1	0	-1
V9	0	0	0	0	0	1	1	1
V10	0	0	0	0	1	0	0	1

Figure 40 – An incidence matrix of a directed hypergraph.

Let (|V|) denote the number of vertices, denoted by (n), and let (|E|) denote the number of hyperarcs, denoted by (m). The size of a hyperarc (e in E) is given by (|e| = |Xe| + |Ye|), and the total size of the directed hypergraph is defined as the sum of the sizes of all its hyperarcs. According to the

formal definitions of directed hypergraphs, two classes are distinguished: B-hypergraphs and F-hypergraphs (Gallo et al. 1993), where the prefixes “B” and “F” correspond to the terms backward and forward, respectively. A BF-hypergraph is defined as a combination of B-hypergraphs and F-hypergraphs, in which each hyperarc is associated with both a non-negative real value and a Boolean value indicating its presence or absence.

This framework is used in optimization problems that involve multiple objectives. The presence or absence of these hyperedges is determined by a set of Boolean constraints. In the following sections, the two classes will be analyzed: B-hypergraph and F-hypergraph, highlighting the formal aspect that distinguishes them, that is, the degree of the exit and input hyperarcs. This is also implied in the degree of input and output of each vertex, since the essence of the relationships between the vertices entities is linked to the connections between these vertices.

3.3.7 B-hypergraph hypergraphs

B-hypergraphs are useful structures for solving combinatorial optimization problems, in which the objective is to find a solution that satisfies a set of constraints, for example the planning of works. They are a type of hypergraph in which each hypergraph is assigned a Boolean value (0 or 1), indicating whether it is present or not. This structure (Figure 41) reveals the formal particularity that, for each hyper-arc "Ei" or total destination vertices is a.

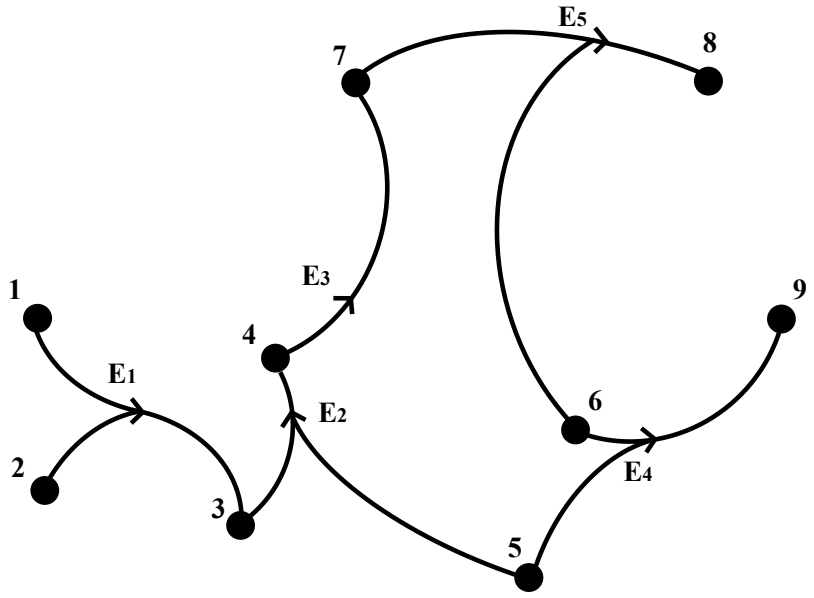


Figure 41 – An example of a B-hypergraph hypergraph.

Early investigations that explored hypergraph extensions for classification tasks included the seminal work of (Gallo et al. 1993), when they developed a hypergraph approach to assessing the importance of attributes in binary classification problems. In this method, hyperedges represented subsets of attributes or instances, to capture complex Boolean relationships that could not be easily mapped in traditional graphs. Experimental results demonstrated significant gains in accuracy when considering advanced combinatorial weights and structures. More recently, they have developed a contrastive algorithm for structural hypergraph learning, applied to the classification of nodes in complex networks. In this promising approach, the hypergraph structure is inferred iteratively, and the weights of the hyperedges guide the aggregation and formation of more coherent clusters, outperforming models based on simple graphs (Misiorek and Janowski 2023, Cai et al. 2022).

In the field of semantic reasoning, the proposal of the hypergraph transformer stands out, a deep learning model capable of incorporating hyperedges as semantic units for complex multi-hop inference tasks, such as Visual Question Answering[19] (Pig et al. 2022). The results obtained indicate that the use of hypergraph-based representations favors a more effective organization of contextual dependencies, in addition to optimizing the distribution of information in large-scale scenarios. Such an approach is particularly promising when it establishes parallels with the challenges of combinatorial optimization.

These examples highlight the usefulness of hypergraphs — including their Boolean and weighted variants — in different domains of Computer Science, from data classification to advanced multimodal reasoning. Like the MLD — "holocode" (Chapter 2) — B-hypergraphs offer operational tools for modeling complex systems and intricate hierarchical relationships. Each of these structures is designed to support highly complex representations. B-hypergraphs stand out for enabling higher-order relationships between entities and customizable data space typing's, while DMN focuses on modeling hierarchical systems with multiple layers of abstraction.

3.3.8 F-hypergraph hypergraphs

F-hypergraphs are also an extension of conventional graphs and hypergraphs, in which each hyperedge is assigned a real numerical value, representing the importance or cost associated with that hyperedge. This is also useful in optimization problems that involve minimizing or maximizing an objective function,

[19] Artificial intelligence task that combines computer vision and natural language processing.

subject to constraints expressed by hypergraphs. A paradigmatic example is the model proposed in, which exploits dynamic hypergraphs structures to represent large-scale urban transport systems. In this approach, hypergraphs are continuously updated to reflect non-binary correlations between different segments and traffic modes, assigning relative weights to the connections according to the intensity and temporal evolution of the flows. This modeling makes it possible to allocate and predict demands in a more robust and integrated way, allowing the efficient management of complex multimodal networks and demonstrating the potential of hypergraphs for urban infrastructure planning. Take the example of (Zhao et al. 2023) Figure 42. The F-hypergraphs denote as a formal particularity, the fact that for each hyperarc "E$_i$" the total of source vertices is one.

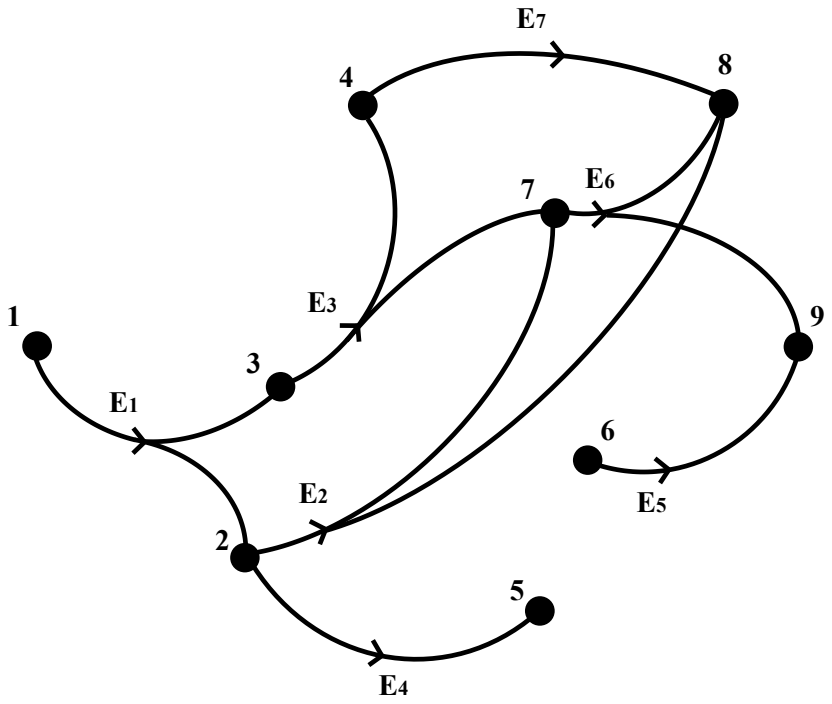


Figure 42 – A F-hypergraph.

Directed hypergraphs, B-hypergraphs and F-hypergraphs constitute a substantial set of applied mathematical structures, establishing a propitious scenario for a dynamic expansion: flow hypergraphs. While these interconnected mathematical abstractions provide tools for optimization and modeling, flow hypergraphs introduce an additional dimension of dynamism by allowing the representation of complex flows between multiple entities simultaneously. This synergy between the different facets of hypergraphs offers an integrated framework to address a variety of complex problems, finding applications from engineering to complex network analysis.

3.3.9 Stream Hypergraphs

Stream hypergraphs have several advantages over conventional flow graphs. They can model flows that involve more than two entities simultaneously, which allows for a representation of complex systems that involve multiple sources and sinks [20] (Cambini, Gallo, and Scutellà 1997). Flow hypergraphs are more flexible in modeling capacity constraints in hyperedges, which allows for a closer representation of real systems. There are also several algorithms to solve the problem of maximum flow in flow hypergraphs. The most common algorithm is the adapted Ford–Fulkerson algorithm, which is a generalization of this algorithm to conventional flow graphs. The Edmonds–Karp algorithm is a variation of the Ford–Fulkerson algorithm that uses the shortest path in the residual network to increase throughput. A flexible model to implement the adapted Ford-Fulkerson algorithm in hypergraph was explored where the flow structure of the algorithm is compared between the model to the graph (FG) and the structure of the hypergraph (HG) (Figure 43).

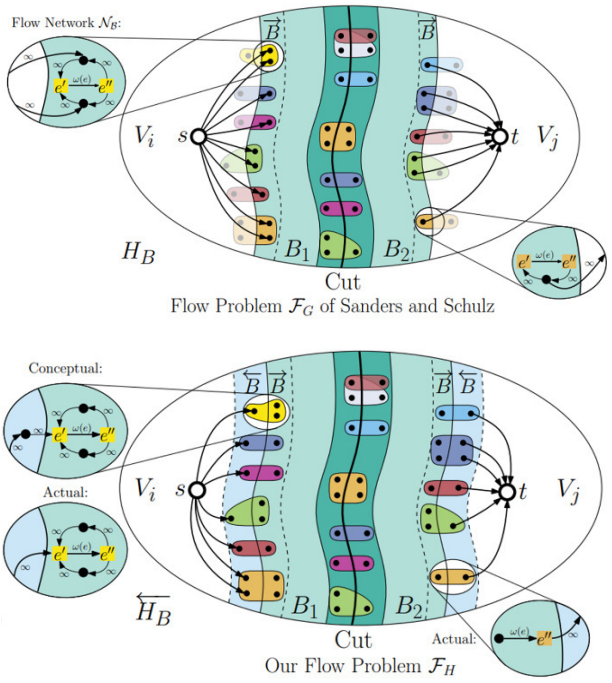


Figure 43 - Comparison between the classical flow formulation in graphs and the proposed flow formulation in hypergraphs, illustrating the cut structure and flow representation (adapted from Cambini, Gallo and Scutellà 1997).

[20] "Source" - Initial vertex of an arc; "Sink" - End vertex of an arc

These algorithms aim to solve the problem of maximum flow in flow hypergraphs, which is an essential task in combinatorial optimization and network analysis. It refers to determining the maximum amount of flow that can be transmitted from a source to a sinkhole at the edges (or hyperedges, as far as it is in hypergraphs) that make up the network. Specifically, these algorithms try to arrive at the optimal flow distribution that involves the maximum amount of flow and fits the capabilities of the edges while preserving the flow balance at all vertices. Flow hypergraphs have a significant extension to traditional graphs, presenting unique advantages, namely, in system representations when modeling the flow that involves several entities simultaneously. Reinforcing that, an important characteristic of flow hypergraphs is their flexibility in modeling capacity constraints in hyperedges. This adaptability makes them particularly well suited to represent real-world systems, where pathway capabilities can vary considerably, just as they do in B-Hypergraphs and F-Hypergraphs. While flow hypergraphs extend our capabilities to model complex systems and multidirectional flows, color hypergraphs provide us with a tool to represent and allocate resources.

Continuing our exploration into hypergraph theory, let's now pay attention to colored hypergraph, which plays a key role in resource allocation in various domains, including industrial production and even construction. We will investigate how this advanced approach can enhance our understanding of the complex relationships between elements and features in diverse systems.

3.3.10 Colored hypergraph

Colored Hypergraphs (HC) can be used for allocation problems, to represent in more detail the complex relationships between elements. With relative ease, this type of model solves the problem of resource allocation, for example, by allocating the communication channel and identifying the node by the occurrence of interference in user communications (U) and device positions (D). Resource capacity is the skill required to prevent any occurrence of interference in communication. In the modeling presented in (H. Zhang et al. 2016).

Figure 44, the communication channel resource is a node, and the hyperedge, the possibility of using the communication channel, is associated with one or more colors to indicate the skills required to perform the work. In other words, hyper-edge is an option in allocating resources to activities that have another suitable skill.

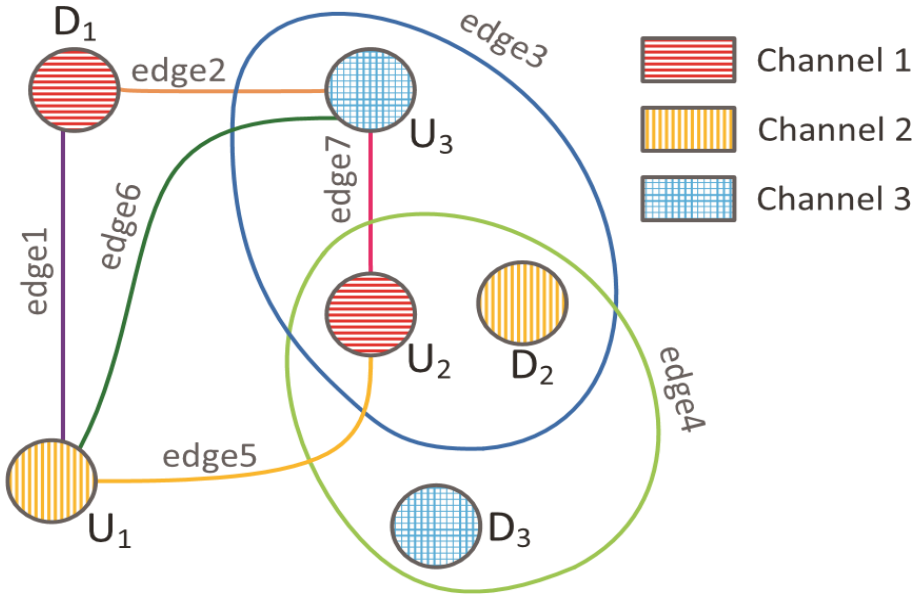


Figure 44 – A colored hypergraph model – adapted from (H. Zhang et al. 2016).

Therefore, HC resource modeling fulfills the more technical approach of considering the dynamic relationships between resources and activities in a building. Initially, the modeling is initiated by identifying all the resources involved, including labor, equipment, material, machinery, subcontractors, among others. The next step is the precise definition of the activities necessary to complete the project, which involves careful consideration of the characteristics of each activity and the resources with the correct competence to carry out each activity, such as the execution of brick masonry and electrical installations, these competencies being represented by different colors. With this detailed information, the color hypergraph to be built shows features such as nodes and activities linked to one or more colors, indicating the skills it needs. Edges on resources are activities that they can be assigned to. For resource allocation, hyperedges are used and set as resource allocation targets for activities where their color is satisfied. The proper construction of the HC is very important for the optimization of resource allocation. Resource allocation is intended to make the best possible use of a resource, by considering each resource's capacity, available time, and time constraints, for greater project efficiency. Once the HC is built, it is possible to perform an optimized analysis to improve resource allocation. Designed with the color hypergraph model, managers can make informed decisions about resource allocation, such as scheduling activities, allocating resources, or organizing specific resources and identifying constraints and conflicts to be resolved. In addition to resource modeling by color hypergraphs, many construction professionals generally employ common methodologies for the planning and management of their projects. The use of CPM networks, a method commonly used in project management, is an example. Microsoft Project uses

the CPM type to examine and trace the critical path of a project. CPM is therefore a method that has been previously used by managers interested in tracking the progress of their project. By combining color hypergraphs with critical path analysis in CPM, managers can create a unified view for resource allocation and graph scheduling. Let's look at a practical example, relating to a small construction project, with the following activities:

- A_1 - Preparation of the Ground. This activity involves preparing the land for construction, which. requires specific skills related to land preparation (yellow color);
- A_2 – Foundation. The next step is the construction of the foundation, which demands foundation skills (red color);
- A_3 – Structure. To build the structure, a skill related to Formwork Equipment (orange color) is required. Remembering that, we have a specific resource for this task;
- A_4 – Masonry. The masonry activity requires skills in Masonry (blue color) and materials, such as bricks;
- A_5 - Electrical Installations. The next step involves electrical installation, requiring skills in Electrical Installations (green color);
- A_6 – Finishes. Finally, we have the finishing phase, which requires skills in Electrical Installations (green color) and materials, such as paint.

Now, according to the schema of Figure 45, let's consider the resources available for these activities:

- $Labor_1$ ('João'). João has skills in Masonry (blue color) and Electrical Installations (green color);
- $Labor_2$ ('Carlos'). Carlos has skills in Land Preparation (yellow color) and Foundation (red color);
- $Material_1$ ('Bricks'). The bricks will be used in the masonry activity;
- $Material_2$ ('Steel'). Steel will be needed for the construction of the structure;
- $Material_3$ ('Paint'). The paint that will be used in the finishes;
- $Equipment_1$ ('André'). André has an orange skill (Equipment - Formwork) for the structure activity.

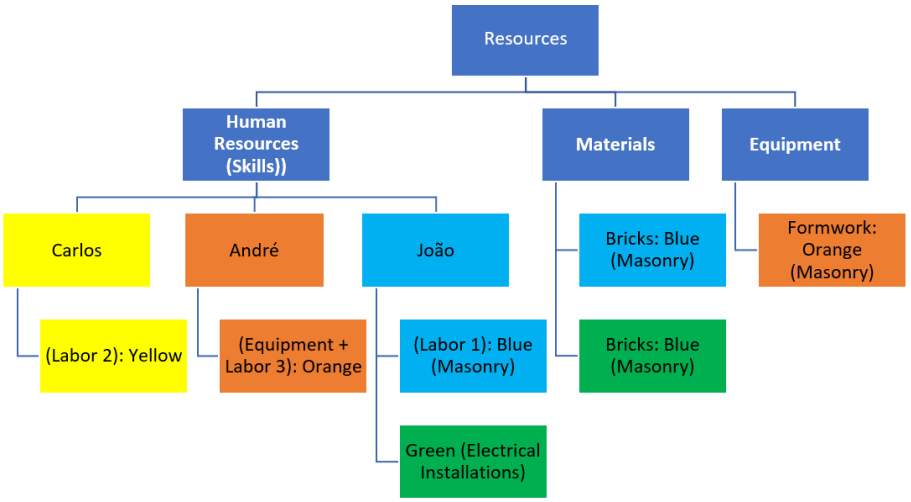


Figure 45 - Color-based representation of resources and associated competencies in the coloured hypergraph model.

After CPM analysis, a Gantt graph is established[21] (Figure 46), which allows to determine which are the critical activities (those that affect the total duration of the project), which are "Land Preparation," "Foundation," "Steel Structure," and "Electrical Installations," and the non-critical activities "Masonry" and "Finishing."

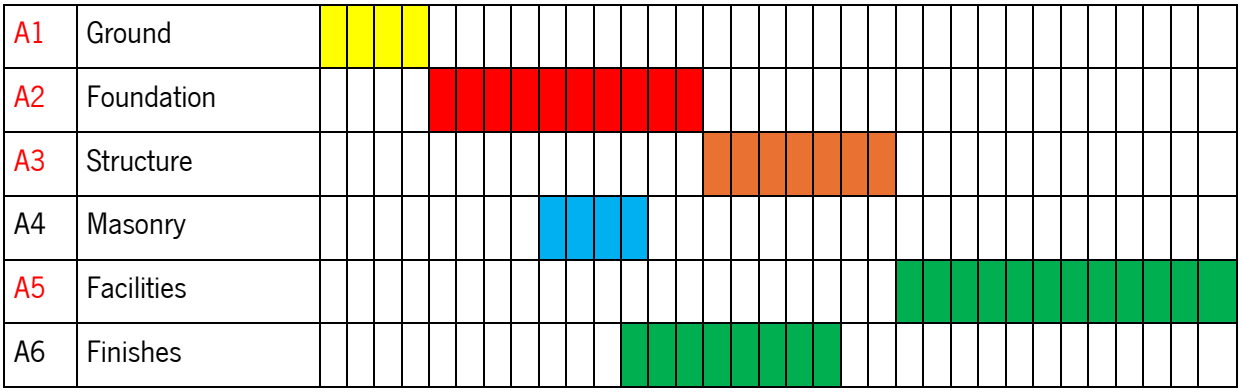


Figure 46 - Gantt chart of a construction project.

As presented in Figure 47, we allocated Carlos, who has skills in "Land Preparation" and "Foundation," to these critical activities, ensuring that they would be completed on time. For the "Structure" activity, we assigned Andrew, who has the orange skill and the necessary equipment to complete this critical activity. Finally, the activity "Electrical Installations," which is also part of the critical path, is carried out by John due to his green skills.

[21] A type of chart that shows a project's activities over time.

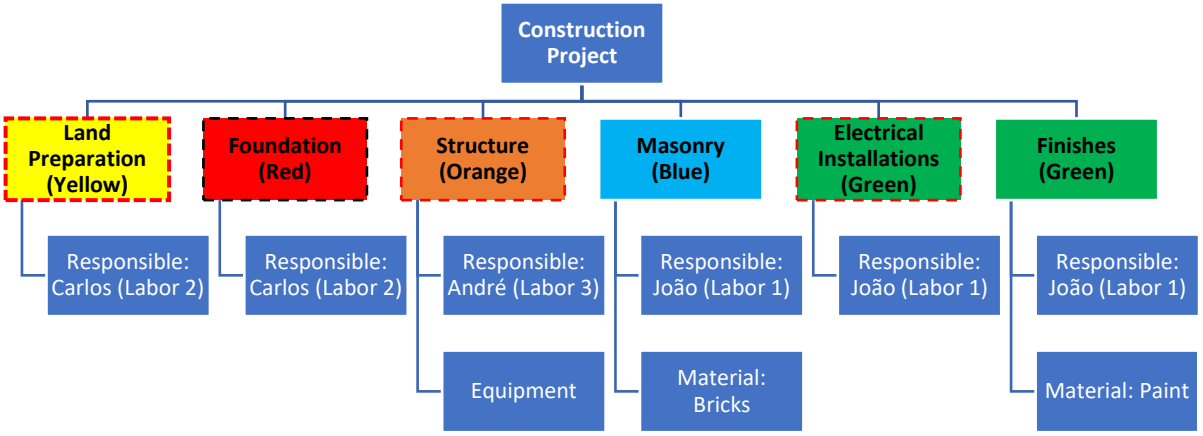


Figure 47 - Resource allocation tree.

The "Masonry" and "Finishing" activities are not on the critical path and can therefore be carried out later, according to the availability of resources. This example demonstrates how CPM is used to determine critical activities in a project and how resources are efficiently allocated according to this critical path, ensuring that the project is completed on time. The combination of the temporal view of the CPM with the relational and multicolored perspective of the HC allows a more useful framework for the integrated management of time and resources, contributing to greater efficiency and predictability in the execution of the project (Figure 48), in which the Color Hypergraph visually and unequivocally illustrates the relationship between skills, resources and activities, as well as their chromatic correspondence.

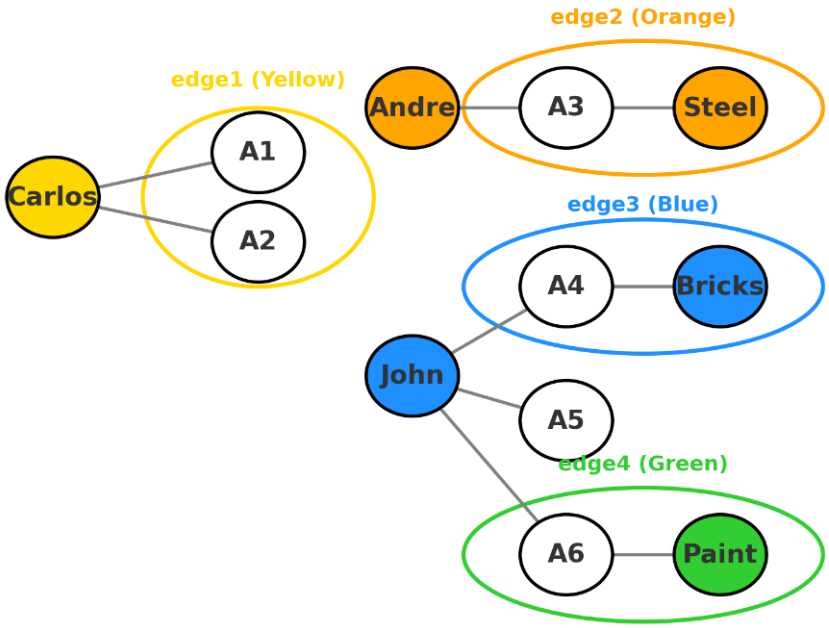


Figure 48 - Colored hypergraph of civil construction.

Modelling with color hypergraphs in construction projects can follow a process, which includes operations such as:

- Identification of essential resources;
- Detailed definition of project activities;
- Associating skills with resources through colors;
- Creation of the color hypergraph representing resources and activities;
- Establishment of hyper-edges for skill-based resource allocation;
- Analysis and optimization of resource allocation;
- Informed decision-making for resource management in the project.

The application of HC in project management, exemplified in the civil construction scenario presented, represents a facilitating approach to face the intricate interaction between resources and activities. Associating specific colors with skills and features provides an accurate and detailed visual representation, facilitating fine-grained modeling of design elements. The use of these color hypergraphs, in conjunction with traditional methodologies such as CPM, is like an integrative approach, which provides managers with a comprehensive view to make informed decisions, ensuring more effective and cost-effective management in complex projects.

3.3.11 Reducible hypergraph

The concept of reducibility in *flowgraphs*[22] refers to the ability to simplify and manipulate the flow of control in computer programs, making it more understandable and efficient. This in-depth analysis involves a formal definition of reducibility in terms of specific intervals and transformations, providing a mathematical framework for dealing with complex control flows. introduced "flow hypergraphs", an extension of traditional flowgraphs. The expansion of the idea of reducibility to this new class of structures aims to establish constraints that maintain the resulting family's ability to model real-world problems, while ensuring algorithmic efficiency. This entails careful analysis of the specific structural properties of flow hypergraphs and determining how certain constraints affect the ability to represent efficient problem solving (Faria, Guedes and Markenzon 2021).

[22] Visual representations that help to understand the sequence and decisions in computer programs. They are useful for analyzing the structure, identifying execution paths, and understanding the complexity of the algorithm.

The importance of directed hypergraphs as a generalization of directed graphs, capable of modelling binary relationships between subsets of a given set, has been highlighted in several works. These types of hypergraphs, as previously discussed, are applicable in multiple areas of Computer Science, including SBD, program analysis, and parallel computing.

The notion of reducibility, originally introduced in the context of directed graphs, constitutes a relevant concept in the study of control flow in computer programs. By extension, hypergraph is said to be reducible if it can be transformed into a smaller hypergraph without loss of essential structural information.

Informally, a hypergraph H is reducible if some of its hyperedges can be reduced until some remain. Formally, a hypergraph H is reducible if there exists a non-empty set of hyperedges R and a hypergraph H′ such that:

- H′ contains all the nodes in H;
- H′ contains exactly the hyperedges in H that are not in R;
- Each hyperedge in R can be expressed as a combination of hyperedges in H′, that is, each hyperedge in R can be "reduced" to a combination of hyperedges in H′.

For example, in hypergraph H = ({1, 2, 3, 4, 5}, {{1, 2, 3}, {2, 4}, {4, 5}, {3, 4, 5}}) the hyperedge {3, 4, 5} can be reduced to hyperedges {4, 5} and {3, 4} (Guedes, Markenzon, and Faria 2011, Allen 1970):

- {4, 5} is a hyperedge in H';
- {3, 4} is a hyperedge in H';
- {3, 4, 5} = {3, 4} U {4, 5}.

In this way, we can create a hypergraph H' = ({1, 2, 3, 4, 5}, {{1, 2, 3}, {2, 4}, {4, 5}, {3, 4}}), which is equivalent to H in terms of its important structural properties.

Essentially, the reducibility of a hypergraph implies the ability to simplify it while preserving its nodes and certain hyperedges. In the context of hypergraph coverage, where structural simplifications are allowed without loss of essential information, this reducibility property becomes relevant to achieve algorithmic efficiency in solving complex challenges in several areas of Computer Science.

3.3.12 Hierarchical Hypergraphs

Hierarchical hypergraphs (Figure 49) are a variation of hypergraphs in which hyperedges can be divided into sub hypergraphs, forming a hierarchy. This approach can be useful in several applications that involve organizing and structuring complex data, such as databases and geographic information systems. This approach has been explored in several studies, such as the work on the use of hierarchical hypergraphs of Chathurika et al. (2023).

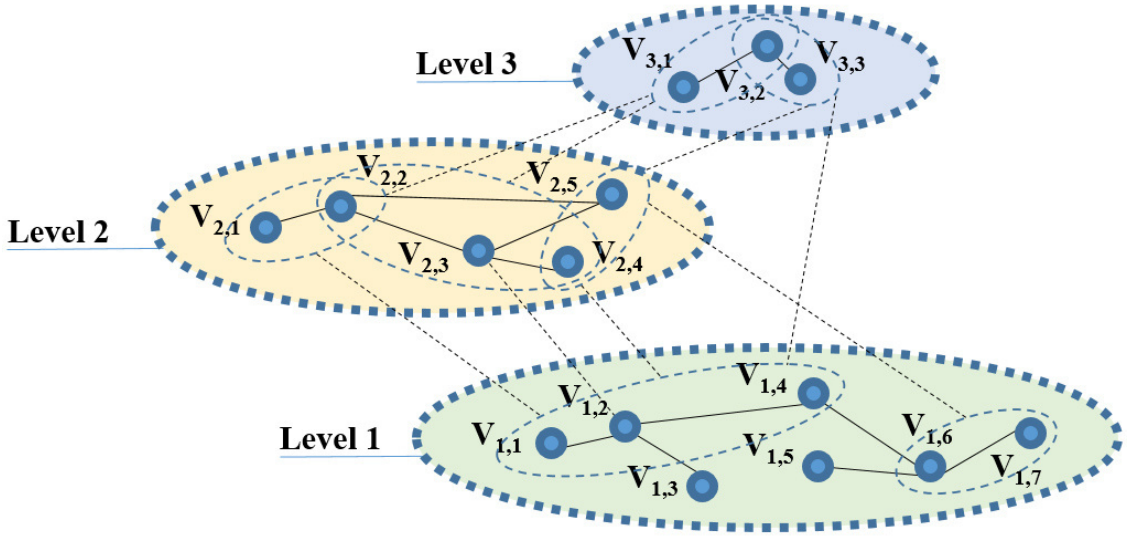


Figure 49 - An example of a hierarchical hypergraph.

The new MLHG functional abstraction, to be presented in detail in Chapter 4, redefines real problem modeling by focusing on inherited relationships between entities. Its implementation denotes the concept of hypergraph hierarchy, combining mathematical and computational formulations and adopting a data structure based on graphs of multiple levels of abstraction. The innovation of the MLHG stands out for the introduction of leveled hyperedges, which establish relationships by inheritance between entities of sequential levels. This logical governance structure outlines the relational flow between the different levels, providing a complete and interconnected view of the relationships between entities.

3.4 Models and Matrix Approaches

This chapter introduced hypergraphs, highlighting their potential to represent complex interactions between sets of elements. Several types of hypergraphs were explored, as summarized in the Table 5,

each with its own characteristics and structures that enable a more adequate modeling of different forms of interaction and relationship between the elements.

Table 5 - Different types of hypergraphs considered.

Hypergraph	**Hallmarks**	**Applications**
Uniform	All hyperedges have the same number of nodes.	Modeling consistent interactions with a fixed number of elements.
Two-Party	Dividing vertices into two sets, hyperedges connect different sets.	Modelling relationships between two different categories of elements.
k-Matches	Dividing vertices into k sets, hyperedges connect different sets.	Representation of relationships between various categories of elements.
Intervals	Assigning ranges to vertices, hyperedges contain overlapping vertices.	Modeling interactions of elements with overlapping features.
Weighted	Assigning weights to hyperedges.	Representation of relationships where intensity or relevance varies.
Targeted	Direction attributed to hyperedges.	Modelling of direct relationships between elements.
B-Hypergraph	Specific model of bisplit hypergraphs.	Representation of relationships in systems with specific bipartition.
F-Hypergraphs	Focused hypergraph framework.	Emphasis on specific subsets of elements in interactions.
Flow	Modeling of flows in complex systems.	Flow representativos in datasets.
Colorful	Assigning colors to vertices or hyperedges.	Analysis of color patterns in networks with a focus on resource allocation.
Hierarchical	Hierarchical structure with sub-hypergraphs, useful in organizing complex data.	Hierarchical data modeling in information systems.

However, a generic understanding of some types of hypergraphs is essential to lay the foundations for the next chapter, in which we will present the new functional abstraction called MLHG to solve real

problems. Here, we discuss the existing models and representations used to describe and analyze hypergraphs – the incidence matrix, representations through edge sets, and adjacency lists. Even taking into account that all models and representations are partially explored, because each one provides a specific model of visualizing them, the understanding of hypergraphs and their uses in different perspectives opens opportunities to extend the analysis of complex systems with a better understanding of the phenomena and development of more efficient algorithms. This is because they offer a more detailed description of the complex interactions that occur in various domains, allowing for correct modeling and a deeper understanding of what is being studied. In the next chapter, we will propose the concept of abstraction MLHG, which is based on many of the concepts, properties, and representations discussed in this chapter. As we will present, MLHG will be appropriate to treat real-life problems due to its flexibility, since problem modeling consists of multi-dimensions. We will use the properties of hypergraphs to provide a more versatile and important description of the actual problem to be addressed. Therefore, the proposal of functional abstraction will be an extension of all the knowledge acquired, a new way of modeling real-life problems.

3.5 Current Limitations in Hypergraph Manipulation Tools

Despite the theoretical advances in the study of hypergraph, their practical adoption in computational environments still faces several critical limitations. Such constraints focus mainly on the ability of existing tools to represent, manipulate and visualize n-ary relationships in a scalable way, especially when the need for multilevel and hierarchical organization is introduced — a central aspect in the proposal of the MLHG model. Libraries such as *HyperNetX* and *xGi* (both developed under the umbrella of Anaconda Inc.'s NetworkX project) offer relatively affordable interfaces for modeling simple hypergraphs but demonstrate some weaknesses when trying to scale models to more complex structures. Studies such as HyperNetX show that while HyperNetX supports basic hypergraph construction and visualization, it does not implement semantic data manipulation operators, such as create, read, update, and delete (CRUD), adapted to the hypergraph paradigm. One of the main challenges is the absence of native support for hierarchies and multi-level encapsulation. Other tools, such as xGi (Extreme Graph Interface), although they can model complex hyperedges, do not offer mechanisms to structure subsets of vertices or hyperedges at multiple logical levels. This limits the representation of real scenarios in which entities and relationships have a context dependent on functional layers, as occurs, for example, in modular project

management or in the representation of sociotechnical systems. On the other hand, the interactive visualization of complex hypergraphs remains a weak point. As pointed out by, most tools use static representations or use inflated hyperedge models that make it difficult to understand the underlying topology. There are, so far, no widely adopted graphical interfaces that support intuitive interactions with dynamic hypergraphs structures, except for experimental projects such as D3 Hypergraph and Tulip Hypergraph Plugin, both of which are poorly maintained (Praggastis et al. 2024b, Ouvrard 2020).

In terms of algorithmic efficiency, classical graph algorithms are not directly applicable to hypergraphs, requiring adapted versions to perform tasks such as partitioning or random walking. Existing libraries still lack robust implementations of these adapted versions. Already in 2009, they warned of the inefficiency of conventional algorithms when applied to hypergraphs in networks of biochemical reactions, a domain that requires high precision in n-ary dependencies (Klamt, Haus and Theis 2009).

In terms of integration with BDOG, hypergraphs are not yet treated as first-order entities. Neo4j, a graph-based database management system, for example, requires the simulation of hyperedges via intermediate nodes, which compromises semantics and efficiency. As they point out, this type of artificial modeling increases the computational cost of queries and makes it difficult to maintain the data model. These gaps make evident the need for a structure such as the MLHG, which combines expressive formalization with an efficient operationalization, natively prepared to represent multiple levels of abstraction and perform CRUD operations in a standardized way. MLHG aims to fill the void left by current tools by offering robust support for both modeling and computational implementation of complex systems (Klamt, Haus and Theis 2009).

3.6 Concluding Remarks

This chapter has established a comprehensive contextualization of the field of hypergraphs, examining their essential characteristics, different typologies, models of representation, and multiple practical applications. It has been shown that hypergraphs offer a more expressive framework than conventional graphs, particularly in representing simultaneous interactions among multiple elements. Their ability to capture higher-order relationships makes them especially relevant for domains in which granularity and semantic density challenge the limits of traditional binary approaches.

The increasing complexity of contemporary systems — from digital infrastructures to social networks or organizational processes — requires richer and more flexible data models that can reflect

internal dependencies and structural hierarchies more accurately. In this context, hypergraphs emerge as an important solution, allowing for more sophisticated approaches to segmentation and analysis. Its integration with multilevel partitioning algorithms, as explored in diverse sources (Zhou et al. 2021), proves to be promising, expanding the possibilities of hierarchical decomposition and efficient manipulation of large volumes of data.

The adoption of hypergraphs as a modeling paradigm thus represents a significant reconfiguration of analytical and computational practices in data sciences, systems engineering, knowledge modeling, and many other domains. Their transversal applicability and ability to adapt to multiple contexts make them indispensable tools in the future of complex data representation.

Thus, this chapter established the necessary foundations for the next step of this work: the formulation of the MLHG model. It aims to extend the capabilities of classical hypergraphs, integrating hierarchical, semantic and operational mechanisms that allow representing and manipulating complex systems with greater granularity, coherence and expressiveness. The MLHG proposal arises, therefore, as a direct response to the identified gaps and represents an innovative contribution to graph-driven data modeling in modern computational environments.

4 Multilevel Hypergraph

Complexity has been an increasingly evident feature in a wide range of fields, from social media to project management in construction. In a world where the interactions between the resources needed for construction — such as materials, labor, or equipment — can be vast and intricate, the need for an innovative approach to modeling and managing these complex systems naturally arises. In this context, we propose in this work a new functional abstraction called MLHG. The proposal of this variant of graphs, combined with the previously discussed structure of the MLD — aims to increase the semantic expressiveness of hierarchical representations, as well as enhance the operational analysis of complex systems through the segmentation, encapsulation and semantic compression of the information to be maintained in a graph. In addition to validating isolated algorithms, this chapter consolidates an architecture that supports the hierarchical modeling of complex systems. By aligning the results obtained with the objectives of the research and with the challenges documented in the literature, it seeks to offer the foundations for the future adoption of MLHG/MLD models in practical tools of Systems Engineering, Data Science and Knowledge Management. Today, we know that high-order associations between elements are difficult to describe topologically using graphs. However, in some circumstances hypergraphs can add valid topologies to the formulation of intricate data correlations, introducing mathematical expressions such as complex connective systems.

The use of hypergraph-based models for a more comprehensive analysis of real-world problems often requires transforming the original hypergraph structure into an equivalent graph representation. Several methods have been proposed to perform this topological transformation, among which the most common are illustrated in Figure 50 (Dai and Gao 2023).

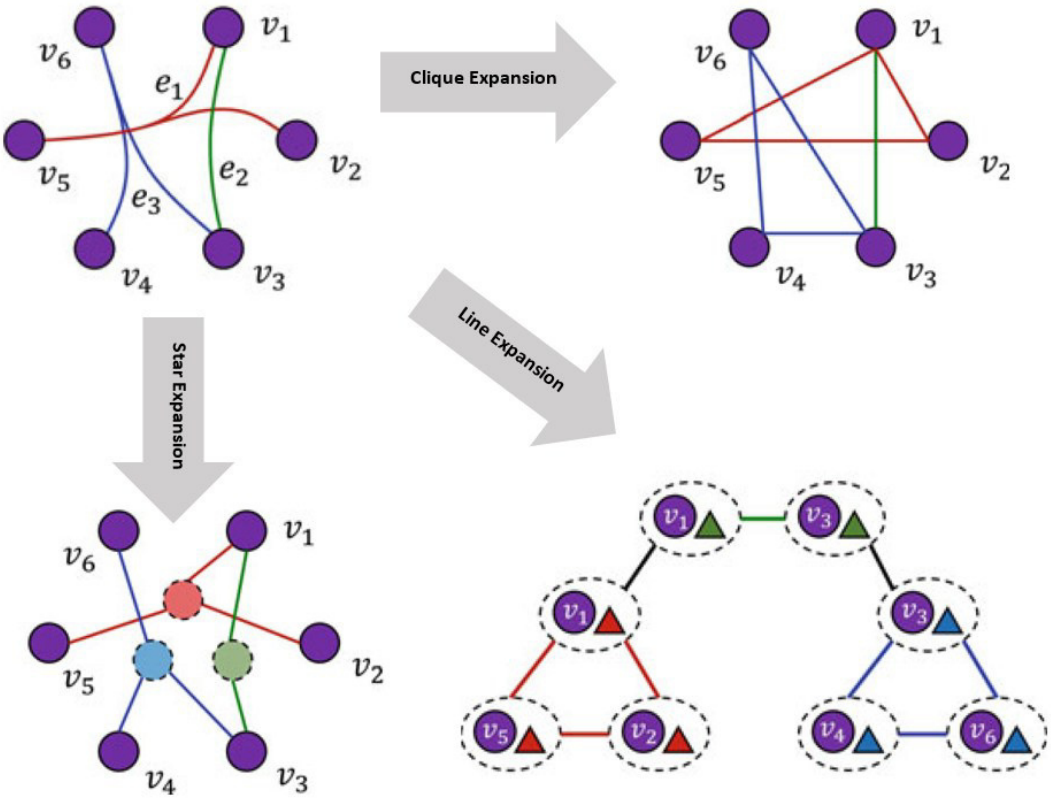


Figure 50 - Methods for the topological transformation of hypergraphs (adapted from Dai and Gao, 2023).

These methods allow the hypergraph structure to be represented in a simpler graph format, while preserving different aspects of the connectivity between the original vertices. Each method has its own characteristics, being chosen based on the specific needs of data analysis or modeling. In summary, these transformations offer different ways of projecting higher-order relationships in a conventional graph, allowing a better understanding of how representation choices influence structural interpretation and motivating the development of more expressive models, such as MLHG.

- Click Expansion. In this method, each hyperedge is replaced with a set of edges that connect all pairs of vertices within the hyperedge. This creates a graph in which the vertices within a hyperedge form a clique;
- Star expansion. In star expansion, each hyperedge is mapped to a new vertex in the graph. This vertex is connected to all vertices within the hyperedge in the original hypergraph;

- Line Expansion. In line expansion, each vertex in a hyperedge is mapped to a new vertex in the graph. Each new vertex represents a point in the original hyperedge. The vertices corresponding to the same hyperedge in the original hypergraph are considered neighbors in the resulting graph.

The idea for developing the MLHG model emerged from the realization that conventional data structures cannot satisfactorily deal with the complexity and interconnectedness that characterize many contemporary systems. Classical models, although effective in simple situations, are limited when they are intended to represent higher-order relations or articulate different levels of abstraction. From this problem arose the proposal of the MLHG, conceived as a hierarchical structure capable of organizing, in a coherent way, information distributed by several relational levels.

The core objectives of MLHG can be summarized as follows:

- Overcome the difficulties in implementing hypergraphs. Although hypergraphs offer an expressive way of representing n-ary relationships between entities, their practical application continues to raise several problems. Among the most frequent are the complex management of hyperedges, the absence of stable norms for the representation of multiple incidences, the lack of adaptation of many traditional algorithms, and the scarcity of tools that allow exploring and visualizing these structures with ease;
- Increase the scalability of graph-based systems. MLHG is intended to expand the growth capacity of graph-oriented databases, ensuring that the increase in the volume of information does not compromise the performance or consistency of relationships. The model supports the creation of more efficient management mechanisms, capable of maintaining structural stability as systems become more extensive and interconnected;
- Strengthen transparency in data analysis. With MLHG, it is sought to make graph-based analytical processes clearer and more verifiable, allowing us to understand how each relationship contributes to the inferences and results obtained. This clarity favors the development of methods capable of justifying, in an accessible and reasoned manner, the decisions extracted from the models;
- Support decisions in real contexts. In addition to the technical component, the MLHG was designed as a decision support instrument in complex scenarios, in which multiple factors and dependencies coexist. The model helps to reveal patterns and relationships that would otherwise remain hidden, offering useful information to guide strategic choices in domains as diverse as engineering, management or systems analysis.

The defined objectives, combined with the ability of MLHG to represent and analyze complex systems, can significantly change the way organizations manage information and inform their decisions. Based on this principle, a first draft was drawn up in Figure 51, which proposes a more comprehensive structure than the MLD presented in Chapter 2. In this new approach, the abstraction process can be repeated hierarchically in different higher-level entities (vertices), and extend successively to even higher levels, without any limitation as to the degree of the vertices—a constraint present in the original MLD.

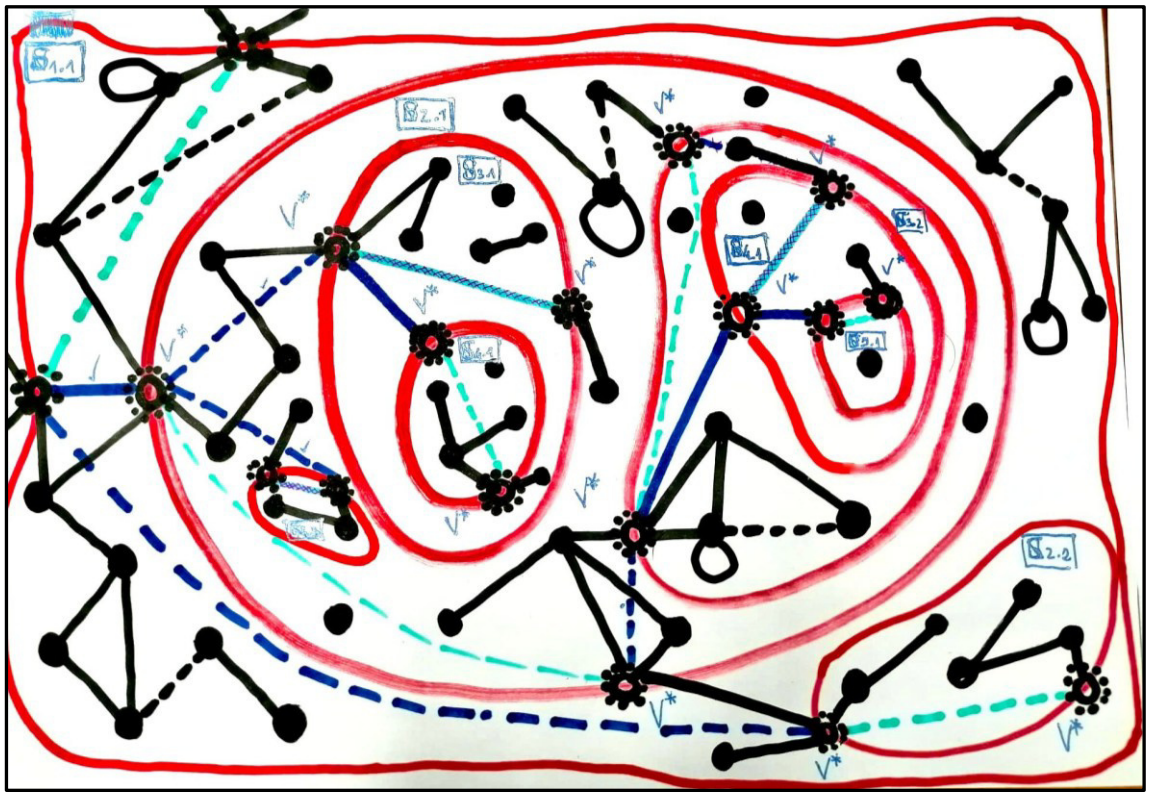


Figure 51 – An outline draft of the initial structure of the MLHG.

Graphs and hypergraphs are remarkably tools for representing real-world situations. It is through mathematical abstraction and computational modeling that it becomes possible to understand and solve complex problems, translating diverse realities into forms that analytical thinking can manipulate. For each type of problem there is almost always a more appropriate structure. The choice of a particular graph or hypergraph results from this correspondence between what one wants to represent and the way in which one intends to explore the relationships involved. The simplest case is that of the graph formed only by one vertex, V, without edges. This vertex, although isolated, symbolizes a single and perfectly defined entity within a given context.

In conceptual terms, one can imagine that this vertex contains within itself the notion of its own existence — as if to recognize that it is only the simplified reflection of other possible entities, situated at a more comprehensive level of description. Such a process of abstraction, in theory, has no end: with each new level, a new layer of meaning can emerge. In practice, however, it is the human understanding itself that imposes a limit. There is a point at which the degree of abstraction becomes sufficient to apprehend the system and intervene in it, stabilizing the representation and allowing viable solutions to be found for the problem studied.

4.1 The Structure of an MLHG

Mathematical abstraction and computer modeling can help solve real-world problems in a variety of ways. Mathematical abstraction is necessary to systemically describe the phenomena being studied. That is, when you model a problem mathematically, you are describing the problem through equations, relationships, and mathematical concepts that help you understand the problem. On the other hand, computer modeling is necessary, because it helps to simulate and analyze how systems behave. Using computer modeling, we can see what happens when you change a value and how the system reacts to the changes. The proposed data structure for an MLHG is based on the graph representation of multiple abstractions. Relationships between entities in level abstractions are established in a traditional way, while relationships between multiple abstractions are assigned for each abstraction to a leveled virtual entity. MLHG, therefore, is a new generalization of hypergraphs, which allows describing relationships between objects.

The mathematical-computational representation of problems through this new functional abstraction will allow the modeling of multi-level relational relationships, which usually cannot be adequately described by other abstractions of hypergraphs. MLHG allows you to represent graphs and hypergraphs at multiple levels of abstraction. Generally, a graph G is defined abstractly as a double G(V, E), where V represents the vertices and E the edges that connect these vertices and establish the relationships necessary to model the relationships between the entities. However, the mathematical-computational representation of a graph is often obtained by means of classical matrix structures, such as matrices or adjacency or incidence lists, which are not always suitable for the representation of complex graphs. The MLHG proposal uses a tetrad ($V_{i,n}$, $A_{i,n}$, $V_{i,n}$*, $E_{i,n}$) as a mathematical-computational representation, conjugated by matrix structures of adjacency, trees of logical structure of levels ($A_{i,n}$) and

the new concept of RI. The MLHG has several benefits and applications in different areas of knowledge, namely in the field of Data Science and Engineering. Its use will allow a more complex and detailed representation of the relationships between entities, in addition to enabling analysis at multiple levels of abstraction. It is also noteworthy that MLHG presents innovative and differentiated aspects in relation to other existing functional abstractions, such as Hypergraphs.

An MLHG ($V_{i,n}$, $A_{i,n}$, $V_{i,n}$*, $E_{i,n}$) can be represented graphically by a "multilevel hypergraph", in which the vertices are organized at different levels of abstraction (Figure 52). The vertices ($V_{i,n}$) represent the entities at the abstraction level n, while the logical structure trees of levels ($A_{i,n}$) govern the flow between the vertices of the various levels. The virtual vertices ($V_{i,n}$*), representative of hyperedges that are candidates for hypergraph at the n level, and the edges ($E_{i,n}$) materialize the relationships between the vertices $V_{i,n}$ and $V_{i,n}$*. The corresponding graphic representation is obtained by means of the adjacency matrix structure, supported by a dictionary of labels, in which each vertex is represented by a row and a column of the matrix. In Figure 52, we can see how the vertices of MLHG are organized into different levels, where the entities represented by Vi.ni at the ni level are elements of a multilevel system. Each of these can be connected, or not, to other entities of the same level or to virtual entities $V_{i.ni}$* through generic Ei.ni relationships. These connections form a complex network of interactions that describe the structure and dynamics of the system at multiple levels of abstraction. Each virtual entity $V_{i.ni}$* is associated with a Gi.ni graph, allowing a visual representation of the connections between entities at a given level, which facilitates the analysis and understanding of the relationships between the components of the system at different levels of detail.

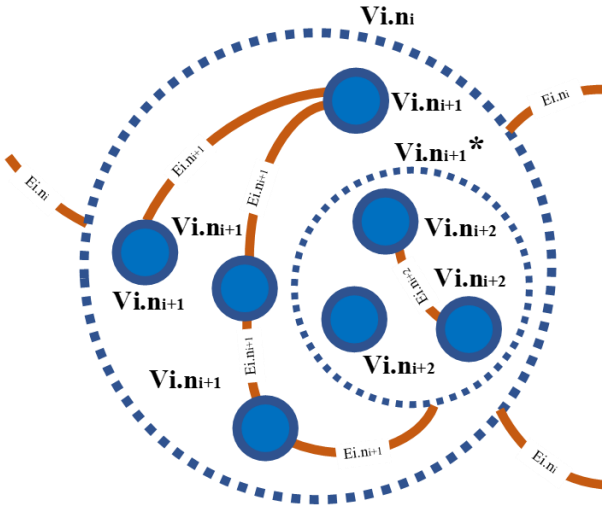


Figure 52 – A graphical representation of a generic MLHG.

As we increase the levels of abstraction, generic relationships $E_{i,n+1}$ is established at the next level n_{i+1}, connecting the entities of the current level to the entities of the next level ni+1. This process continues with the introduction of generic $E_{i,n+2}$ relationships at the ni+2 level, further increasing the complexity of interconnections within the system. This hierarchical structure of relationships is at the heart of understanding the organization of the system, offering a comprehensive view of its interactions at all levels. Through this representation, it is possible to analyze and model the behavior of complex systems, identifying patterns that may not be evident in a conventional analysis. In this way, the graphical representation of MLHG allows you to visualize the different entities and relationships present at a given level of abstraction, facilitating the understanding and analysis of the model. The trees of logical structure of levels, ($A_{i,n}$) have an essential function in defining the hierarchical flow of the relationships between the different levels of abstraction of the MLHG, functioning as a function of hierarchical incidence, which groups and hierarchizes all the components of the MLHG. As illustrated in Figure 53, each tree begins with a root representing the vertices ($V_{i,n}$) and the virtual vertices ($V_{i,n}$*), at the ni level. As the tree develops to a higher level, the virtual vertices ($V_{i,n}$*) turn into branches, which feed other branches and provide leaves (vertices) at the next higher level, n+1.

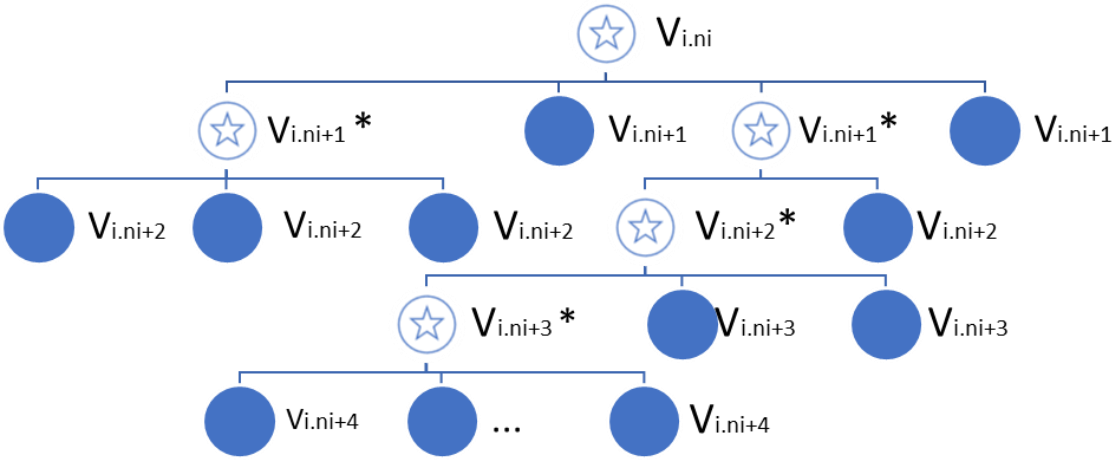


Figure 53. A logical level structure tree ($A_{i,n}$) of an MLHG.

Level logical structure trees provide a hierarchical structure for the flow of relationships between the different MLHG abstraction levels. For this purpose, a structure was created for the computational labeling of vertices, which represent the entities at the level of abstraction n. This structure consists of a unique index for each vertex V, in the form <i(n1).i(n2).i(n3).i(n4)... i(nn)>. The index size makes it possible to set the cardinal of the level, as well as the cardinal of the vertex at each level that the stream

traverses. For example, the label <1.3.4.2> refers to a vertex, $V_{1.3.4.2}$, at the maximum level, n=4, where it represents the second vertex of that level 4, the fourth vertex of level 3, the third vertex of level 2, and the first vertex of level 1. This labeling structure is important for the accurate and unique identification of each vertex in an MLHG.

Figure 54 illustrates the adjacency matrix of the tree structure of an MLHG, evidencing the importance of this tool for the understanding of the graph. Similarly, the use of virtual vertices ($V_{i,n}$*) allows MLHG to adequately represent cases where an entity has not yet been fully defined or is not relevant to the analysis in question. This allows for the coherent and efficient organization of the graph representation of multilevel of abstraction, facilitating the understanding of the relationships between the entities involved in each level of abstraction.

	V1	V1.1	V1.2	V1.3	V1.4	V1.5	V1.6	V1.7	V1.8	V1.1.1	V1.1.2	V1.3.1	V1.3.2	V1.3.3	V1.3.4	V1.5.1	V1.5.2	V1.5.3	V1.5.4	V1.5.5	V1.3.4.1	V1.3.4.2	V1.5.2.1	V1.5.2.2	V1.5.2.3
V1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.3	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
V1.4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
V1.6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.1.1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.1.2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.3.1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.3.2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.3.3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.3.4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
V1.5.1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.5.2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
V1.5.3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.5.4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.5.5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.3.4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
V1.3.4.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
V1.5.2.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
V1.5.2.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
V1.5.2.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 54 – An adjacency matrix of the tree structure of an MLHG.

With the adjacency matrix of the logical level structure tree, you can directly identify the virtual vertices ($V_{i,n}$*) and edges ($E_{i,n}$) associated with each MLHG abstraction level. This approach allows a more organized and efficient representation of the hypergraph of multilevel of abstraction, which facilitates analysis of the relationships between the entities involved, through the labels of their vertices. The choice of the adjacency matrix (Madj) was considered as the most appropriate structure to represent the MLHG

mathematically and computationally. The adjacency matrix is a square matrix of dimensions (V x V) that can be denoted by [V_{ij}]. This structure was chosen for a few important reasons: it is possible to know its dimensions in advance, it is recommended for dense graphs, it allows the creation of algebraic operators, and it has a numerical predisposition to create digraphs. The definition of the adjacency matrix is given simply by the condition:

$$V_{i,j} = \begin{cases} 1 \text{ if an edge exists between the vertices } V_i \; e \; V_j \\ 0 \text{ if no edge exists between the vertices } V_i \; e \; V_j \end{cases}$$

Although the use of an adjacency matrix to store a graph requires an O(n^2) memory space, in our case, this will be justified by the small size of the matrices necessary for the development and implementation of the works and by the relational density of the state graphs computed by those studies in Chapter 5. The algorithm created to build the adjacency matrix and its associated vertex labeling dictionary of an MLHG ($V_{i,n}$, $A_{i,n}$, V_{i,n^*}, $E_{i,n}$) was developed with the objective of ensuring relational integrity. This is important because there is inheritance of relationships between each level of abstraction n and its successor level, n+1. That is, for a given relationship $E_{i,ni+1}$ existing at the ni+1 level between two vertices of this level: $V_{i,ni+1}$ and another virtual vertex, $V_{i,ni+1}$*, candidate to be governed by constituents at the upper level, ni+2, it is necessary to ensure replication by inheritance of relationships, $E_{i,ni+1}$ (edges), of the vertex $V_{i,\,ni+1}$ for all vertices $E_{i,ni+2}$, constituents of $V_{i,ni+1}$*, which in this process will no longer be part of the ni+2 level.

As illustrated in the generic example presented in Figure 55, in the process of RI between levels, due to a certain relationship $E_{i,ni+1}$ at the ni+1 level, which related the vertex $V_{i,ni+1}$ with the virtual vertex $V_{i,ni+1}$* at the ni+2 level, it is observed that, although a relationship is lost, three new relationships emerge, one for each vertex contained in $V_{i,ni+1}$*.

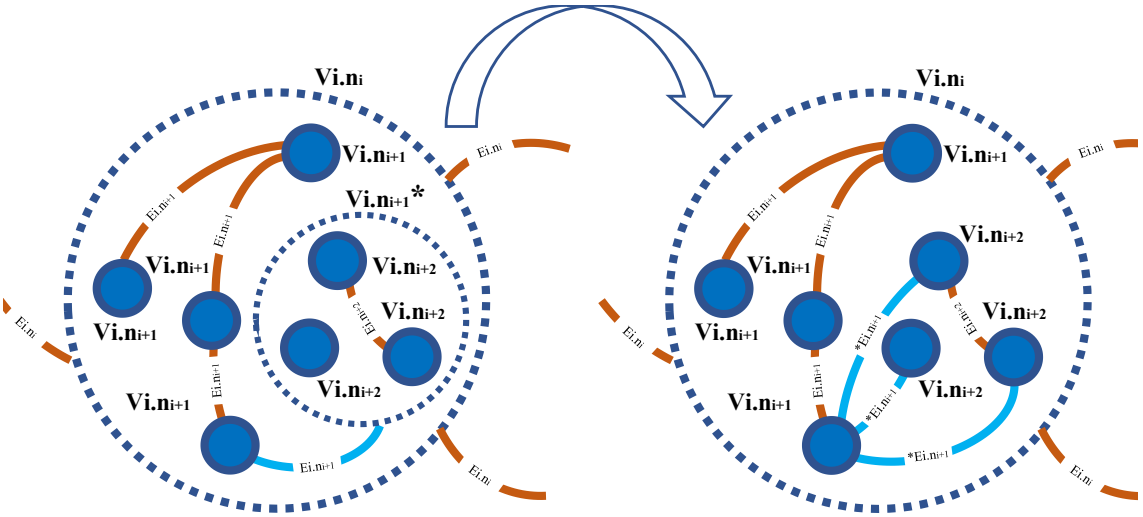


Figure 55 – Showing the inheritance process between consecutive levels.

The pre-dimensioning of the adjacency matrix, Madj, (V x V) is established for each MLHG ($V_{i,n}$, $A_{i,n}$, $V_{i,n}*$, $E_{i,n}$) by the expression (2) where, $|V_{i,n}|$ and $|V_{i,n^*}|$ respectively, represent the cardinality of the vertices (entities) and the cardinality of the virtual vertices, at the level of abstraction n.

$$V = \left(\sum_n \sum_i \left|V_{i,n}\right| - \left|V_{i,n}{}^*\right|\right) - 1 \qquad (2)$$

In order to construct the Madj matrix of MLHG effectively, it is necessary to define previously for each level n, all the singular adjacency matrices Madj (n+1) of the graphs G($V_{i,n+1}$, $E_{i,n+1}$), considering all the vertices $V_{i,n+1}$ contained in the n+1 level, including the virtual vertices, $V_{i,n+1}*$. Then, the succession guided by the tree $A_{i,n}$ from the n+1 levels to the highest level is made, and through a leveled recursive process of expansion of the Madj(n+1) matrix, the various Madj(n+1) are assembled into a single level, where Madj(n+1) is completed by recurrence with the replication of values [V_{ij}] corresponding to the relationships inherited from lower levels (n-levels).

The necessary creation of the MLHG Madj ($V_{i,n}$, $A_{i,n}$, $V_{i,n}*$, $E_{i,n}$) is carried out according to the following algorithm:

Input: Graph labeled by, including all virtual vertices in $.G\left(V_{i,n_i}, E_{i,n_i+1}\right)V_{i,n_i}, V^*_{i,n_i+1}, V_{i,n_i+1}$

Initialization:

1. Establish, in the conventional way, the adjacency matrix for the initial level $.\mathrm{Madj}(G)n_i$
2. Level Iteration: For each level, until the global matrix is obtained, do:$n + k$ Madj(MLHG)

2.1. For each virtual vertex with incident edges, proceed as follows:$V^*_{i,n_i+1}E_{i,n_i+1}$

a) Define with dimension, where ;$\mathrm{Madj}\left(V_{i,n_i+2}; E_{i,n_i+2}\right); (V^* \times V^*); V^* =| V_{i,n_i+2} |$

b) Using the current adjacency matrix as reference, execute the following three steps:

Step 1 – Dimensional expansion: Expand the matrix at the row and column corresponding to the position of vertex, adding new rows and columns. $(i\text{-th} + 1); V^*_{i,n_i+1}; (V^* - 1)$

Step 2 – Inheritance replication: In the new coordinates created in *Step 1*, replicate all row and column values associated with from the current matrix $.V^*_{i,n_i+1}; \mathrm{Madj}$

Step 3 – Subgraph integration: Integrate the subgraph matrix directly into the coordinates corresponding to the virtual vertex, starting from the original position of $.\mathrm{Madj}(V_{i,n_i+2}; E_{i,n_i+2}); V^*_{i,n_i+1}$

Output: The resulting matrix contains all hierarchical relations and inherited structures from the lower levels. $\mathrm{Madj\ (MLHG)}$

The algorithm presented, in this way, makes it possible to construct an adjacency matrix for a multilevel hypergraph MLHG that contains virtual vertices and levels of abstraction, n. The adjacency matrix (Madj (G)) at the first level is initialized according to the standard convention for labeled graphs. The algorithm iterates through each virtual vertex $V_{i.ni+1^*}$ with edges $E_{i,n+1}$ at each level n+k, and establishes a new adjacency matrix Madj ($V_{i.ni+2}$; $E_{i,n+2}$) with dimension V* x V*, where V* is the number of vertices in $V_{i.ni+2}$. To expand the dimension of the current Madj array and accommodate the new Madj array ($V_{i.ni+2}$; $E_{i,n+2}$), the algorithm performs three steps. The first step is the dimensional expansion of the matrix in the (i-thimum+1) row and column referring to the position of the vertex $V_{i.ni+1}$*, by adding |V*|-1.

The second step replicates the constant values with respect to the virtual vertex Vi.ni+1* in the new coordinates resulting from step 1, per column and per line. The third step is the direct integration of the new Madj matrix into the correct coordinates in the current Madj matrix, starting from the initial coordinates of vertex $V_{i.ni+1^*}$. For virtual vertices without edges $E_{i,n+1}$, the algorithm performs a similar process, but with a dimensional expansion starting at the origin of the adjacency matrix, and with the assignment of zeros in all new coordinates.

The result is a Madj (MLHG) adjacency matrix that represents the graph G with levels and without virtual vertices. The algorithm is efficient because it uses constant values and direct integration processes to minimize the number of calculations required to construct the final adjacency matrix. On the other hand, value replication reduces the amount of memory accesses required to populate the new Madj array,

making the algorithm faster and more space efficient. In order to clarify the algorithm described above, we present in the Figure 56 an example of an MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$), composed of four levels of abstraction, where it is possible to visualize the hierarchical structure of the vertices and edges, representing each level of abstraction of the multilevel hypergraph.

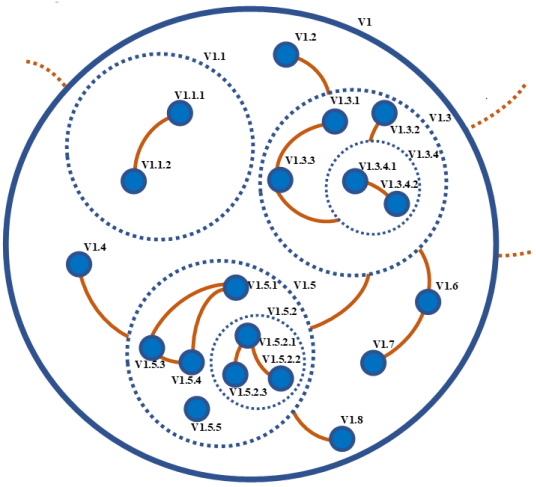
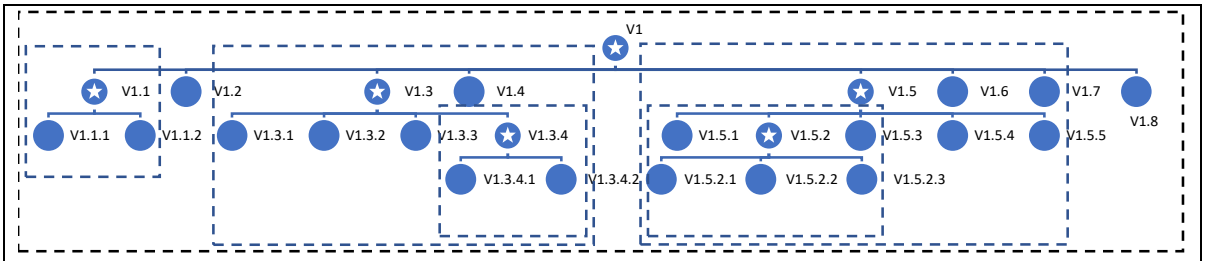


Figure 56 - Multilevel Hypergraph, MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$).

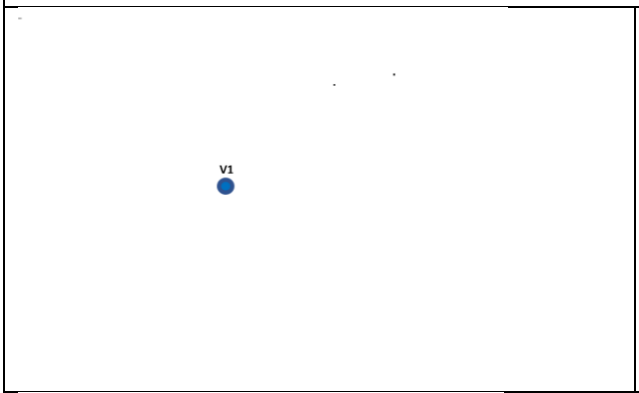
The Table 6, presents the logical structure tree of the levels and the initial sequence of formation of the example MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$), composed of four levels of abstraction. This representation shows how the entities and their connections are organized in the early stages of the model, serving as a basis for the subsequent hierarchical construction. In

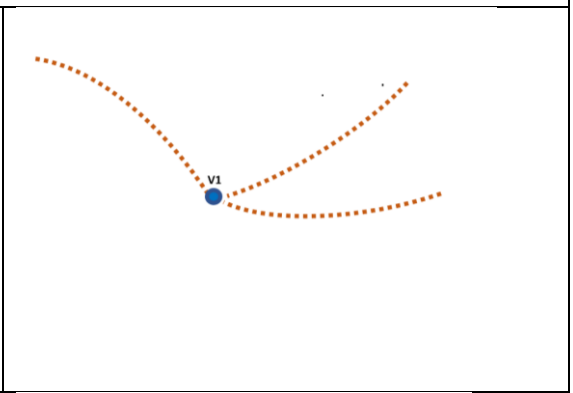
Table 7 describes final process sequence, which comprises the development of MLHG from level 2 to level 4, already reflecting the transformations resulting from the integration of virtual vertices and the consolidation of relationships between levels. Certain virtual vertices — $V_{1.1}$*, $V_{1.3}$*, $V_{1.5}$*, $V_{1.3.4}$* and $V_{1.5.2}$* — were not included in the final representation, as they were not part of the effective graph. These vertices correspond only to hyperedges of higher levels, used to demonstrate the abstraction mechanism of the model, without interfering in the concrete structure of the MLHG presented in the tables.

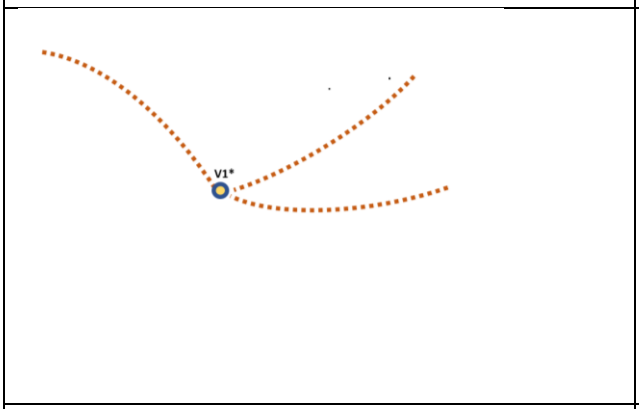
Table 6 - The level tree and the complete sequence of formation of an example, MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$).

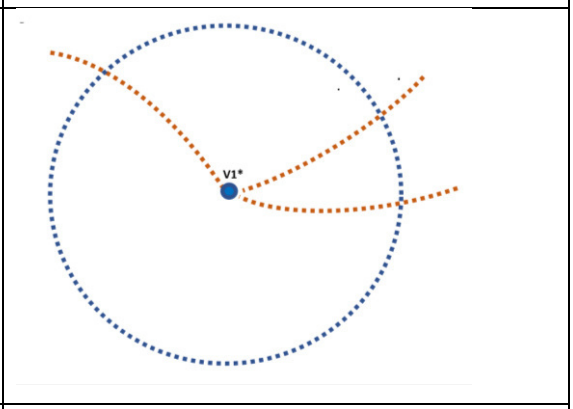


Tree structure, $A_{1.4}$



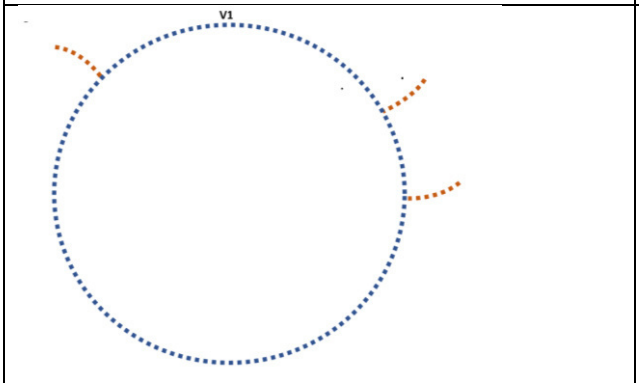


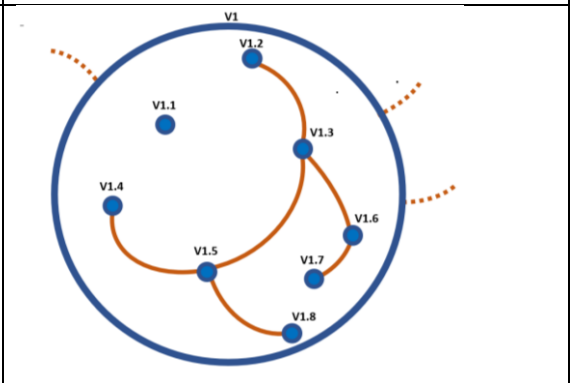




V_1, at level 1

V_1, at level 2

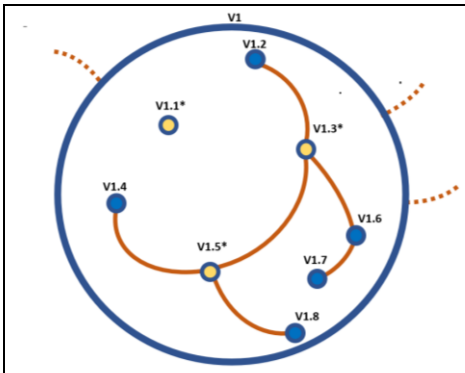
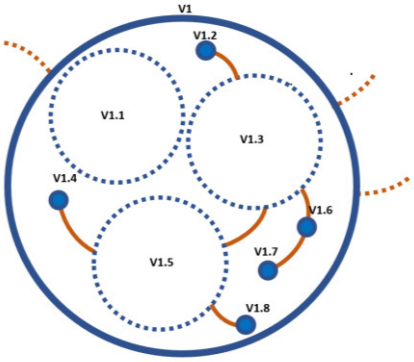
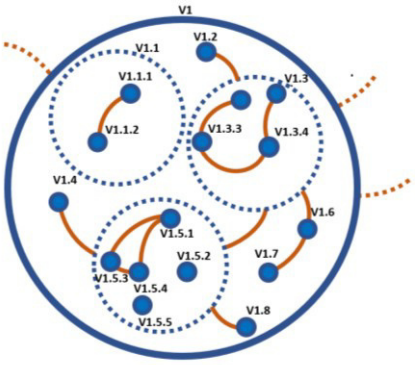
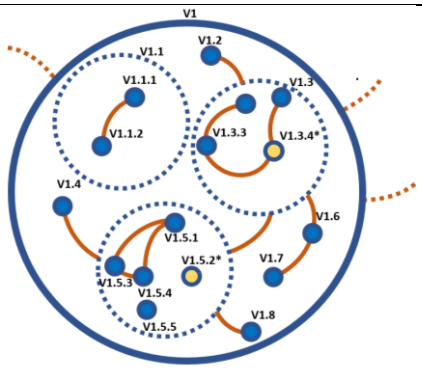
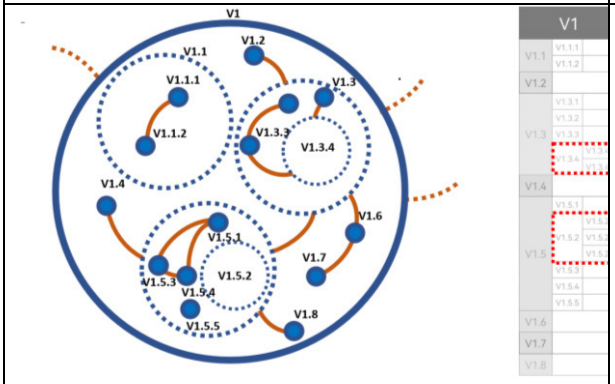
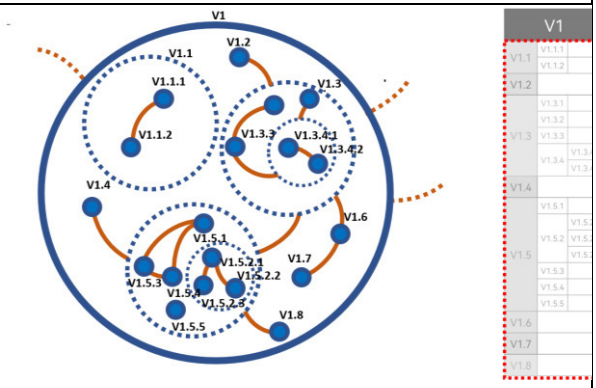




V_1, at level 2, suitable for creating level 2 elements

V_1, at level 2, with level 1 and 2 elements

Table 7 - Final training sequence from level 2 to level 4 of the MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$).

	
V_1, Level 2, Expanding Level 3	V_1, at level 2, after cutting your level 2 relationships and able to create level 3 elements
	
	
V_1, at level 3, after cutting your level 3 relationships and able to create level 4 elements	V_1, at level 4, with level 1, 2, 3 and 4 elements

To illustrate how the algorithm works, the three steps applied at a given level of the model are explained below.

Let us consider, as an example, level 3, which contains three virtual vertices:$V_{1.\#}^{*}$

Establishment of the base matrix

The adjacency matrix is constructed, in a conventional way, represented in the $\mathrm{Madj}(G)$ Figure 57. The graph is labeled $G_V_1 = G\left(V_{1,n_i+1}; E_{1,n+1}\right)$and defined including in all associated virtual vertices, namely, $V_{1,1}^{*}$; $V_{1,3}^{*}$; $V_{1,5}^{*}$.

V1	V1.1*	V1.2	V1.3*	V1.4	V1.5*	V1.6	V1.7	V1.8
V1.1*	0	0	0	0	0	0	0	0
V1.2	0	0	1	0	0	0	0	0
V1.3*	0	1	0	0	1	1	0	0
V1.4	0	0	0	0	1	0	0	0
V1.5*	0	0	1	1	0	0	0	1
V1.6	0	0	1	0	0	0	1	0
V1.7	0	0	0	0	0	1	0	0
V1.8	0	0	0	0	1	0	0	0

Figure 57 – Madj (G) matrix, where G is labeled by V_1.

- for each virtual vertex of level 2: $V_{1,1}^{*}; V_{1,3}^{*}; V_{1,5}^{*}$ and its Madj(), at level 3 (Figure 58).

V1.1*	V1.1.1	V1.1.2
V1.1.1	0	1
V1.1.2	1	0

V1.3*	V1.3.1	V1.3.2	V1.3.3	V1.3.4*
V1.3.1	0	0	1	0
V1.3.2	0	0	0	1
V1.3.3	1	0	0	1
V1.3.4	0	1	1	0

V1.5*	V1.5.1	V1.5.2*	V1.5.3	V1.5.4	V1.5.5
V1.5.1	0	0	1	1	0
V1.5.2*	0	0	0	0	0
V1.5.3	1	0	0	1	0
V1.5.4	1	0	1	0	0
V1.5.5	0	0	0	0	0

Figure 58 - Madj Matrices(), at level 3.

Virtual vertex processing $V^*_{1,1}$

As the vertex has no edges (), the three steps of the algorithm are executed on the current adjacency matrix, represented in the $\mathrm{Madj}\ (V^*_{1,1}; E_{1,2}) = \emptyset$, Figure 59:

- Step 1 – Dimensional expansion: In this step, the matrix is expanded, adding new rows and columns in the position corresponding to the origin of the vertex;$(V^* - 1)$.
- Step 2 – Definition of the new coordinates and initialization: In this phase, the new coordinates resulting from the expansion made in Step 1 are defined. All the values corresponding to the new rows and columns are initialized to zero, ensuring that the matrix expands its dimension without changing the existing relationships in the current; Madj.
- Step 3 – Direct integration of the subgraph: Direct integration of the matrix is performed at the coordinates associated with the source vertex. This step inserts the subgraph corresponding to the vertex into the overall structure, preserving the previously defined incidences and updating the matrix to the next level of abstraction. $\mathrm{Madj}\ (V_{1,1,2}; E_{i,3})$.

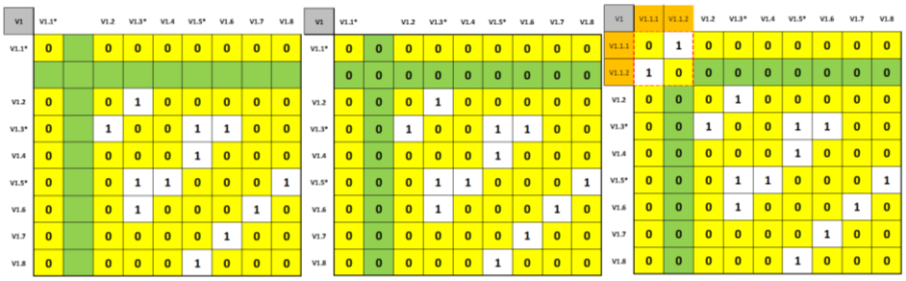


Figure 59 - Processing of the 3 steps, at level 2, for the virtual vertex.$V^*_{1,1}$.

Virtual vertex processing $V^*_{1,3}$

For the virtual vertex, the procedure follows the three steps defined in the algorithm, as described below $V^*_{1,3}$:

- Step 1 – Dimensional expansion of the matrix: The expansion is carried out at the position corresponding to the virtual vertex of the matrix. In this operation, the initial matrix, with dimension, $\mathrm{Madj}\ (9 \times 9)$, is enlarged by adding two rows and two columns, resulting

in a new dimension matrix, $\mathrm{Madj}\ (12 \times 12)$. This expansion creates the necessary space to incorporate the entities associated with the virtual vertex at the next level; $V^*_{1,3}$;

- Step 2 – Inheritance Replication: The row and column values corresponding to the vertex in the original array are replicated to the new positions of the expanded array, both per row and per column. This process ensures that the relationships incident to the virtual vertex are fully inherited by the new vertices created at the top level;$V^*_{1,3}$
- Step 3 – Direct subgraph integration: The level 4 matrix is integrated directly from the virtual vertex coordinates into the level 3 matrix. Thus, the new submatrix corresponding to the top level is inserted into the overall structure, maintaining the coherence of the hierarchical links and updating the adjacency representation to the expanded state of the $\mathrm{Madj}\ V^*_{1,3}$ MLHG.

Still at level 3, the virtual vertices and, both associated with the edges, are proceeded identically. The $V^*_{1,3}$; $V^*_{1,5}$; $E_{i,3}$ Figure 60 illustrates the corresponding steps applied to the last virtual vertex, the execution of which leads to obtaining the final array for this level $\mathrm{Madj_}V^*_{1,5}$.

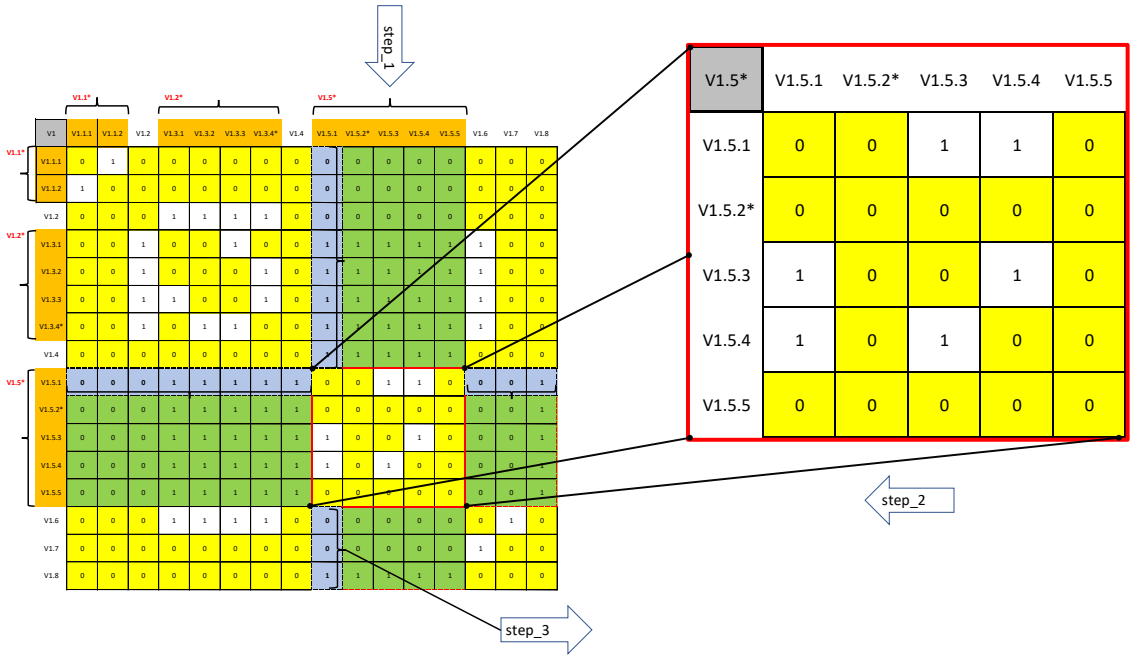


Figure 60 - Processing at virtual vertex level 3 $V^*_{1,5}$.

This process is repeated for each virtual vertex at each level of the multilevel hypergraph, until the final Madj array, corresponding to the highest level, is obtained. Following the same procedure for level 4 (Figure 61), we get the corresponding Madj (MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$)).

V1	V1.1.1	V1.1.2	V1.2	V1.3.1	V1.3.2	V1.3.3	V1.3.4.1	V1.3.4.2	V1.4	V1.5.1	V1.5.2.1	V1.5.2.2	V1.5.2.3	V1.5.3	V1.5.4	V1.5.5	V1.6	V1.7	V1.8
V1.1.1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.1.2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1.2	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
V1.3.1	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	0
V1.3.2	0	0	1	0	0	0	1	1	0	1	1	1	1	1	1	1	1	0	0
V1.3.3	0	0	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	0	0
V1.3.4.1	0	0	1	0	1	1	0	1	0	1	1	1	1	1	1	1	1	0	0
V1.3.4.2	0	0	1	0	1	1	1	0	0	1	1	1	1	1	1	1	1	0	0
V1.4	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0
V1.5.1	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	1
V1.5.2.1	0	0	0	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1
V1.5.2.2	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1
V1.5.2.3	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1
V1.5.3	0	0	0	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1
V1.5.4	0	0	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	1
V1.5.5	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1
V1.6	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0
V1.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
V1.8	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0

Figure 61 - Result of the Madj_MLHG in level 4.

Virtual vertices (V_{i,n^*}) are not included in the final array, Madj_MLHG, because they represent abstractions, virtual vertices, and not real entities. The main objective in modeling a given phenomenon is to support certain levels of abstraction, n, to promote relational enrichment among the entities involved. These main entities, which govern the phenomenon in question, are considered the real entities in the modeling.

In Figure 62, referring to the example being demonstrated MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$), the Madj_2 (level 2), Madj_3 (level 3) and Madj_4 (level 4) are presented, the last one coinciding with the adjacency matrix, Madj_MLHG. Also, the graphs associated with each of the matrices are shown at a superior. It should be noted that the size of the final adjacent matrix depends only on the number of effective vertices, i.e., it is independent of the number of virtual vertices admitted and the maximum level of abstraction at which the effective vertices are considered as entities. However, the same does not occur with the number of edges, that is, with the relationships between entities. As we advance in the levels of abstraction, the

relationships become more complex, resulting in a numerical densification of the adjacency matrix and an increase in the degree of the vertices, making the graph more connected. Despite this, it is possible to verify that the same pre-established subgraphs at lower levels maintain the partition structure.

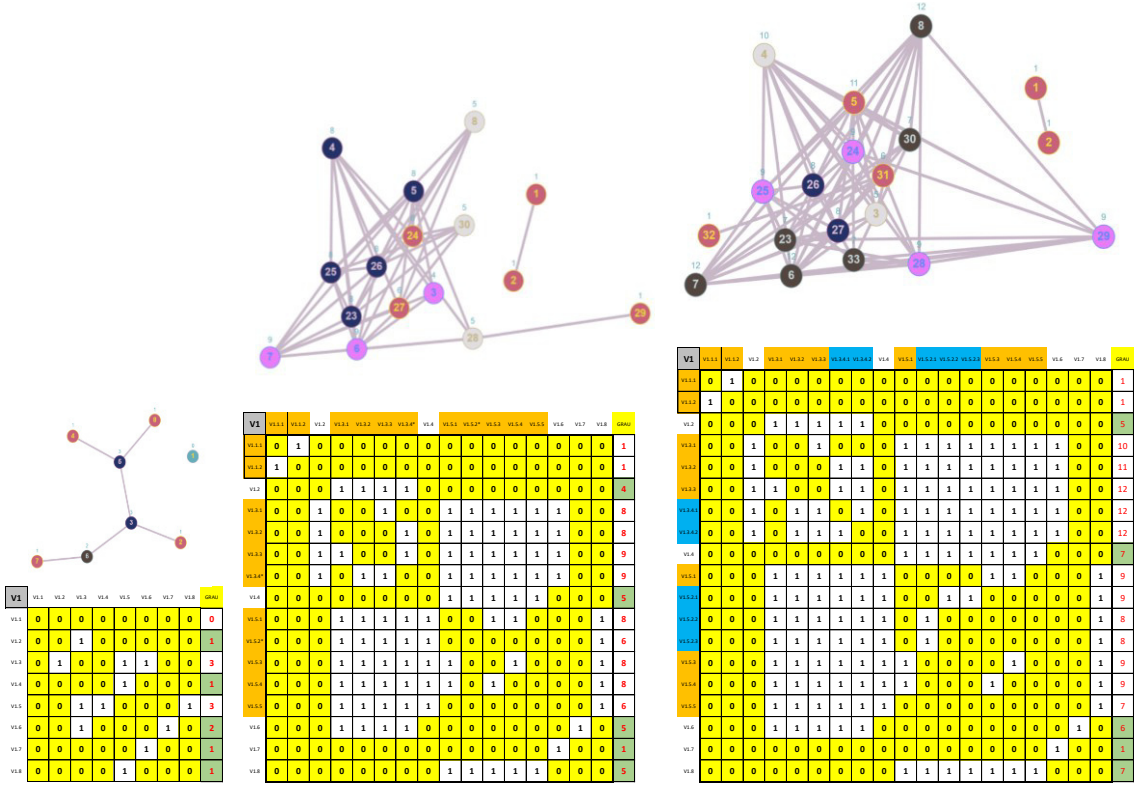


Figure 62 - Adjacency Matrices of level 2, 3 and 4 - MLHG ($V_{1,4}$, $A_{1,4}$, $V_{i,4}$*, $E_{i,4}$).

In the development of this work, the concept of the basic structure of an MLHG was identified. Given the importance of this concept in the application described in Chapter 5, it is considered appropriate to provide a brief.

An MLHG can be understood as a hierarchical composition formed by other MLHG of smaller structural scale. The size of each of these sub models depends directly on the number of virtual vertices included in its base structure, represented by the logical level tree. This tree may contain subtrees, which in turn integrate other subdivisions, reflecting the recursive and expansive nature of the MLHG.

Based on the example of the multilevel hypergraph presented in the Table 6, it is possible to observe, in the Figure 63, the existence of six subtrees, which represent different potential instances of derived MLHG. The base structure (EB) of a multilevel hypergraph thus manifests itself in the presence of these subtrees, which can be generated from a specific virtual vertex within its tree. In general, the

total number of possible subtrees corresponds to the number of virtual vertices identified in the upper levels, plus one unit.

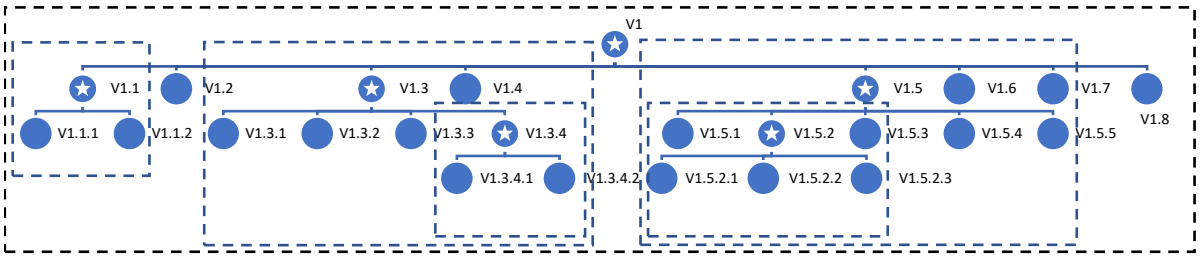


Figure 63 - Subtrees, $A_{I,n'}$ of MLHG ($V_{1.4}$, $A_{1.4}$, $V_{i.4}$*, $E_{i.4}$).

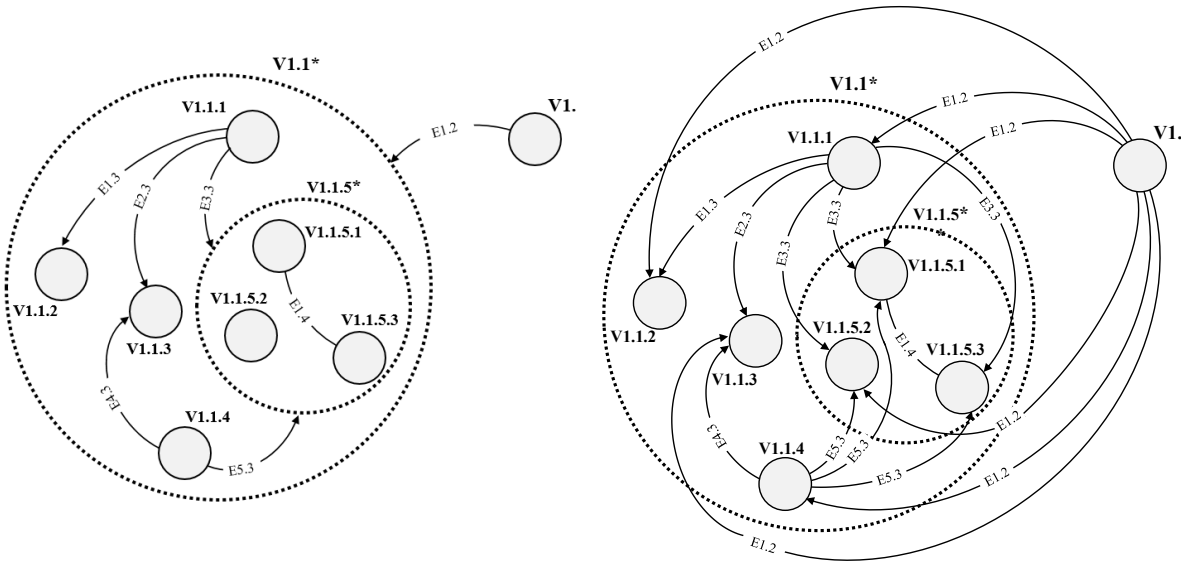
Therefore, we can identify an index capable of measuring the relative intensity of relationships in MLHG, IEB (MLHG) through expression (3).

$$\boldsymbol{IEB}\ (HGMN\ ) = \frac{\sum HGMN\ [V^*_{i,n}\ ]}{|V^*_{i,n}|} \qquad (3)$$

For the example presented in Figure 63, we have EB (MLHG [V_1*]) = 6; EB (MLHG [$V_{1.1}$*]) = 1; EB (MLHG [$V_{1.3}$*]) = 2; EB (MLHG [$V_{1.5}$*]) = 2; EB (MLHG [$V_{1.3.4}$*]) = 1; EB (MLHG [$V_{1.5.2}$*]) = 1, i.e., the basic structure of a multilevel hypergraph is the distribution of possible subtrees extracted from tree $A_{i,n}$. of the MLHG, when observed from a given virtual vertex Vi,n*, resulting in the base structure index value IEB(MLHG) = (6 + 1 + 2 + 2 + 1 + 1) / 6 = 2.16 in MLHG (V_{1}*).

4.2 Features and Utility of an MLHG

We can use MLHG to model real-world systems, representing data elements and relationships, using levels to define hierarchical dependencies, or to separate elements (aggregates) from different components.



a) MLHG regular schema b) MLHG expanded schema

Figure 64 - A generic model of a MLHG.

Let us now consider a small example of an MLHG. Figure 64a. It is a clear and useful example to understand how we can model a graph database using an MLHG, representing its data elements, their relationships, as well as how these elements can be organized at different levels of abstraction, through the establishment of hierarchies. The MLHG presented integrates several vertices that are organized into two virtual vertices, $V'_{1.1}$ e $V'_{1.1.5}$, which are used to represent hyper-edges.

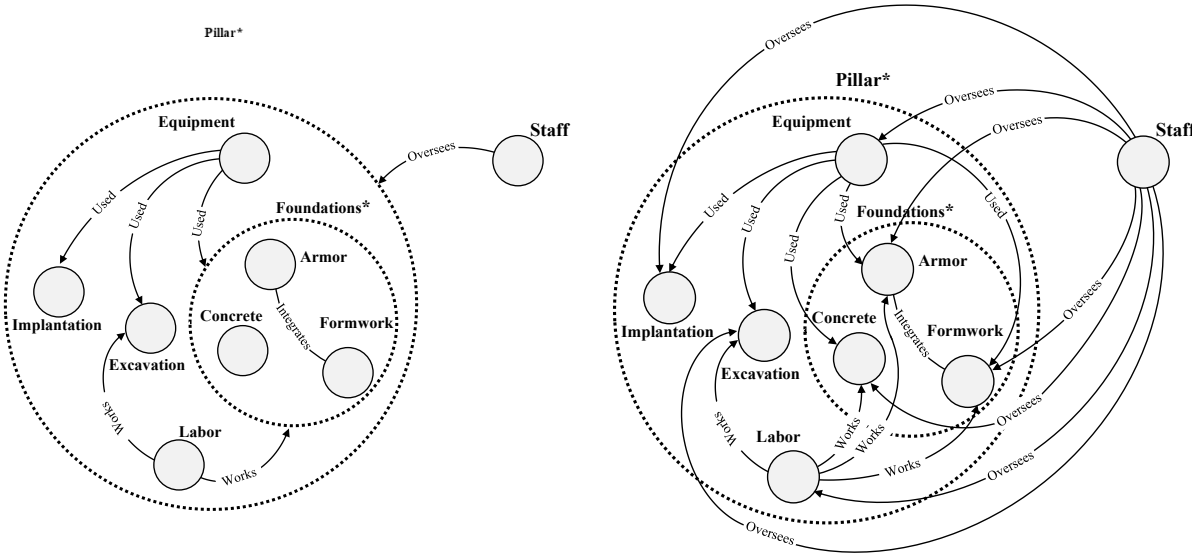
The relationship of virtual vertices to simple vertices contained at the same level of abstraction must consider what is shared with other vertices at other levels, which may be other hyperedges or singular vertices.

Table 8. Matrix of adjacency of the graph presented in Figure 64a.

	V1.2	**V'1.1**	**V1.1.1**	**V1.1.2**	**V1.1.3**	**V1.1.4**	**V'1.1.5**	**V1.1.5.1**	**V1.1.5.2**	**V1.1.5.3**
V1.2	0	1	0	0	0	0	0	0	0	0
V'1.1	0	0	0	0	0	0	0	0	0	0
V1.1.1	0	0	0	1	1	0	1	0	0	0
V1.1.2	0	0	0	0	0	0	0	0	0	0
V1.1.3	0	0	0	0	0	0	0	0	0	0
V1.1.4	0	0	0	0	1	0	1	0	0	0
V'1.1.5	0	0	0	0	0	0	0	0	0	0
V1.1.5.1	0	0	0	0	0	0	0	0	0	1
V1.1.5.2	0	0	0	0	0	0	0	0	0	0
V1.1.5.3	0	0	0	0	0	0	0	1	0	0

Table 8 presents the Matrix of adjacency of the MLHG, illustrated in Figure 64. This matrix consolidates the hierarchical abstraction of an MLHG, since it establishes the structure of its native graph, in which its relations are formally established. In this way, it is possible to set up a hierarchical data model, as shown in the Figure 64b, considering the three levels of abstraction.

Let us now move on to a concrete case of application of an MLHG, the modelling of a civil construction problem related to the execution of columns in a building. To do this, we'll use the graph presented earlier in Figure 64a, and we will instantiate it with the data elements we use to model the referred project. In practice, we replace abstract labels with the real terms used in the modeling of the project we carry out. The graph we obtained (Figure 65a) it gives us a concrete view of the real problem, including all the data elements used (''Staff'', ''Pillar'', ''Equipment'', ''Foundations''', etc.), as well as the relationships established between them.



a) MLHG regular schema

b) MLHG Expande Schema

Figure 65 - An MLHG model for a civil construction project.

In this way, we can see that "Staff" controls and supervises a set of processes (or entities) grouped into two hierarchical levels: a first contemplating the simple vertices "Equipment", "Implantation", "Excavation" and "Work", and a second integrating the virtual vertex "Foundations*". The latter allows the establishment of another hierarchical level, involving the elements "Concrete" and "Reinforcement", in which the latter, in turn, is related to "Formwork". Once all the relationships have been established (Figure 65b), the virtual vertices "Pillar*" and "Foundations*", lose their direct relations with "Personnel", "Equipment" and "Work", as they are execution processes. This is because your data elements are now directly related to your higher hierarchical elements after the expansion process. With this concrete example, we can see how we can design a model for a graph-supported database using an MLHG. In the next section, we will use this last graph to present, discuss, and demonstrate how traditional data manipulation operations can be performed on a concrete MLHG.

4.2.1 Fundamental Operations

The data structure of an MLHG is quite complex. Their manipulation requires a proper implementation of CRUD operations, so that they can work in the complex network of entities and relationships that integrate the various hierarchical levels of an MLHG. By providing CRUD operations capable of handling MLHG data structures, we are enabling systems to gain operational efficiency. In each specific instance of an

MLHG, at all levels of abstraction, it is imperative to perform a complete aggregation operation, consolidating the entire structure of the MLHG into a single global graph or into one of its intermediate stages of processing. In Figure 65a, we can see an example of a graph processed at a given level of abstraction, along three distinct steps of aggregation from that level to a particular instance. During the process of performing aggregation, not only the number of evident vertices increases, but also the established relationships, contributing to a densification of the global graph. This not only adds complexity but also reveals the most hidden interconnectedness between the elements of the MLHG model conceived (Figure 65b). This detailed analysis is important to understand the final structure of the graph and the dynamic evolution of the model, as new instances are incorporated, removed or modified by a CRUD operation. To implement CRUD operations, it is necessary to define how each operation affects the structure of an MLHG. CRUD operations can be characterized as follows:

- CREATE – this operation allows new vertices and relationships to be added to a graph, which means that new entities and relations can be introduced, expanding the representation of the model established for a given problem;
- READ – the read operation is responsible for obtaining a certain information from the graph, allowing the consultation of vertices and relationships according to the definition of a certain set of criteria;
- UPDATE – update allows existing data in a graph to be modified, including updating the properties of the vertices and the relationships established between them;
- DELETE – the remove operation eliminates vertices and relations from a graph; When deleting a node, regardless of its type (simple or virtual), all established relationships must be removed from the graph, even between different levels of abstraction.

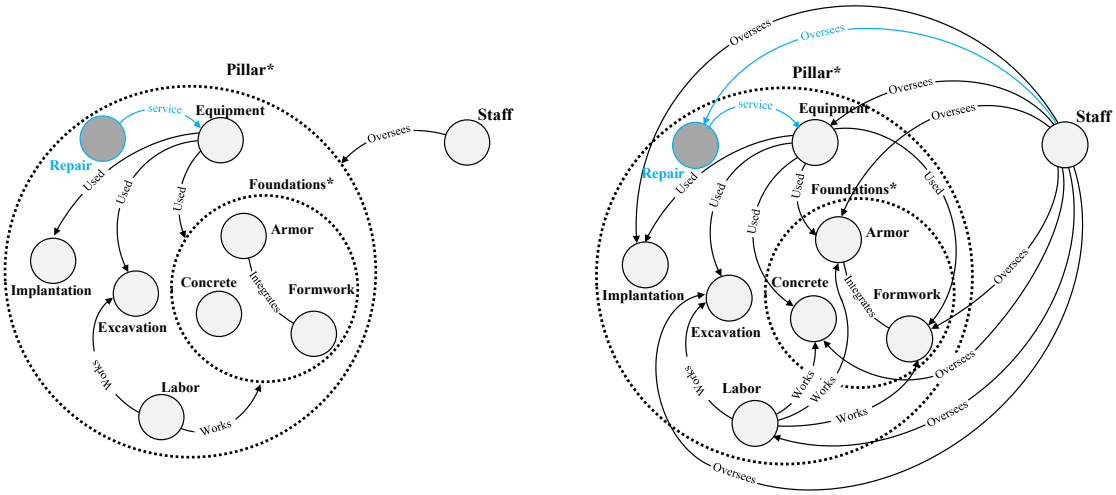
The adaptation of these operations to the context of the MLHG still requires some specific formulation capable of accommodating the nature of its structure and operationalization. During the next few sections, we formally introduce and describe all of these operations in the context of an MLHG.

The Create Operation

The CREATE operation plays a relevant role in instantiating the model structure of an MLHG. This operation goes beyond a simple insertion of new vertices or relations in a specific graphical tree (A_i), since it also requires the establishment and increment of a hierarchy during the instantiation process. Thus, when a new vertex ($V_{i,w}$) is created at an abstraction level w, it is necessary to define its properties ($P(V_{i,w})$) as well as its associated relations ($E(V_{i,n})$). However, creating a vertex in an MLHG involves constructing relationships between the existing vertices at the w level and the relationships with some possible virtual vertices ($V^*_{i,w}$), which are projected to the subsequent level w+1. To formalize this interconnected structure, we must consider the subgraphs SGw+1, as they represent the relations of the w-level. Formally, we can define the create operation using four variants, normally to create one:

- *Simple vertex* – create($V_{i,w}$) = (($P(V_{i,w})$, $E(V_{i,w})$). In this operation it is necessary to identify the properties P(Vi, w) of the vertex, as well as to define its various relations $E(V_{i,w})$, at the level of the abstraction w;
- *Tree* – create ($A_{i-j,w}$) = (V($V_{i,w}$, $V^*_{j,w}$), E($V_{i,w}$, $V^*_{j,w}$)). To create an instance of a graph tree ($A_{i-j,w}$) it is necessary to establish a hierarchy for all vertices (V), virtual ($V^*_{j,w}$) or non-virtual ($V_{i,w}$), as well as to define their properties, in a similar way to the previous creation variant, as well as the relationships (E) that can be established between the various sets of vertices (V). All relationships will be established by inheritance;
- *Virtual vertex* – create ($V^*_{i,w}$) = ($SG_{i,w}$ (V($V_{i-k,w}$, $V^*_{i-j,w}$), E($V_{k,w}$, $V^*_{j,w}$)), where $SG_{i,w}$ corresponds to a subgraph i of level w. The creation of a virtual vertex ($V^*_{i,w}$) is the creation of a subgraph $SG_{i,w}$, which represents the virtual vertex. This operation requires the creation of all vertices V ($V_{i-k,w}$, $V^*_{i-j,w}$) contained at the abstraction level w+1, which is subsequent in the hierarchy, as well as the creation of their relations E($V_{k,w}$, $V^*_{j,w}$) at the level w. The creation operation involves establishing the structure of the subgraph through its adjacency matrix ($SG_{i,w}$);
- *Relationship* – create ($E_{i,w}$) = (E($V_{i-j,w}$, $V^*_{j,w-1}$), {$SG_{j,w}$, $SG_{j+1,w-1}$ ... $SG_{j,1}$}). The creation of a relationship ($E_{i,w}$) requires prior knowledge and the ability to explore the entire hierarchical structure of MLHG, through the use of the read operation. In this way, the knowledge, at least up to level w, substantiated by the levels below that level, as well as the knowledge of the registration of all the subgraphs of the SG crossed by the hierarchical structure up to level 1, is incorporated.

The vertex or relationship creation operation on an MLHG can be performed at the schema level, at the database level, or at the level of a specific database instance. For example, in the Figure 66a, the vertex "Repair" was created at the abstraction level n=2, within the virtual vertex "Pillar*", and a "service" relationship was established with the "Equipment" vertex. After graph expansion (Figure 66b), considering the MLHG inheritance mechanisms, another "supervises" relationship is created with the "Staff" vertex. In the creation of the "Repair" vertex, it became clear that one of the first prerequisites of the operation is the identification of the level of abstraction at which the vertex will be defined. In this case, at the second level of abstraction, n=2. At this level, it is also necessary to identify the corresponding virtual vertex, on which the new node will be created, which will be the virtual vertex '*Pillar*' (Figure 66a).



a) Initial configuration of MLHG after the create operation.

b) The MLHG expanded after the create operation.

Figure 66 - Creating a simple vertex.

In other words, if the creation is done at the first level of abstraction, n=1, that level will also be inappropriately taken as a virtual vertex. If the vertex to be created is a virtual vertex, its implementation can result in an increase in MLHG abstraction levels, especially when the level identified for the implementation of the new vertex corresponds to the highest level of abstraction existing in MLHG at that time. In this case, the introduction of this new virtual vertex would imply an increase in the maximum number of abstraction levels, thus expanding the MLHG hierarchy itself. The create operation is highly oriented, allowing vertices and relationships to be added in a specific and controlled way in different instances of the database. This makes it possible to adapt to the needs of each context, such as civil construction, providing flexibility to modeling and ensuring that the database structure evolves according to the guidelines of the base manager and their specific requirements.

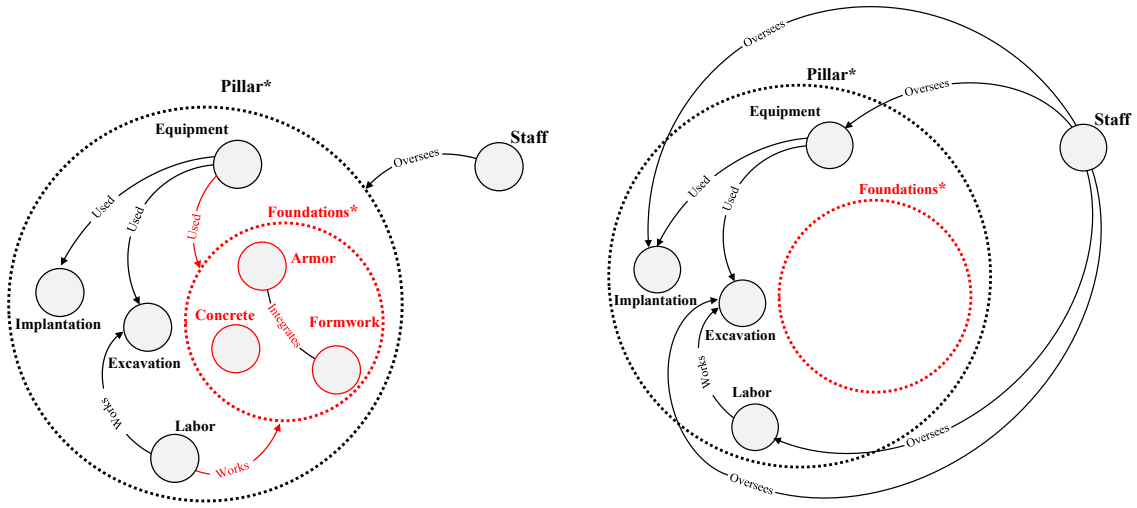
The Delete operation

Performing a removal operation to eliminate elements of a graph is always quite delicate and requires some care, especially when the structure involved has the characteristics of an MLHG. Thus, to perform an operation to delete one or more elements of an MLHG, it is important to analyze all its dependencies with respect to other elements of the graph. Thus, the removal of a virtual or simple vertex $V_{i,w}$, or a relation, in each instance of an MLHG is not a trivial operation, since it requires a careful analysis of its four components ($V_{i,n}$, $A_{i,n}$, $V^{*}_{i,n}$, $E_{i,n}$), as with the other operations already discussed. However, the delete operation requires greater attention to the dependencies established in the respective hierarchical structure ($A_{i,w}$), and presented during the transversality of the graph instance, at all levels compromised by the hierarchical dependencies established from level 1 to level w. Removing a vertex, simple or virtual, or a relationship implies updating and structurally adjusting all relationships up to level w+1, as well as updating the inherited relationships of the crossed virtual vertices at the different levels of abstraction below w. Formally, we can specify a delete operation in an MLHG taking into account three variants, i.e. delete one:

- *Simple vertex* – exclude ($V_{i,w}$) = ($SG_{i,w-1}$(V($V_{i-w,w}$, $V^{*}_{i-w,w}$), E($V_{k,w}$,$V^{*}_{j,w-1}$)), $A_{i,w}$). In this case, the exclusion of a simple vertex ($V_{i,w}$), belonging to a given level (w), infers the new morphological structures resulting for the subgraphs (SGj,w-1), taking into account the new subset V_{i-w} of simple vertices and the potential relations (E), dictated or not, by the RI of existing virtual vertices in Ai, w, which results from the removal of the simple vertex itself ($V_{i, w}$);
- *Virtual vertex* – delete ($V^{*}_{i,w}$) = ($SG_{j,w-1}$(V($V_{i-w,w}$, $V^{*}_{i-w, w}$), E($V_{k,w,}V^{*}_{j,w-1}$)), $A_{i,w}$). Similarly, when removing a virtual vertex ($V^{*}_{i,w}$), belonging to a given level w, the new morphological structures resulting from the subgraphs ($SG_{i,w-1}$) are inferred, taking into account the new subset of virtual vertices (V^{*}_{i-w}), and the potential relations (E), also dictated, or not, by the RI of the existing virtual vertices in Ai, w, resulting from the virtual vertex deletion operation itself ($V^{*}_{i,w}$);
- *Relationship* – exclude ($E_{i,w}$) = ($SG_{j,w-1}$(V($V_{i-w,w}$, $V^{*}_{i-w,w}$), E($V_{k,w,}V^{*}_{j,w-1}$)), $A_{i,w}$). On the other hand, when excluding a relationship ($E_{i,w}$), belonging to a given level (w), the new morphological structures resulting from the subgraphs ($SG_{j,w-1}$) are also inferred, which are implied by the set of vertices V ($V_{i-w,w}$, $V^{*}_{i-w,w)}$. In this case, the set of vertices is immutable due to the operation being of exclusion of a given relationship and the set of relationships (E) resulting from the deletion itself, and dictated or not, by the RI of virtual vertices existing in $A_{i,w}$.

In these variants of the remove operation, $SG_{i,w-1}$ represents an open set of all existing subgraphs, from level 1 to level w, while Ai,w represents the tree structure associated with the vertex $V_{i,w}$ or the relation $E_{i,w}$, which have been removed. In the most trivial case of deleting a relationship (E_w) between two vertices, at a given level w, the operation updates the corresponding value in the adjacency matrix $Mad_{j,w}$ representative of $SG_{j,w}$ to {0}, with the appropriate aggregation of all subgraphs of the instance in question, in order to update the instantiated structure. Thus, it is necessary that the delete operation covers all subgraph structures ($SG_{j,1}$ ... $SG_{j,w}$) of the transverse made, and corrects its corresponding adjacency matrices ($Mad_{j,1}$ ... $Mad_{j,w}$) to update the MLHG instance. The delete operation requires updating all relationships inherited by the virtual vertices Vi,n+1 that had relations with $V_{i,n}$. This update is necessary to ensure that the virtual vertices maintain their correct and up-to-date relationships, and preserve the integrity of the information maintained along the abstraction levels, and the integrity of the multilevel hierarchical structure $A_{i,w}$.

Now, let's see how the operation of removing a vertex works and its relations. Figure 67a and Figure 67b illustrates the removal of the virtual vertex "Foundations*", at the n=2 abstraction level of the graph, which also implies the removal of two relations associated with it, i.e., "Used" and "Works". After graph expansion (Figure 67b), the "supervises" relationship, originating from the "Staff" vertex, is established with only four other vertices ("Deployment", "Equipment", "Work" and "Excavation"). This eliminates the relationship of this type for the number of vertices that originally made up the virtual vertex "Foundations*", target of the removal operation.



(a) the MLHG before the delete operation.

b) The expanded MLHG after the delete operation.

Figure 67 - Excluding the virtual vertex "Foundations*".

The update operation

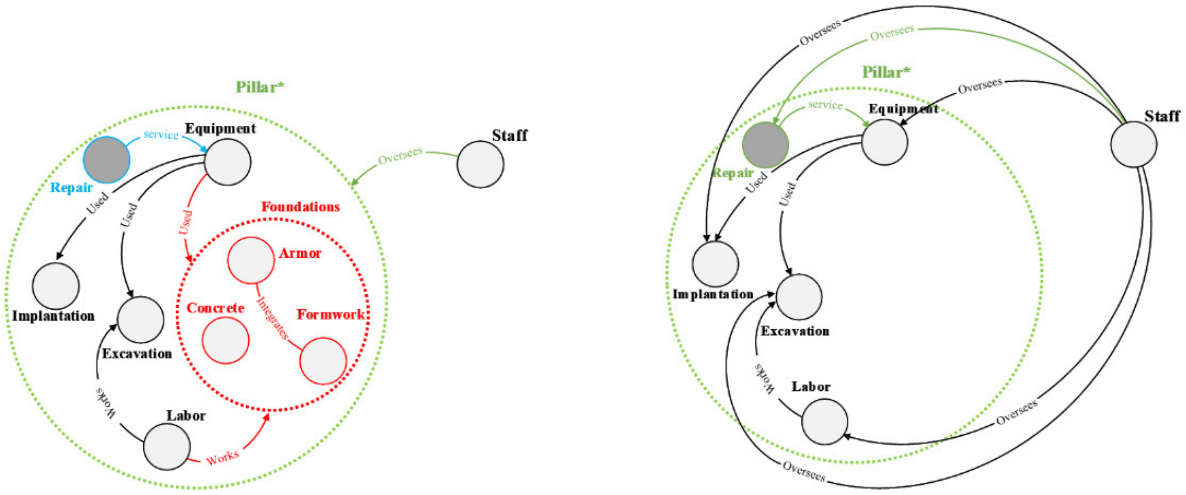
The update operation is used to modify properties and relationships inherent to a given vertex of an MLHG. In addition to allowing, you to make changes directly to the data of a $V^*_{i,n}$ vertex, the update operation also allows you to adjust and manipulate the data structures present in a $V^*_{i,n}$ virtual vertex, without compromising the consistency and coherence of the global model. As with other data systems, the update operation is particularly useful for making one-off modifications, reorganizing information, or optimizing aspects of an MLHG model representation. To make changes to a vertex ($V_{i,n}$), as well as to its relations ($E_{i,n}$), an update operation must adjust and manipulate the data structures present in each virtual vertex ($V^*_{i,n}$), or even in its own graphical tree ($A_{i-j,n}$). In addition, after the changes made, the hierarchy between the levels of abstraction i-j must also be reorganized. Formally, we can define an MLHG update operation considering its fundamental components ($V_{i,n}$, $A_{i,n}$, $V^*_{i,n}$, $E_{i,n}$), considering four update variants, namely for updating of:

- *Simplex vertex* – update($V_{i,n}$) = (($P(V_{i,n})$, $E(V_{i,n})$). In this variant, the properties $P(V_{i,n})$ of a vertex Vi,n are updated, as well as its relations $E(V_{i,n})$ established at the level of abstraction n. This is the simplest update variant, since we only need to visit the vertex and directly update its properties P(Vi, n), as well as its relations $E(V_{i,n})$, as if it were a property graph;
- *Tree*, between two levels – update($A_{i-j,n}$) = ($V(V_{i,n}$, $V^*_{j,n})$, $E(V_{i,n}$, $V^*_{j, n})$). By updating the structure of a graph tree ($A_{i-j,n}$), it is possible to update the hierarchy for all vertices (V) – virtual ($V^*_{j,n}$) or non-virtual ($V_{i,n}$) – contained in the tree and its properties, in a similar way to the first variant presented, and their relationships (E) existing between these sets of vertices. In the latter case, it is necessary to re-establish the relationships, by inheritance, for each virtual vertex visited;
- *Virtual vertex* – update($V^*_{i,n}$) = $SG_{i,n}(V(V_{i-k,n}$, $V^*_{i-j,n})$, $E(V_{k,n}$, $V^*_{i,n}))$. The update of a subgraph i of the n level ($SG_{i,n}$), which represents a virtual vertex $V^*_{i,n}$ requires the indication of the set of vertices $V(V_{i-k,n}$, $V^*_{i-j, n})$ contained in the abstract level n+1, subsequent in the hierarchy, and its relations E ($V_{k,n,}V^*_{j,n}$) that belong to the level n;
- *Relation* – update($E_{i,n}$) = ($E(V_{i-j,n}$,$V^*_{j,n-1})$,{$SG_{j,n}$, SG_{j+1}, SG_{n-1}... $SG_{j,1}$}). To update a relation ($E_{i,n}$), it is necessary to know how to explore the hierarchical structure of an MLHG, at least up to level n, based on all levels below it, as well as to know the record of all subgraphs crossed by the hierarchical structure up to level 1.

The update operation acts beyond the domain of a single vertex as it can adjust data residing in virtual vertices ($V'_{i,n}$), which allows users to specifically manage the data structure that is distributed across the various MLHG segments. This feature allows for very precise update operations on MLHG structures, according to a precise set of requirements, without introducing unwanted discontinuities or inconsistencies into your overall structure. This operation is a valuable tool in problem modeling, especially when dealing with dynamic contexts, in which the flexibility to adjust data models is essential to receive any variations of the model over time, according to the management requirements of the system.

Updating vertices or relationships in an MLHG goes beyond the modification of its properties, since it can contemplate, in an integrated way, the two operations previously presented, create and remove. Thus, we can represent an update operation as shown in the Figure 68a and Figure 68b, using the two operations mentioned. Thus, in the update operation, the structural modification of the MLHG is carried out by creating new vertices and removing existing vertices, while changing the relationships. In Figure 68a the vertex "Repair" is created at the level of abstraction n=2, within the virtual vertex "Pillar'". This creation gives rise to a new "Service" relationship with the "Equipment" vertex. Simultaneously, another vertex, such as "Foundation'", can be deleted, which would also result in the exclusion of its incidental relationships, "Used" and "Works", respectively. This type of operation allows MLHG to adapt to new data needs, according to possible changes in context or operational needs, for example.

In Figure 68b we can see the reconfiguration of MLHG after performing an update operation. The "Repair" vertex also adds a new "Supervises" relationship with the "Personal" vertex, acquired through the MLHG concept of RI, while the exclusion of the virtual "Foundation" vertex'" results from the removal of the "Used" and "Works" relationships with the "Equipment" and "Work" vertices, respectively. This restructuring reveals how the update can manipulate the graph as needed, allowing its structure to be adjusted in a regulated way, to better respond to the hierarchy and relationships required in dynamic data environments, such as construction.



a) Creating a vertex and removing a virtual vertex. b) Expanded MLHG after removing a virtual vertex.

Figure 68 - Updating the structure of an MLHG.

The Read Operation

The read operation allows access to the various data elements of an MLHG, in a generalized or specific way. This operation can be performed in a classical way, in all or part of the graph structure. Within a range bounded by two levels of abstraction (N_i and N_j, with $N_i<N_j$), even if not consecutive, it is possible to access a set of vertices $V_{k\#}$, virtual vertices $V^*_{k\#}$ or $E_{k\#}$ relationships, through a traversal structured by the subtree $A_{k,Hj}$. The tree structure is captured in the instance hierarchy, when the read operation itself is executed, crossing only the part of the graph structure defined for read. Formally, we can define the read operation considering its fundamental components ($V_{i,n}$, $A_{i,n}$, $V^*_{i,n}$, $E_{i,n}$) and considering three variants of reading, namely for reading a set of:

- *Simple vertex* – read($V_{k\#}$, (N_i, N_j)) = (P($V_{k,Hj}$), P(Ek, $_{Hj}$), $A_{k,Hj}$). In an operation of reading a set of vertices $V_{k\#}$, belonging to an interval of two levels of abstraction, Ni and N_j, a subtree of graphs, $A_{k,Hj}$, is inferred, allowing to identify the properties P($V_{k,Hj}$) of the entire set of vertices V_k, between the two levels mentioned, as well as capturing the properties P($E_{k,Hj}$) of the whole set of relations E_k;
- *Virtual vertex* – read($V^*_{i\#}$, N_i, N_j) = (P($V^*_{k,Hj}$), P($E^*_{k,Hj}$), $A_{k,Hj}$). The operation of reading a set of virtual vertices $V^*_{k\#}$, belonging to a range of two levels of abstraction, N_i and N_j, makes the inference of the nodes of a subtree of graphs, $A_{k,Hj}$, which allows the identification of the properties P($V^*_{k,Hj}$) of the entire set of virtual vertices V^*_k, between the two levels under analysis, as well as captures the properties P($E^*_{k,Hj}$) of the entire set of relationships of the virtual vertices, E^*_k;

- *Relationship* – read(($E_{w\#}$), N_i, N_j) = (P($V_{k,Hj}$), P($V_{k^*,Hj}$), $A_{k,Hj}$). In an operation of reading a set of relations $E_{k\#}$, belonging to a range of two levels of abstraction, Ni and N_j, the structure of a subtree of graphs, $A_{k,Hj}$, is inferred, which allows the identification of the properties P($V_{k,Hj}$) of the entire set of vertices V_k, between the two levels under analysis, as well as captures the properties P($V^*_{k,Hj}$) of the entire set of virtual vertices V^*_k.

To perform a read operation directly, i.e. without considering its various components, the hierarchical structure of MLHG, it is necessary to previously create a resulting graph (G_r). This graph is constructed through a process of aggregation of all subgraphs $SG_{k,Hj}$, marked as nodes in the connected paths of subtree $A_{k,Hj}$.

4.2.2 Implementing MLHG functions using Python

As part of the practical exploration of the MLHG model, a library of applications in Python language was designed with the aim of enabling the exploration of MLHG and demonstrating its practical application in several case studies. The library incorporates a set of functions capable of processing and analyzing the hierarchical structures defined by the representative labels of an MLHG, as it was formally defined previously. These functions provide a more comprehensive manipulation and view of the relationships between the constituent elements. In Table 9 presents the 16 main functions that were developed are presented.

Table 9 – Functions that process and analyze the structure of an MLHG.

Ref.	**Function**
1.	*get_parents_by_level (labels)*
2.	*get_nodes_by_level (labels)*
3.	*get_nodes_of_level (level, nodes_by_level)*
4.	*create_graph_for_level (level, nodes_by_level, rakes)*
5.	*plot_parent_levels (labels)*
6.	*get_subtrees (parents_by_level)*
7.	*get_subtree_dict (labels)*
8.	*plot_subtrees (labels)*
9.	*get_immediate_upper_nodes (trees)*
10.	*generate_adjacency_matrix_and_dict (edge_list, node_list)*
11.	*build_matrix_with_dict (A, B, virtual_node, dict_nodes_labelsA, dict_nodes_labelsB)*
12.	*create_nodes_and_relationships(...)*
13.	*query_nodes_and_relationships (...)*
14.	*get_descendants_by_level (parents_by_level)*
15.	*get_ancestors_by_level (parents_by_level)*
16.	*get_children_by_level (parents_by_level)*

The functions developed allow the treatment and organization of the MLHG's hierarchical information in a direct and efficient way. Each of them is designed to support the processing, analysis, and storage of the data that make up the structure of the model. In a practical way, these functions can

be used to analyze data with various levels of hierarchy, to instantiate models in graph-oriented databases, or even to represent systems with multiple relationships between their elements. Its use makes it easier to read the existing interconnections and makes it easier to identify the hierarchical dependencies between the vertices. Integrating these functions into an application enables a better understanding of the behavior of the hierarchical structure and its internal organization. In the following paragraphs, some examples and a detailed description of each function are presented, showing how they can be applied and what their contribution is to the interpretation of the data.

Let's start with the *get_parents_by_level* function (labels). This function receives as input a list of labels that describe the hierarchy of an MLHG.

Taking as an example: MLHG = ["1", "1.1", "1.2", "1.2.1", "1.3", "1.4", "1.3.1", "1.1.1", "1.1.2","1.1.3","1.1.4","1.1.5","1.1.6","1.1.6.1", "1.2.1.2", "1.2.1.2.1"], the function goes through each label and counts the number of existing points, thus identifying the hierarchical level to which it belongs. Then, it extracts the parent label from each element and builds a dictionary where each level is associated with the list of its direct parents. The result obtained allows you to immediately visualize the hierarchical structure and understand how the elements are organized among themselves. For example, the dictionary:

parents_by_level : {0: [], 1: ['1'], 2: ['1.2', '1.3', '1.1'], 3: ['1.1.6', '1.2.1'], 4: ['1.2.1.2']}, represents the parents of each level according to the labels provided.

The *plot_parent_levels function* (labels) was developed with the aim of validating and understanding in a more intuitive way the hierarchical structure of a given MLHG.

This function generates the visualization of the hierarchy through a directed graph, in which the labels of the nodes and the hierarchical level associated with each parent node are presented. The Figure 69 shows the complete hierarchical structure of the labels, represented in the form of a digraph, allowing a clear observation of the relationships of dependence between the different levels.

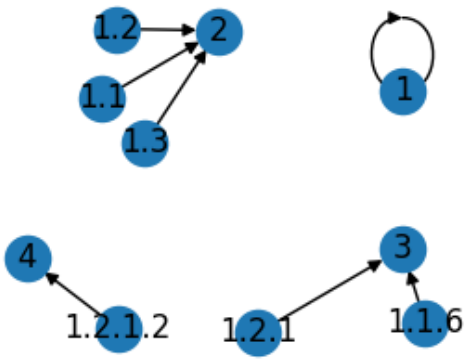


Figure 69 - Node labels pointing to the level of each parent.

Let us now analyze the function *get_subtrees* (parents_by_level). From the parent's dictionary by level obtained with the previous functions, this function constructs all the subtrees contained in the MLHG. Each subtree is represented by a list of labels that starts with a parent and includes its child nodes. The function groups label into specific subtrees and allow you to perform analysis and operations on each subtree separately. These subtrees are stored in a list and returned as a result. For the example considered, with this function we get the following subtrees:

[['1', '1.1', '1.2', '1.1', '1.1', '1.1', '1.1.', '1.1.', '1.1.', '1.1.', '1.1', '1.1', '1.1', '1.1

This functionality is useful for grouping labels into specific subtrees, allowing for more focused analysis and targeted operations to be performed on each subtree separately. Subtrees can be used for more detailed analysis or to perform specific manipulation operations within the context of MLHG.

Another function that was created to handle an MLHG was the *get_subtree_dict* function. This function takes a list of labels as input and returns a dictionary that maps each level to a list of subtrees. To get this mapping, this function uses the *get_parents_by_level* functions to get the parents by level and the get_subtrees function (parents_by_level) to get the subtrees. It then sequentially processes each subtree, returning the dictionary get_subtrees – see the current example: subtrees_by_level:

[(0, [['1', '1.1', '1.2', '1.2.1', '1.3', '1.4', '1.1.1', '1.4', '1.1.1', '1.1.4', '1.4', '1.4', '

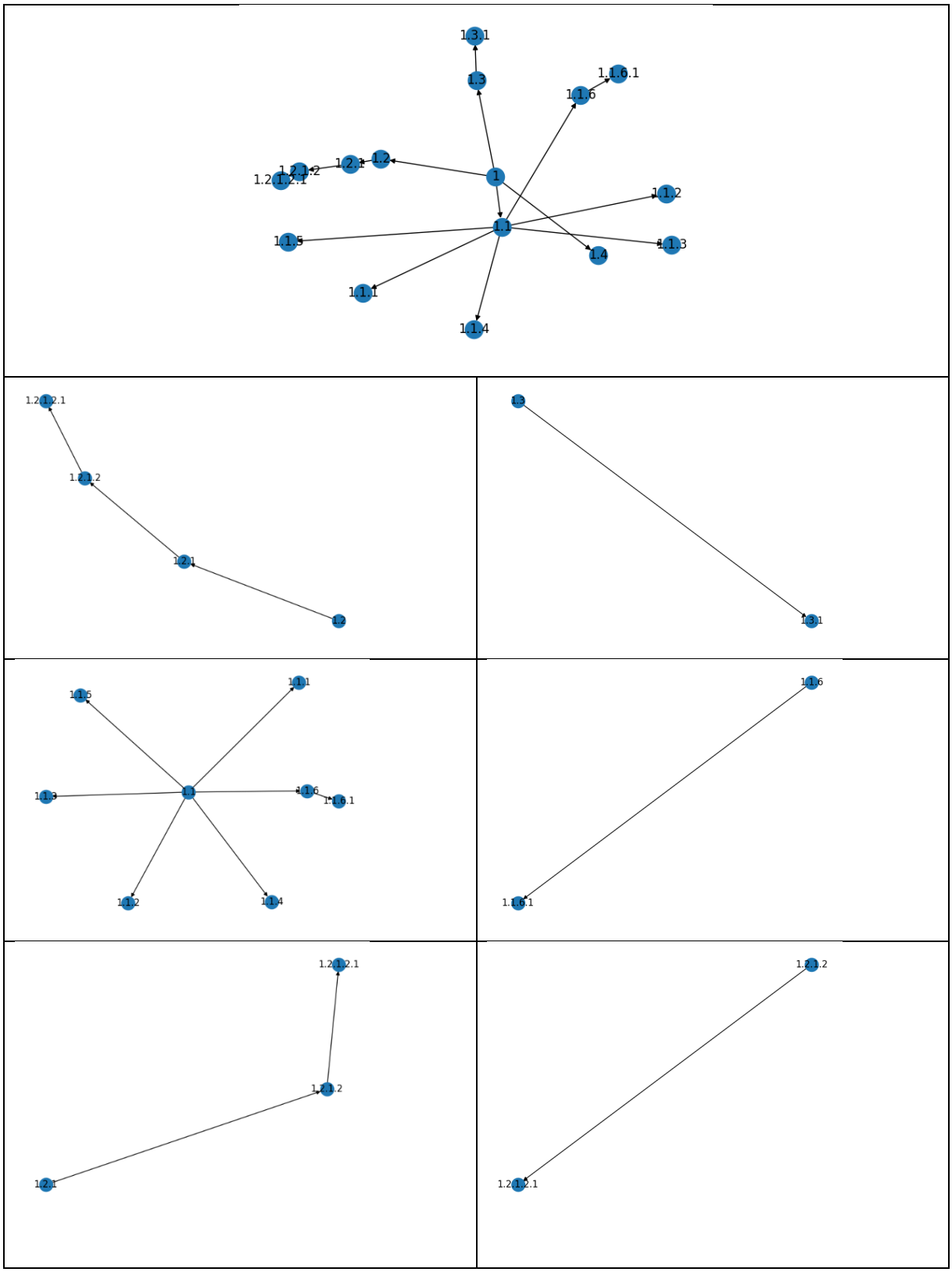


Figure 70 - Visualization of the different subtrees present in the labels of an MLHG

Another function that must be handled appropriately is the *get_nodes_of_level* function. This function accepts a list of labels as input and returns a dictionary that maps each level to a list of nodes. To do this, the function calls the *plot_parent_levels* function to collect the parents by level, and then sequentially processes each label to label them according to level. The label designations always start with the same prefix, after which they come (periods) to indicate sublevels. Thus, one can use the number of (points) that come after the prefix to identify the level of a node. This function simplifies the way in which data is analyzed and manipulated at one level. For example, when applying the function to the given labels, we get the output *nodes_by_level:*

{0: ['1'], 1: ['1.1', '1.2', '1.3', '1.4'], 2: ['1.2.1', '1.3.1', '1.1.1', '1.1.2', '1.1.3', '1.1.4', '1.1.5', '1.1.6'], 3: ['1.1.6.1', '1.2.1.2'], 4: ['1.2.1.2.1']}.

This represents the nodes grouped by level based on the labels provided. In this follow-up, the *get_nodes_of_level* function *(level, nodes_by_level)* was also created as a complement. This function takes a level and the dictionary *nodes_by_level* as input and returns a list of nodes corresponding to the given level. The function checks to see if the level is present in the dictionary and, if it is, returns the list of nodes for that level. Otherwise, returns an empty list. This function allows you to get the nodes of a specific level to perform targeted operations or analysis. For example, when using the function *get_nodes_of_level* (2, nodes_by_level), we get nodes_by_level_2:

['1.2.1', '1.3.1', '1.1.1', '1.1.2', '1.1.3', '1.1.4', '1.1.5', '1.1.6'].

To create a graph for a given level, we have the function *create_graph_for_level*. This function receives the level identifier as well as the structures nodes_by_level. A list of edges is defined. Then, the edges of the edges object, which were connected to nodes in the previous step, are added to the graph. After that, the resulting graph is returned. The function *create_graph_for_level* It is useful for graphically visualizing the structure of the graph at a specific level, which allows a better understanding of the connectivity of nodes at that level. For example, in the Figure 71 we can see level 2 graph. This graph was created based on the following input arguments: *nodes_by_level_2*:

['1.2.1', '1.3.1', '1.1.1', '1.1.2', '1.1.3', '1.1.4', '1.1.5', '1.1.6'] and arestas_2: [('1.1.1', '1.1.2'), ('1.1.2', '1.1.3'), ('1.1.4', '1.1.3'), ('1.1.4', '1.1.2')].

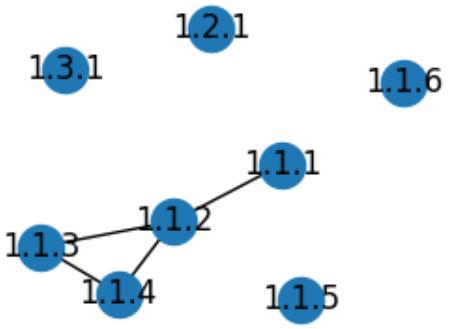


Figure 71 - Level 2 graph.

Let us continue the presentation of the functions developed by analyzing the implementation of the functions that are responsible for formalizing the composition of the MLHG. The first of these functions is the *generate_adjacency_matrix_and_dict* function. This function is responsible for representing the relationship between nodes through an adjacency matrix. The function inputs are two lists: one with the edges and one with the nodes. As an output, a dictionary called *dict_nodes_labels* is generated, in which each key corresponds to the label of a node and its value represents the index assigned to it in the array created by the function. The adjacency matrix and the node labeling dictionary constitute the main outputs of the process. For example, when applying this function with the input arguments: nodes_of_level_2: ['1.2.1', '1.3.1', '1.1.1', '1.1.2', '1.1.3', '1.1.4', '1.1.5', '1.1.6'], and aresta_2, [('1.1.1', '1.1.2'), ('1.1.2', '1.1.3'), ('1.1.4', '1.1.3'), ('1.1.4', '1.1.2')], the function will return the adjacency array and the mapping dictionary of the nodes. These two elements are presented in the Figure 50 and Figure 72.

Matriz_Adj2:

[[0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 1, 1, 0, 0, 0, 0],
[0, 1, 0, 1, 0, 0, 0, 0],
[0, 1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0]]

Dic_2: {0: '1.1.1', 1: '1.1.2', 2: '1.1.3', 3: '1.1.4', 4: '1.1.5', 5: '1.1.6', 6: '1.2.1', 7: '1.3.1'}

Figure 72 – Example for the main outputs of the *generate_adjacency_matrix_and_dict* function.

The *get_immediate_upper_nodes* function *(trees)* plays an important role in the hierarchical composition of the MLHG. It receives as input a list of trees and returns a dictionary where each node is associated with its next top node within the structure. The process is simple: the function goes through each subtree, identifies the highest node in each of them, and records that information in a set of dictionaries. Thus, each subtree has its own mapping, which relates its main node to the corresponding upper nodes. This function is especially useful for detecting direct precedence relationships between nodes, allowing you to clearly understand which elements are located immediately above a given node. By applying the function to the trees provided, a hierarchical dictionary is obtained, which shows, in a structured way, the connection between each node and its top node.

parents_by_level: {0: [], 1: ['1'], 2: ['1.2', '1.3', '1.1'], 3: ['1.1.6', '1.2.1'], 4: ['1.2.1.2']}

subtrees: [['1', '1.1', '1.2', '1.2.1', '1.3', '1.4', '1.3.1', '1.1.1', '1.1.2', '1.1.1', '1.1', '1.1', '

by inheritance between the levels of the MLHG. This function receives as input two adjacency matrices, A and B, a virtual node *(virtual_node)* and two node mapping dictionaries for labels *(dict_nodes_labels_A and dict_nodes_labels_B),* to establish a new expanded matrix *(AExp),* combining matrices A and B according to the virtual node and label dictionaries. In a first step, the function creates an AA matrix with the adjusted dimensions, shifting the columns from A to accommodate the matrix B. In the next step, the AAA matrix is created with the correct dimensions, shifting the AA rows to accommodate the matrix B. Then, in a third step, the *AExp* matrix is created, copying AAA and replacing the virtual node with the corresponding labels of B. Finally, In the last step, the *dict_nodes_labels_AExp* dictionaries are created, adjusting the labels of the nodes according to the expansion performed. The resulting *AExp* array and mapping dictionary of the expanded nodes are returned results by the function. This function is essential to perform the operations of expanding and replacing nodes in the adjacency matrix, configuring the essential support for the existence of any MLHG. A practical example of obtaining the AExp matrix and its respective dictionary dict_nodes_labels_AExp can be analyzed in the Table 10.

Table 10 - Obtaining a practical result from the build_matrix_with_dict function.

Entrance			**Result**	
Matrix A: [[0 1 0 0 0 0 0 0] [1 0 0 1 0 0 0 1] [0 0 0 0 0 0 0 0] [0 1 0 0 0 0 0 0] [0 0 0 0 0 1 0 0] [0 0 0 0 1 0 0 0] [0 0 0 0 0 0 0 0] [0 1 0 0 0 0 0 0]]	Matrix B: [[**X Y**] [**W K**]]	dict_nodes_labels_A: {0: '1.1.1', 1: '1.1.2', 2: '1.1.3', 3: '1.1.4', 4: '1.1.5', 5: '1.1.6', 6: '1.2.1', 7: '1.3.1'}	AExp Matrix: [[0 1 0 0 0 0 0 0] [1 0 0 1 0 0 0 1] [0 0 0 0 0 0 0 0] [0 1 0 0 0 0 0 0] [0 0 0 0 **X Y** 1 0 0] [0 0 0 **0 W K** 1 0 0] [0 0 0 0 1 1 0 0 0] [0 0 0 0 0 0 0 0 0] [0 1 0 0 0 0 0 0 0]]	dict_nodes_labels_AExp: {0: '1.1.1', 1: '1.1.2', 2: '1.1.3', 3: '1.1.4', 4: '1.1.6.1', 5: '1.2.1.2', 6: '1.1.6', 7: '1.2.1', 8: '1.3.1'}
	virtual_node: '1.1.5'	dict_nodes_labels_B: {0: '1.1.6.1', 1: '1.2.1.2'}		

Subsequently, to the process of creating adjacency matrices and their respective label dictionaries, the firm possibility of instantiating MLHG in a BDOG, or specific parts of its structures between levels of abstraction, is introduced. For this, two more functions were developed. The first function, *create_nodes_and_relationships (adjacency_matrix, dict_nodes_labels, dict_relationship_types,dict_node_properties, dict_relationship_properties, subgraph_id, node_labels, dict_vertex_label),* allows storing the hierarchical structure of the elements in a Neo4j database, creating nodes and relationships based on the adjacency matrix and the node label mapping dictionary. By allowing us to represent the elements and their relationships in Neo4j, this function allows us to visualize and explore the hierarchical structure of the data more easily. Finally, we have the *query_nodes_and_relationships (labels, properties, values, relationship_types, relationship_properties, relationship_values, label_number, label_interval, label_number_tag, abstraction_level, vertex_labels)*. This function is used to perform queries in a Neo4j database. Basically, it allows you to find nodes and relationships according to a set of specific criteria, namely node labels, node properties, property values, label number, label range, abstraction level and vertex labels. In addition, you can specify relationship types and filter relationships by property and property values. The result of the query is returned as a list of dictionaries containing the nodes found and their properties, as shown in the following example:

[{'node': Node('FOUNDATIONS', duration=60, ref='E10', label='1_1.1.1', vertex_label='Sunlighting', cumulative_cost=3100, estimated_cost=32600, start_date='24/05/2022')}, {'node': Node('FOUNDATIONS', ref='VF12', label='1_1.1.2', vertex_label='Foundation beam', start_date='24/05/2022')}, {'node': Node('FOUNDATIONS', ref='S12', label='1_1.1.3', vertex_label='Footings')}, {'node': Node('FOUNDATIONS', ref='VL2', label='1_1.1.4', vertex_label='Lintel beam', cumulative_cost=90, estimated_cost=200)}, {'node': Node('FOUNDATIONS', tax_id=252977785, top_name='Antonio Lima', label='1_1.1.5', vertex_label='Site layout')}, {'node': Node('FOUNDATIONS_All, label='1_1.1.1', vertex_label='Sunlighting')}, {'node': Node('FOUNDATIONS_All', label='1_1.1.2', vertex_label='Foundation beam')}, {'node': Node('FOUNDATIONS_All, label='1_1.1.3', vertex_label='Footings')}, {'node': Node('FOUNDATIONS_All, label='1_1.1.4', vertex_label='Lintel beam')}]

This function enables interaction with the Neo4j database, allowing queries to obtain specific information in MLHG.

To use MLHG, below will be described the various steps of the process that we have to follow to deal with the data, so that it is possible to explore hierarchical relationships and extract knowledge from MLHG. This process allows the data to be structured into trees or hypergraphs, in which the nodes represent data elements and the connections represent the relationships established between them at different levels. To facilitate data storage and consultation, the structures were integrated into a Neo4j database, using the *create_nodes_and_relationships* function.

4.3 Relational Inheritance Process

In *object-oriented programming* (OOP), inheritance is fundamental to modeling a system, allowing subclasses to inherit attributes and methods from super classes, which promotes code reuse and polymorphism support[23]. Such characteristics are useful in the representation of hierarchies, in which specific entities are formed from generic entities, inheriting and extending their behavior. Similarly, in an MLHG, RI acts as substitution and semantic expansion, where a top vertex is replaced by a lower-level relational structure, while retaining the relationships and properties inherited from the original vertex. As with OOP, the inherited vertex can be seen as a specialized instance, detailing the original entity without compromising the integrity of relationships. This analogy reinforces the hierarchical nature of MLHG, providing an expressive capability like that of object-oriented architecture when applied to the structural and relational domain of multilayer graphs and hypergraphs. In this context, RI can be seen as a form of expanded instantiation, in which each substitution of a subordinate relational structure promotes greater granularity and semantic richness to the resulting global graph.

The central component of an MLHG is precisely the RI mechanism, which enables hierarchical structural expansion in a controlled, coherent and iterative way of the levels and entities that make up the model, preserving the logical and semantic continuity between each layer of the structure. This mechanism allows a virtual vertex, at a given level of abstraction, to be replaced by a lower-level relational structure, maintaining the relational properties of the model and promoting granular and expressive representation of the modeled entities. RI is, therefore, a systematic process of substitution and integration of substructures, based on operations on adjacency matrices. The process allows the propagation of properties and relationships from superior vertices to subordinate vertices, thus creating

[23] It is the principle that allows you to invoke the same operation on objects of different types, obtaining specific behaviors of each type.

a relational tree that expands the hypergraph over multiple levels of abstraction. For the implementation of this mechanism, the following input parameters are used:

- an adjacency matrix A, representing the top-level graph (existing graph);
- a matrix B, representing the graph to be inherited (subordinate graph);
- a virtual node that will be replaced by structure B;
- the respective label dictionaries of A and B, which map the indexes to the entity names.

The inheritance process (Figure 73) is carried out considering the following operations:

1. Identification of the virtual vertex to be replaced in matrix A, establishing the insertion point for matrix B;
2. Dimensional expansion of A to include additional space for the vertices of B while maintaining the integrity of the original structure;
3. Replication of the existing relationships associated with the virtual vertex, to the new coordinates of the expanded array, by row and by column;
4. Direct insertion of submatrix B in the appropriate positions, respecting the coordinates of the virtual vertex and ensuring the topological continuity of the network;
5. Updated the label dictionary, eliminating the virtual vertex and replacing it with the new vertices inherited from B.

This process gives rise to a new Aexp matrix, which incorporates the inheritance of graph B, as well as an updated label dictionary, which reflects the new configuration of the multilevel relational structure. The implementation of this mechanism was carried out in Python. The implementation was fundamental for the construction of the Madj (MLHG) matrices (Chapter 5). In Figure 52 The transition from a virtual vertex to a set of integral vertices as specified in RI is illustrated.

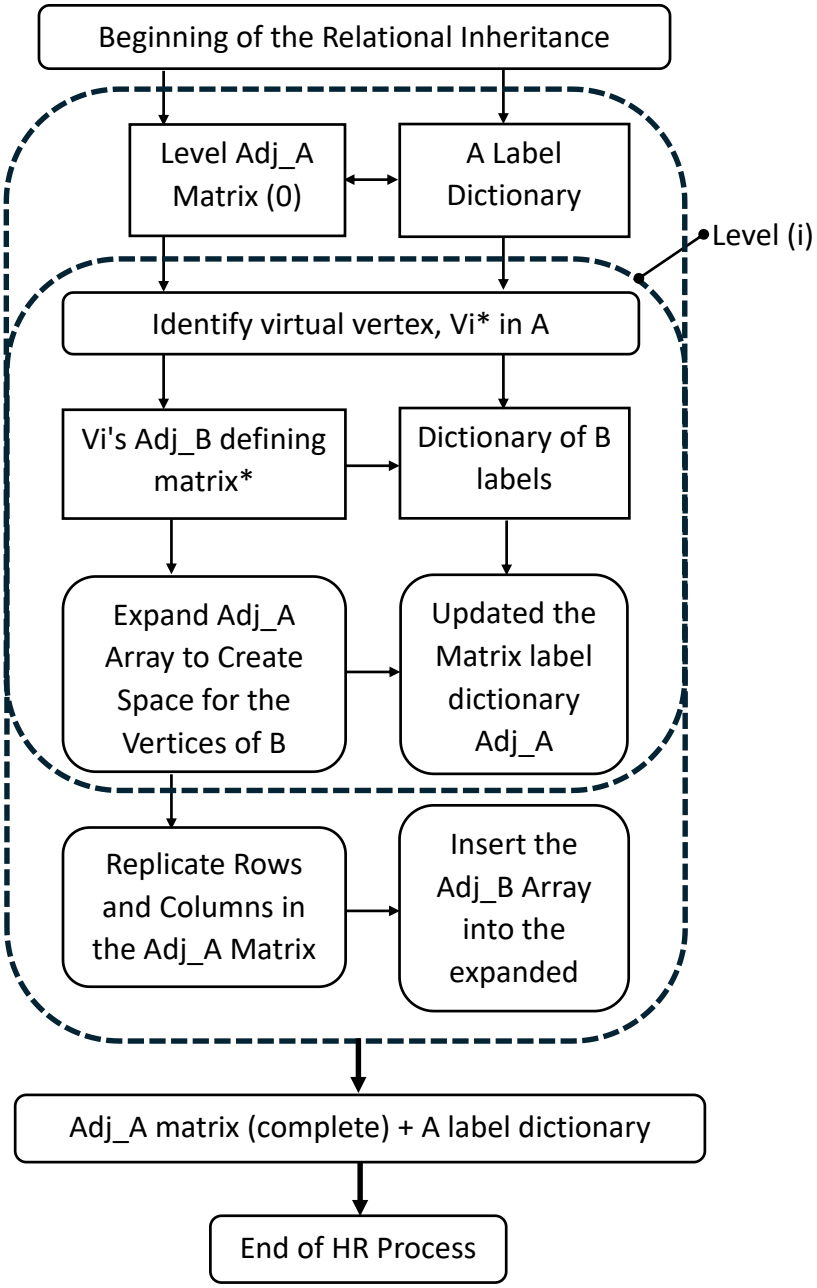


Figure 73 – RI Process.

RI contributes decisively to the expressiveness of MLHG, allowing us to model complex systems with multiple layers without compromising the relational integrity between its components and offering a robust operational basis for future dynamic extensions of the structure. It is particularly relevant in contexts such as hierarchical document management, organizational processes, and contextual semantic networks.

4.4 Property Graphs and MLHG

In terms of structural representation, a property graph is formed by vertices or nodes connected by binary edges, usually oriented. Relationships between nodes include only one root and one destination. Each "common" edge connects two nodes, although we can create more dependencies with intermediate nodes to mimic larger parity relationships Vertices and edges can have labels and attributes that provide them with semantic context. In contrast, an MLHG can be understood as a particular form of hypergraph, in which the hyperedges establish n-to-one relationships, simultaneously connecting several nodes (Francis et al. 2018). Each hyperedge corresponds to a set composed of two or more nodes, which is treated as an autonomous relational unit within the structure. MLHG also introduces a multilevel organization, through which virtual vertices can contain complete substructures, located at different levels of abstraction, allowing a more detailed description of the existing dependencies between system components.

In a graph, by definition, relationships are pairs, vertex-vertex, and therefore connecting more than two entities requires the introduction of additional constructs. Although property graphs support edge labels and rich attributes, their semantics are limited to peer relationships, not allowing the representation of set relationships. Hyperedges allow for higher-order relationships by nature. Thus, n vertices can be connected by a single simultaneous hyperedge that captures the multilateral fact or event without omitted information. For example, a joint article is better represented by a hyperedge connecting all the authors of the publication rather than connecting simple pairs of authors, for example. The ability to group the entities in a relationship gives the MLHG model a rich semantic expressiveness capable of modeling situations where relationships are not strictly binary. The model's multilevel architecture allows you to represent hierarchies in an explicit and semantic aggregation way. Data can be organized into different levels of abstraction. For example, by using a virtual vertex, you can group multiple vertices and hyperedges into a closed entity concept. These actions correspond to composing or encapsulating subgraphs to represent complex structures in a modular fashion. Hyperedges are suitable for modeling complex objects with a high level of hierarchy, where it is necessary to know what a cluster of entities within the same node is or on a higher-level edge, allowing the concept to develop from a simple object to definitions that depend on many relationships without losing its expressiveness. Finally, hypergraph allows multiple entities to share common sub-entities through hyperedges without the need for data redundancy. For example, a hyperedge might join multiple components to the same shared sub-entity, rather than duplicating the sub-component to each relationship. Another semantic difference is the ability

to represent meta-relations or qualify relations with respect to each other. However, in generalized hypergraphs, the relationship is no longer just a connection between nodes and becomes an entity of its own, since hyperedges can also refer to or establish connections with other hyperedges (Kivelä et al. 2014) (Battiston et al. 2020) (By Domenico et al. 2013).

This generalization gives equality to all relationships in the model, which confers an advantage: relationships between relationships can be made explicit and annotated with less artifice than would be necessary in the graph model. Comparatively, the finer granularity of the property graph makes it easier to annotate details specific to each binary bond, implying that instead of a hyperedge connecting several nodes together, an edge by itself can be annotated. That is, the choice between models depends in part on what kind of semantic expressiveness the domain needs – an MLHG offers more expressive power for complex, hierarchical relationships, while a property graph offers more expressiveness and detail for each individual connection (P.P.-S. Chen 1976).

In property graph-based systems, the data schema is usually much more flexible—and sometimes non-existent—compared to that of relational databases. Referential integrity is, by nature, less rigid, since the model privileges structural freedom over the imposition of formal constraints. Although platforms such as Neo4j offer control mechanisms, such as ensuring the uniqueness of properties or verifying the presence of mandatory labels and relationships, the coherence of data in property graphs depends, in practice, on the modelling rules and validations defined in the application logic. The binary nature of the edges means that conceptually n-ary relations must be fragmented into several double bonds, which introduces risks of loss of consistency when there are no additional control mechanisms. Without explicit safeguards, any edge can be changed or removed outside the group to which it belongs, compromising the cohesion of the relationship it was intended to represent. For example, when modeling a ternary relation through three binary edges, nothing prevents one of these links from being eliminated or modified, degrading the semantic integrity of the set.

In a multilevel hypergraph model, the cohesion between related data tends to be intrinsically preserved. A hyperedge that spans multiple objects can act as an atomic unit of association — it is either altered, maintained entirely, or removed altogether —which reduces the risk of partial inconsistencies. In addition, the ability to nest properties or substructures within virtual vertices significantly reduces the likelihood of incoherent states arising. In an MLHG, no vertex remains isolated, because each relation keeps the set of entities logically connected. The possibility of embedding graphs in virtual vertices reinforces logical encapsulation, ensuring that the interdependent components evolve together and

coherently. In this way, the multilevel model introduces a kind of layered integrity: the elements contained in a virtual vertex are an integral part of the top-level entity, avoiding scattered or disconnected references outside the context to which they belong. Studies on hypergraph-based models show that this unified approach reduces redundancy and prevents semantic inconsistencies. Values shared between a sub-object and its super-object are not duplicated but maintained cohesively through the combined structure of hyperedges and virtual vertices, ensuring the logical integrity of the set.

Neo4j and other similar systems offer declarative languages, such as Cypher, as well as APIs for several programming languages, allowing CRUD operations to be performed on vertices and edges, while also supporting ACID transactions. The syntax of these queries has been optimized to quickly traverse binary relationships and express complex graph patterns concisely. Thus, tasks such as finding a path, adding a vertex with properties, or creating a new edge require only a few lines of Cypher code, since the system ensures transactional persistence.

On the other hand, the absence of a widely adopted declarative query language for hypergraph means that many interactions are performed through imperative programming or require the conversion of queries to equivalent graph representations. Still, no model beats a well-indexed property graph when it comes to real-time queries or recommendations based on proximity to nodes. Hypergraphs, however, are expanding into new application domains, enabling distributed architecture by partitioning large graphs into clusters, suitable for massive volumes of data—albeit at the expense of greater complexity in preserving consistency. Even though they rarely outperform graphs in terms of performance, hypergraphs offer substantial advantages in modeling complex and interdependent structures. As hyperedges connect multiple vertices, the cost of storage and traversal tends to grow in combination with the number of connections per hyperedge. For example, a hyperedge that connects k vertices can be equivalent to k binary edges, due to internal transformations for bipartite representation or traversal optimization.

Another advantage of the hypergraph hierarchy is its easy maintenance. In a multilevel hypergraph, this characteristic is manifested by encapsulating the internal dependencies, keeping the implementation details hidden behind defined and stable interfaces. This reduces the likelihood of complex changes and prevents errors from propagating between different levels. Even when inconsistencies arise, they tend to remain confined to the source level, without compromising the overall structure. This hierarchical modularity is also reflected in performance. In large systems, it is not always necessary to access all the available information. Thanks to the organization into levels, it is possible to select only the relevant portion of the graph, concentrating the analysis on the desired level and the directly associated vertices. In

practice, whenever a hypothesis is formulated, the system can load only the necessary subgraph, avoiding unnecessary processing of the remaining data layers.

5 Evaluation of Multilevel Hypergraphs

The demonstration of concepts and structures in real application cases constitutes an essential step in the scientific process. Ideas that remain only on the theoretical plane lack evidence as to their application, resistance and scalability in real situations. In this context, case studies constitute a privileged methodological approach to examine how abstract ideas materialize in real contexts, revealing their usefulness, limitations, and possibilities for expansion. In this chapter we will evaluate the capabilities of an MLHG to deal with domains involving complex structures, multiple relationships and hierarchies of abstraction. Unlike classical property graph models, such as those implemented in systems such as Neo4j, MLHG allows for a formalization of the interdependence between entities of different levels, in which integrity constraints, RI and matrix operations together ensure the structural consistency and generalizability of the system. The main objective is, therefore, to evaluate the adequacy and expressiveness of MLHG in various contexts where it is necessary to represent complex hierarchies and interdependent relationships. Thus, it is intended to demonstrate that, in contrast to traditional models oriented towards operational efficiency or simplified representations, the MLHG model achieves a balance between conceptual clarity, formal rigor and practical application (Yin 2018, Francis Et al. 2018, P.P.S. Chen 1976, Codd 1970b).

More specifically, we will demonstrate the expressiveness of an MLHG in the representation of n-ary relations and multiple levels of abstraction, validate the robustness of RI as a hierarchical expansion mechanism, and compare the scalability and operational clarity of MLHG against classical graph models. Data models require rigorous formalization and an integrative capacity aligned with the classical foundations of data modeling. Codd and Chen (1970) established the foundations of data abstraction by proposing logical-physical separation and constraints for consistency. By using graphs, especially property graphs, one can materialize structures close to natural networks, representing entities and relationships intuitively. However, such structures have limitations, namely, the lack of advanced semantic operators, reveal fragility in constraints and some difficulties in the consistent representation of multiple levels of abstraction. An MLHG emerges as an advanced conceptual model that integrates graph logic with the structuring and integrity mechanisms proposed by Codd and Chen, complemented by an algorithmic infrastructure based on matrix adjacency operations and hierarchical RI. An important feature of MLHG is the possibility of hosting inheritance inspired by object-oriented inheritance, allowing them to replace

virtual vertices with complete relational structures at lower levels, which allows to ensure the topological coherence of the global model, maintaining the continuity of relationships and the consistency of dependencies between the different levels of abstraction. The validation process that we will present will show that an MLHG maintains relational consistency at multiple levels, supports integrity constraints analogous to relational systems, represents complex semantic relationships accurately and allows scaling a graph to demanding domains. The evaluation carried out was supported by three distinct application cases, which were chosen for their structural diversity and their potential to show the capabilities of an MLHG in different areas.

The first case study aims to analyze the environment of a sports organization, which involves several elements in its representation, such as athletes, coaches, boats, competitions, and calendars. The complexity of creating a graph model for this organization lies in the coexistence of hierarchies (clubs, teams, seasons), multiple relationships (coach-athletes-boat-race) and dynamic constraints (selection, logistics, eligibility). An MLHG makes it possible to represent these dimensions simultaneously, offering an integrated view of the sporting activity of that organization. The second case study addresses the application of MLHG to Construction Project Management, an area in which decision tracking, task hierarchies, time sequences, and resource dependencies are key elements. The use of an MLHG model allows the various phases of a civil construction project to be structured, from basic elements to the execution and monitoring of its evolution. The modeling of the system using MLHG promotes more articulated and efficient management, enhancing risk mitigation and resource optimization. Finally, a third example addresses the area of Electronic Auctions, a domain in which there are highly interactive processes in real time, involving various entities such as lots, bidders, organizers or bidding cycles. Auctions pose a challenge to traditional data models. The flexibility and hierarchy of MLHG will make it possible to model the interdependent relationships that emerge during the competitive process and simultaneously analyze different levels of abstraction, from the individual behavior of a bidder to the overall dynamics of the auction.

5.1 Rowing Club

Sport represents a highly complex domain, characterized by different levels of interaction between technical, logistical, human and competitive dimensions, organized in a dynamic and interdependent way. Its management goes beyond the simple statistical analysis of individual or collective performances,

something that is often privileged in the scientific data literature applied to sport. However, the management of sports activities requires models capable of capturing the relational structure that sustains the organization of sports, clubs and federations in a very complex way. In this extraordinarily complex context, rowing emerges as a paradigmatic example of a collective modality, whose organization combines both structured hierarchical relations and extremely flexible horizontal relations. Hierarchical relationships include the articulation between various elements, such as boards, dedicated coaches, athletes and prepared technical teams. However, horizontal relationships are manifested in synchronized coordination between crews, the shared use of vessels and infrastructure, the management of planned competitive calendars and the synchronization of teams during races in a way. This structural duality — simultaneously vertical and horizontal — generates a dense network of interdependencies that can hardly be represented by traditional data models, based on simplified graphs or relational schemas of reduced expressiveness.

The management of rowing clubs and federations illustrates this complexity well: in addition to the technical and physical preparation of athletes, it is necessary to integrate the tracking of performance metrics, the skillful prevention of potential scheduling conflicts, the definition of fair eligibility rules, the careful maintenance and rotation of valuable equipment, and even the simulation of strategic scenarios, such as the reconfiguration of a crew in the event of an athlete's unexpected unavailability. Such requirements reveal the relevance of representation models that can deal with various relations and different levels of abstraction, offering robust support to decision-making processes. Through mechanisms such as RI and matrix operations, an MLHG allows abstract entities to be replaced by structures that include their components, ensuring the coherence of the network of dependencies and interactions that connect the various technical, human and organizational dimensions of the system. In addition to its organizational importance, rowing has a strong cultural dimension. Initially conceived as a means of travel and an instrument of war, rowing became, in the eighteenth century, a sport of great prestige, especially valued in London. The emblematic Oxford-Cambridge regatta, created in 1829 and still held today, consolidated rowing as one of the most demanding sports. Currently, rowing encompasses a wide variety of categories, which involve individual and collective boats with up to eight rowers, and may or may not have a helmsman responsible for guiding the vessel and regulating the crew's pace. The contemporary management of rowing clubs and competitions highlights complex challenges, for example, organizing training and competitions at different levels, optimizing human and material resources, maintaining competitive quality standards or meeting institutional and regulatory requirements. This scenario makes

rowing a fertile field to validate the application of MLHG and demonstrate how this type of graph can capture in a structured way the richness of interactions that characterize modern sport.

5.1.1 Modeling the Club

The MLHG provides a consistent means of representing the variety of relationships and interactions that structure the organizational life of a rowing club. Its flexibility is important to capture the complexity of the management environment, in which there are multiple intricate and dynamic connections between the various data elements that the club involves such as athletes, coaches, boats, competitions, directors and infrastructures. As we have seen before, one of the most relevant aspects of an MLHG is the possibility of multi-layered representation, which allows the analysis of interactions at different levels of detail: from the individual level (athletes and coaches), through the functional level (competition vessels), to the institutional level (management of the club as a global entity). From this perspective, club managers can develop an integrated analysis of internal interactions, identifying patterns, trends, and correlations that would hardly be visible through a traditional management approach. This capacity is particularly relevant in a dynamic context and subject to frequent changes, such as the entry of new athletes, the hiring of additional coaches or the definition of competitive calendars. An MLHG offers decisive advantages in this scenario, by allowing new elements to be easily incorporated into a graph-oriented database, with automatic updating of relationships and preservation of structural coherence.

The case study presented here focuses on the modelling of a rowing club using the MLHG model. The characterized club has nine competition boats (Table 11), each identified by a unique label (Bi). Athletes carry out daily training in their own facilities, which include water tanks and ergometers, ensuring their physical preparation and specialized technique. Of the universe of athletes available, only 12 can occupy the Ai positions, previously defined in each boat. Figure 74 presents the MLHG structure developed for the club and Figure 75 shows its corresponding adjacency matrix. This matrix was conceived as a global representation of management, incorporating hyperedges and vertices for athletes (A), coaches (T), boats (B), helmsmen (TM), competitions (C) and directors (D). The MLHG model developed reveals the existing network of relationships, as well as its hierarchical and multilayer articulation.

Table 11 - Types of boats.

	B1	B2	B3	B4	B5	B6	B7	B8	B9
Skiff	**1**	**1**	**1**	0	**1**	0	0	0	0
Double	0	0	0	**1**	0	0	**1**	0	0
Quadruple	0	0	0	0	0	**1**	0	**1**	0
Eight	0	0	0	0	0	0	0	0	**1**

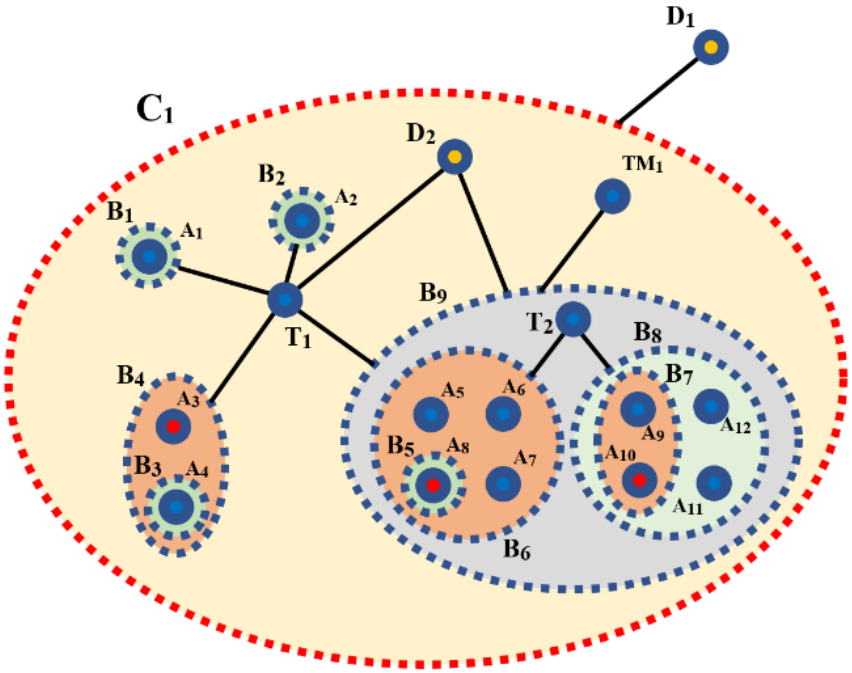


Figure 74 - Illustration of the club management MLHG.

In the club's MLHG model, the edges indicate the various relationships established between the club's entities, the attribution of athletes to coaches or their participation in competitions, for example. Since we are dealing with an MLHG, the hyper-edges can have associated some relationships by inheritance, which in turn can connect different entities, such as athletes, coaches, competitions and types of boats. Each entity (hyper-edge) can have multiple attributes and be associated with several specific relationships established between entities.

	C_1	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9
A_1	**1**	**1**	0	0	0	0	0	0	0	0
A_2	**1**	0	**1**	0	0	0	0	0	0	0
A_3	**1**	0	0	**1**	**1**	0	0	0	0	0
A_4	**1**	0	0	0	**1**	0	0	0	0	0
A_5	**1**	0	0	0	0	0	**1**	0	0	**1**
A_6	**1**	0	0	0	0	0	**1**	0	0	**1**
A_7	**1**	0	0	0	0	0	**1**	0	0	**1**
A_8	**1**	0	0	0	0	**1**	**1**	0	0	**1**
A_9	**1**	0	0	0	0	0	0	**1**	**1**	**1**
A_{10}	**1**	0	0	0	0	0	0	**1**	**1**	**1**
A_{11}	**1**	0	0	0	0	0	0	0	**1**	**1**
A_{12}	**1**	0	0	0	0	0	0	0	**1**	**1**
T_1	**1**	**1**	**1**	**1**	0	0	**1**	0	0	**1**
T_2	**1**	0	0	0	0	0	**1**	**1**	**1**	0
D_1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
D_2	**1**	0	0	0	0	**1**	**1**	**1**	**1**	**1**
TM_1	**1**	0	0	0	0	0	**1**	0	**1**	**1**

Figure 75 - The adjacency matrix of the MLHG.

We will now analyze, more carefully, the various entities that we have defined for the club and the MLHG adjacency matrix that represents the various relationships established between them. In the adjacency matrix, each column corresponds to a relationship (hyper-edge) between the club's entities, namely:

- Competition (C_1), which represents the athletes, coaches and types of boats participating in the competitions; Through this hyper edge we can obtain information about the athletes and coaches who are involved in each competition and what types of boats can be used;
- Type of vessel (B_1..B_9), which defines each of the types of boat in the competition; Each type of boat has a specific relationship with the athletes and coaches, for example, we can identify which athletes were assigned to each type of boat and which coaches are responsible for their training;
- Athlete (A_1..A_{12}), which represents the club's athletes; athletes may be related to the types of boats they compete with and the coaches who supervise them; the defined adjacency matrix allows you to determine which athletes are in each boat and which are their coaches;

- The coaches (T_1, T_2), which characterize the club's coaches, are associated with the athletes they supervise and the types of boats they are involved with; This information allows us to know how the coaches are distributed among the different types of boats and athletes;
- Helmsman (TM_1), who represents the helmsmen of the boats; each helmsman is associated with the various types of boat he drives and, implicitly, with the athletes who are involved in each boat; This information can be used to know its influence on the results between different types of boats and athletes;
- Director (D_1, D_2), which characterizes the members of the club involved in the organization and logistics of competitions and training; Their relationships can be inferred indirectly from the relationships between athletes, coaches and competitions;

Based on the MLHG representation shown in Figure 74, the following practical application cases are analyzed. Athletes are directly related to the types of boats available. Each athlete is associated with one or more types of boat, which indicates that they compete in these boats. For example, the athlete A_8 is related to the boats B_5, B_6 and B_9, which means that he participates in competitions with these three types of boats. Each trainer is related to one or more types of boats, indicating supervision and training actions on these vessels. For example, the T_1 trainer is related to all types of boats available. The D_1 director is not directly related to the types of boats, but rather to the competitions. On the other hand, the D_2 director is related to the B_9 boat – this boat, with eight seats, is the one that will bring the most prestige to the club, if it succeeds in the competition. The helmsman is related to the types of boats available, which require his steering. For example, the helmsman TM_1 is associated with boats B_6, B_8 and B_9.

5.1.2 A Graph-Based Database for the Club

To establish a schema for a graph-based database from the structure of an MLHG, it is important that its creation is carried out according to the schema presented in Figure 76 and that the trees of the logical structure of levels reflect the hierarchical organization of the elements involved in the management of the club.

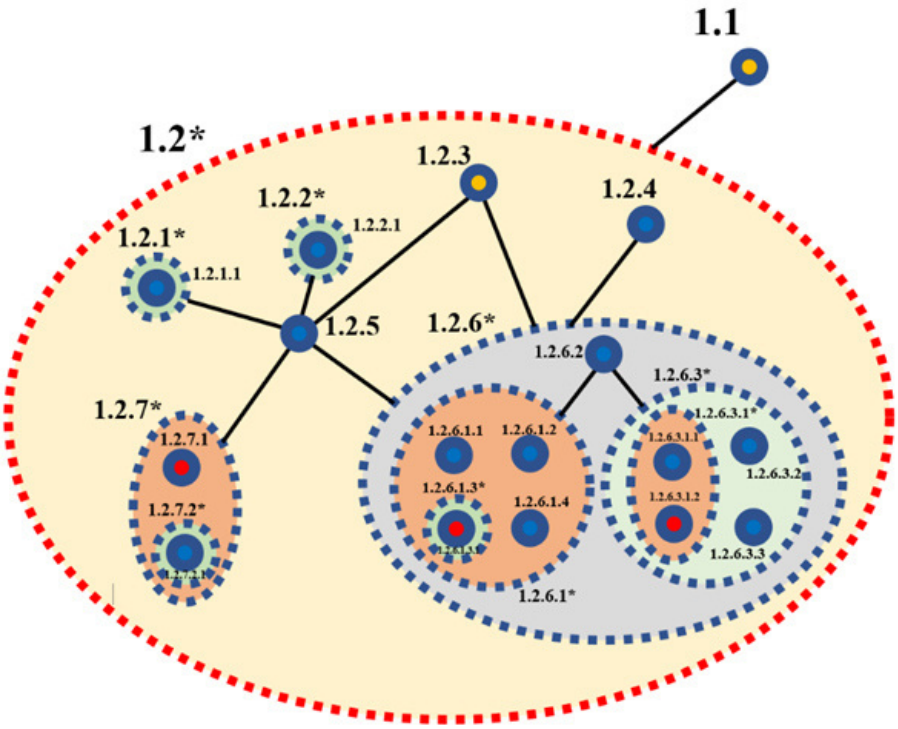


Figure 76 - The schema for the club's graph-based database.

The MLHG structure provides a flexible basis for representing information by defining hierarchical dimensions. It also allows you to create a database model with different levels of detail and complex relationships, having the ability to represent many relationships between the entities of the club. The schema for the database should be defined in such a way as to capture all relevant elements of club management, including the entities, relationships and attributes necessary to represent the data involving athletes, coaches, competitions, resources and other aspects related to the management of the club. The schema presented in Figure 76 represents the club and the relationships between the various management elements. However, it could, if necessary, include other vertices to represent other aspects, such as the coordination of training and competitions or the performance of athletes. As mentioned earlier, logical-level structure trees are essential for dividing club management into multiple hierarchical layers. Each level of the tree includes a specific set of elements, and their relationships, which allows us to obtain a detailed and structured view of the club's management (Figure 77).

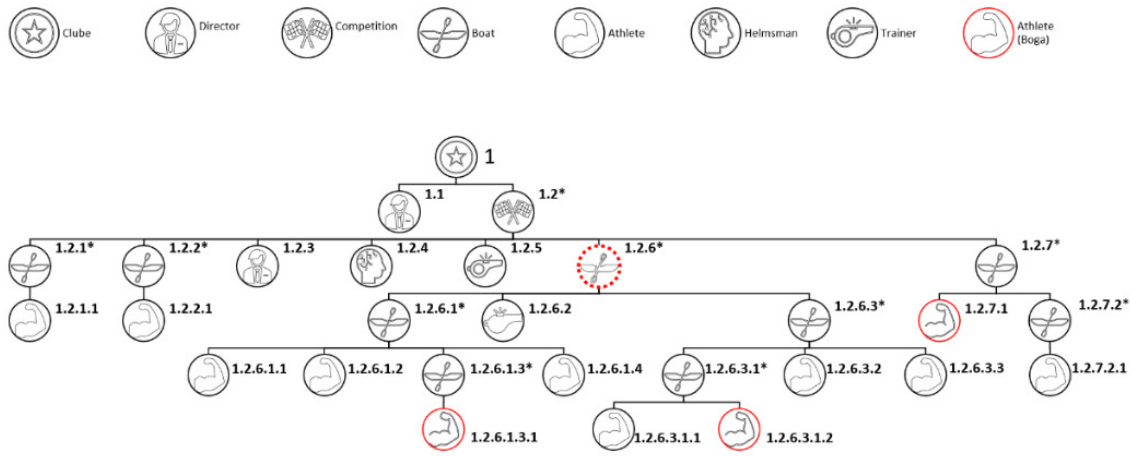


Figure 77 - Logical structure of the club's levels.

In the context of modeling the management of the rowing club, the logical schema defined for the MLHG shows how the 6 levels of abstraction are connected, from the individual level of athletes and coaches to the level of competitions. To create an instance of a given contest (Figure 79), after completion of the aggregation process, previously presented in Chapter 4, an aggregate subgraph representative of the instance is incorporated into the database, using its dictionaries and adjacency matrix (Figure 78) which, at the limit, can be coincident with the general adjacency matrix of the schema.

	1.1	1.2.1.1	1.2.2.1	1.2.3	1.2.4	1.2.5	1.2.6.1.1	1.2.6.1.2	1.2.6.1.3.1	1.2.6.1.4	1.2.6.2	1.2.6.3.1.1	1.2.6.3.1.2	1.2.6.3.2	1.2.6.3.3	1.2.7.1	1.2.7.2.1
1.1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.2.1.1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1.2.2.1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1.2.3	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
1.2.4	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0
1.2.5	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0
1.2.6.1.1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
1.2.6.1.2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.6.1.3.1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.6.1.4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.6.2	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0
1.2.6.3.1.1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.6.3.1.2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.6.3.2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.6.3.3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1.2.7.1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1.2.7.2.1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Figure 78 - The adjacency matrix of an instance.

The overall adjacency matrix of the schema provides a visual mapping representation of the relationships established between club entities for the competition instance. In the array, each cell indicates whether there is a relationship between two entities. This visual representation of relationships is essential to understanding the connections established within the club's organizational structure. For example, if a cell contains a value that indicates a relationship between an athlete and the competition, it suggests the athlete's participation in that competition. Associating this matrix with the formal definition of relations (Table 12), it is possible to establish a labelled subgraph (Figure 79) to support a representative body of a given competition.

Table 12 - Relationship definition.

Entity	**Relationship**	**Description**
Director	RELATED_TO	The person in charge to coordinate the members of the club.
Helmsman	ACCOUNTABLE	The responsible for a specific set of boats during competitions or training.
Coach	TRAINS	The one that trains a specific set of athletes.
Athlete	TRAIN	An athlete trained by one or more coaches specialized in a given sport.

After typifying the relationships involved, using the MLHG adjacency matrix to support the process of instantiation of the competitions, it is possible to obtain the support subgraph presented in the Figure 79. This subgraph offers a detailed overview of the various relational flows between the different entities of the club, in the context of a given competition, without revealing the identity of the athletes, coaches or managers. Through the subgraph, it is observed, for example, that the athlete (1.2.6.1.3.1) relates exclusively to the coaches (1.2.5) and (1.2.6.2), who, in turn, maintain a direct connection with the nodes (1.1) and (1.2.3). This configuration reveals his condition as an athlete under direct and supervised guidance within the club's technical structure, revealing the hierarchical and relational coherence of the MLHG model in the representation of these interdependencies.

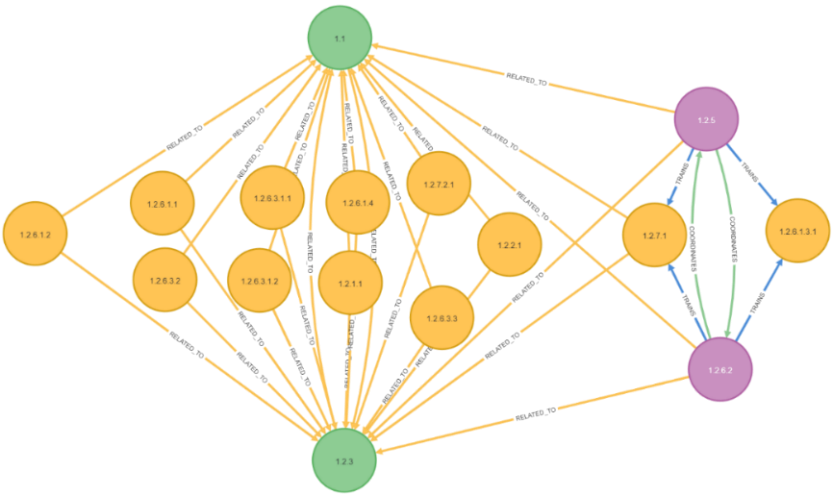


Figure 79 - Aggregate subgraph of support for instantiation.

In the final stage of filling out the database, the individuals to be included in the database as athletes were selected. The athletes were chosen based on their performance history, which includes their technical level, endurance and physical strength. In addition to physical abilities, the potential of the athletes was also evaluated, based on their mental abilities, so that we can achieve better competitive results. The desire of the athletes to train and compete, along with their commitment to the club, was also considered as determining factors in the selection of athletes. This process resulted in the formation of a more detailed subgraph (Figure 80), which allows the identification of all participants in each competition. In this subgraph, we can see, for example, that the athlete 'Ethan' relates exclusively to two coaches, 'David' and 'Henry', who in turn communicate directly to the directors James and Michael, his situation as an athlete.

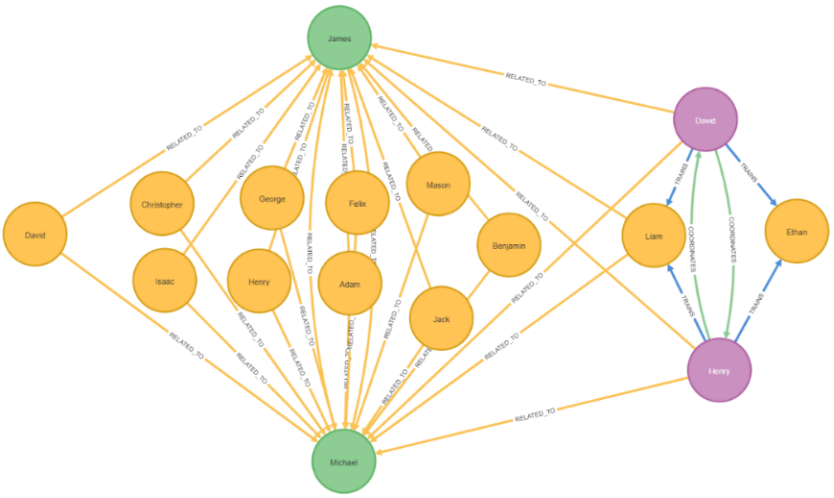


Figure 80 - Aggregate subgraph of participants' support for the competition.

To conclude, we must highlight, once again, the importance of creating a graph-based database for the club, using an MLHG model to represent the complex relationships that can be established between one or more entities of the club. The database created is a robust structure to store and organize information related to athletes, coaches, competitions, types of boats and other essential aspects in the management of the club. By using an MLHG as a fundamental database model, we can capture and model the interactions and interdependencies between the different entities of the club, which facilitates the implementation of analysis and decision-making processes and provides a high-level conceptual model for the database.

5.2 Modeling a Building Project

Civil construction is one of the most complex and determining sectors of human activity, with a direct impact on the economy, society and the well-being of populations. The development of infrastructures, buildings and urban spaces involves technical and architectural design, as well as the rigorous coordination of processes, resources and agents, whose articulation is often marked by high uncertainty and existing levels of interdependence. The nature of this domain places particular demands on representation and analysis models, since simple linear task description or traditional graph modeling are not enough to capture the structural and procedural richness involved proposed a formalization for the analysis of spatio-temporal conflicts in construction projects, highlighting the shortcomings of conventional linear approaches. This context served as a basis for automating the identification of conflicts and for sustaining methodologies that the MLHG approach aims to overcome (Managing Construction Projects An Information Processing Approach 2021, Marx and König 2013).

A building project can be understood as a dynamic socio-technical system, in which different entities — engineers, architects, designers, builders, suppliers, regulators, and customers — interact simultaneously with physical elements — pillars, beams, foundations, slabs, and technical installations. In turn, these relationships are conditioned by contractual, legal and financial factors that determine the fulfillment of deadlines, budgets and quality standards. The effective management of this ecosystem therefore requires conceptual models capable of integrating human, material and procedural dimensions in a coherent and flexible way. It is in this context that MLHG can be a particularly promising approach. Through the possibility of hierarchical representation and high-order interactions, the model allows structuring the complexity of building works, capturing from the most elementary level (such as the

connection between a column and a beam) to the most abstract levels (such as the coordination of human resources or the contractual management of the project). As evidenced in the previous case of rowing, the formal flexibility of MLHG makes it possible to deal with intricate and dynamic networks, but now with application in a technical-scientific domain in which structural robustness and procedural rigor are decisive.

In this second application case, the modeling of a small building will be presented and discussed, used as a representative example of the various challenges that we can find in the field of Civil Engineering. Through this example, we will seek to demonstrate how an MLHG can support the analysis of the interdependencies between structural elements, optimize the management of construction processes and provide a conceptual basis for information systems oriented to building processes. The management of large works requires the articulation of several work fronts (earthwork, structure, installations, finishes, etc.), all interconnected by precedence relationships, functional dependencies and command hierarchies. Through the use of a MLHG model, it is possible to develop a data structure that serves as a conceptual schema for a BDOG, applicable to real scenarios in the construction sector. In this application, the virtual vertices ($V_{i,n}$*) should be used to represent constructive processes, and their relations with the vertices belonging to the same level of abstraction should reflect the elements, resources or entities shared with these processes — and may, in some cases, involve other interdependent processes. In the case of non-virtual vertices, also inserted at the same level of abstraction as a specific virtual vertex ($V_{i,n}$*), the modeling must consider the dependencies and functional interactions necessary for the execution of the process, ensuring coherence between the represented entities.

The structured modeling of complex phenomena in civil construction based on MLHG is, therefore, one of the central objectives of this work. To illustrate this, we can take as a reference the model presented in the Figure 81, which includes vertices that represent government entities and two virtual vertices, $V_{1.2}$* and $V_{1.2.1}$*, which represent, respectively, a main construction process and a sub-process associated with the execution of structural components. Based on the elements represented in this figure, it is possible to describe in a synthetic way the relationships underlying the modeling, evidencing the logic of dependencies and RI that leads to the result obtained.

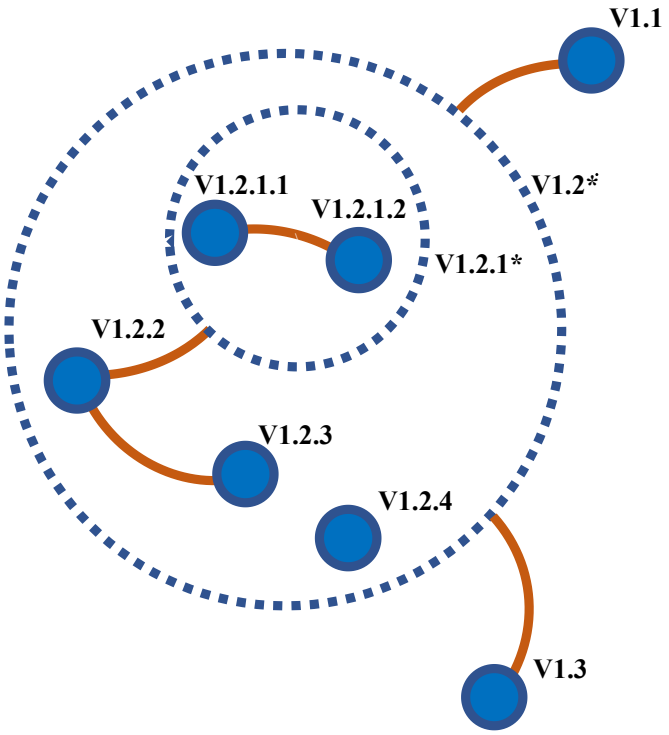


Figure 81 - Initial MLHG configuration.

In the MLHG model, virtual vertices $V^*_{1.2}$ and $V^*_{1.2.1}$ can represent the planning phase of a construction project and the choice of materials, for example. The relationship between these virtual vertices, and the remaining abstraction-level vertices are the relationships shared with a given process. Similarly, for non-virtual vertices, contained at the same level of abstraction as the virtual vertex under consideration, it is necessary to consider what is needed for the process, other processes, or other entities. In this way, it is possible to model the branches that represent the way dynamic relationships in civil construction, such as human resources entities, material resources, financial resources, equipment, technological resources, legal regulations, and other things. The modeling carried out supports the representation of the construction processes involved, which facilitates the identification of possible problems and more efficient decision-making in the field of civil construction.

For the implementation of a BDOG, a pre-defined schema is not strictly necessary, since these systems are, by nature, schema-less. However, it is convenient to define a guiding conceptual schema, which ensures consistency and coherence in the instantiation of the basis in the context of the real problem. In some BDOG approaches, as already discussed, the logical schema may partially coincide with the physical schema, which is usually represented by a digraph. It is important to note that the instantiation of the vertices associated with the entities reflects the existence of a certain type of node in

the conceptual model and that, consequently, the relationships between these instantiated vertices are automatically reflected in the graph.

In this specific context of civil construction, a construction process can be represented through a virtual vertex, $V^*_{1.2}$, which can share technical-economic relationships with other entities, represented by vertices $V_{1.1}$ and $V_{1.3}$. Similarly, the existential support of $V^*_{1.2}$ results from another process that involves the interaction of several entities, represented by the vertices $V_{1.2.2}$, $V_{1.2.3}$, $V_{1.2.4}$, and the $V^*_{1.2.1}$ process. The existence of $V^*_{1.2.1}$ in turn depends on the relationship between the entities $V_{1.2.1.1}$ and $V_{1.2.1.2}$. Thus, it is possible to identify a total of 14 relationships (Figure 82) that can be represented from the 5 initial relationships considered in the MLHG modeling presented in the Figure 81.

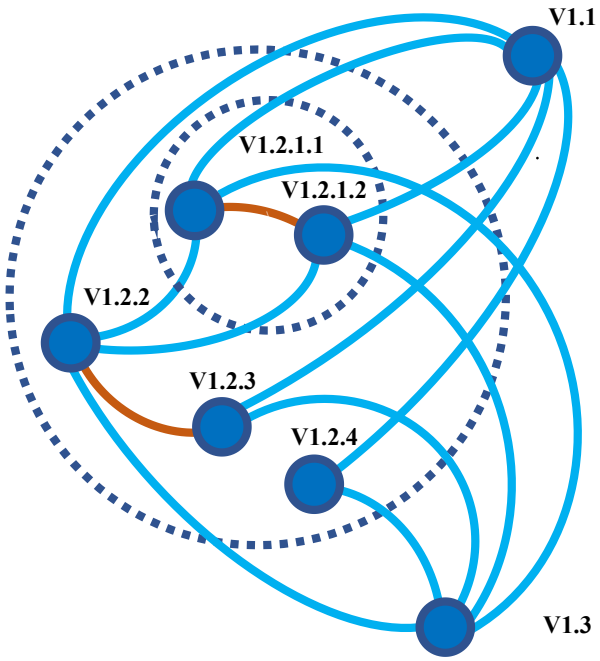


Figure 82 – The final MLHG for the propagated structure.

The BDOG schematic is now represented by an MLHG, composed of levels with sets of vertices (entities) and edges (relationships). In this proposed new model type, a virtual vertex can be thought of as the root of a set of related entities. To instantiate a vertex at a given level, it is necessary to identify all the relationships in which it participates, including those that do not belong to RI, that is, the functional or contextual connections that do not derive from hierarchical dependence. While inheritance relationships represent substitution or specialization—in which an upper vertex is expanded by a lower structure—non-hierarchical relationships describe complementary interactions between independent entities, such as technical associations, information flows, resource sharing, or temporal links between

processes. These relationships ensure that the instantiated vertex maintains semantic coherence with its operational context, allowing to represent both the hierarchical verticality and the functional transversality of the modeled structure. Each instantiated vertex generates a structure for entry into the database, without adding new types of elements to the graph. Instantiating a hierarchical level implies the instantiation of all elements contained within that level and of those at higher levels, whereas instantiating a vertex at a given level entails the instantiation of all vertices at higher levels. You can instantiate vertices at different levels, but you must establish all relationships by RI beforehand. Instantiation is complete when all RI relationships are established at all levels involved. With this model, it is therefore possible to preserve the structure of the elements already instantiated in the database, including the relationships between entities, even in the face of possible partial changes in the data structure.

It is important to remember that one of the founding motivations of this work was the definition of an innovative data model, designed to work as a BDOG conceptual schema, ensuring semantic control in advance of instances and relationships between entities. The objective was to ensure that, in the face of structural modifications in the model or in the abstraction layers, it was possible to preserve the topological persistence and dynamic referential consistency of the elements already instantiated in the database, including their relational links, without compromising the overall integrity and structural continuity of the graph. For this purpose, we propose a data model of a graph-oriented database based on MLHG, to model a real-world situation in civil construction. To that end, let's look at some issues related to this new approach:

- Can the schema of a graph-driven database be represented by multilevel hypergraph?
  Yes. Although conventional BDOG schemas are usually described by simple digraphs, an MLHG allows for a more comprehensive representation, capable of reflecting the hierarchical and functional structure typical of construction processes. In this context, a digraph can be interpreted as a particular case of MLHG, limited to a single level of abstraction.
- What can a virtual vertex, V* represent in multilevel hypergraph?
  In an MLHG, a virtual vertex V* represents both a level of abstraction and the root of a subset of related entities or processes. In the context of civil construction, it may correspond, for example, to a construction process composed of several interconnected stages, functioning as an aggregating element of the hierarchical structure of the graph.
- Instantiate levels or instantiate elements of a given level?

In MLHG construction process modeling, instantiating an abstraction level n implies knowledge of its adjacency matrix Madj, n, which describes the relationships between the technical entities represented at that level — for example, between activities and resources. This operation allows all entities associated with that level and higher levels to be materialized in the model, ensuring the hierarchical coherence of the whole. However, the most common procedure is to instantiate specific elements of a given level, such as tasks, equipment or sub-processes, as their relationships are more easily identifiable in each construction context. To this end, the instantiation algorithm must be executed iteratively, aggregating the technical and functional dependencies between consecutive levels, to correctly reflect the structure and logical sequence of the execution of the work.

- To instantiate a vertex, do you need to identify all relationships by RI?

  In MLHG-based modelling of construction processes,, the instantiation of a vertex — corresponding to an entity or technical component — occurs at a specific level of abstraction. While it is not necessary to manually identify all relationships by RI, it is essential that the aggregation algorithm automatically establishes the links between entities belonging to different levels. This way, when instantiating a vertex, such as a structural element or a constructive activity, the system propagates the hierarchical and functional relationships that link it to subprocesses, resources or higher phases, ensuring the integrity and overall coherence of the model.

- For each vertex instantiated at a given level of abstraction, is its base structure created for entry into the global base structure?

  Yes. In MLHG-based applications in the field of civil construction, each instantiated vertex — representing, for example, a construction process, a structural element or a technical resource — originates its own base structure, which is then incorporated into the overall structure of the graph.

  The integration of these new elements occurs by agglutination, that is, by the controlled union of the structure generated during the instantiation with the existing base. This process ensures that new entities and their relationships are added without compromising the coherence of the database, preserving the hierarchical and relational integrity of the overall model.

- Is it possible to instantiate vertices by levels?

Yes. In the context of modeling construction processes with MLHG, it is possible to instantiate vertices by levels of abstraction, if it is ensured that the instantiation process is only considered complete when all relationships by RI have been properly established. This ensures that each vertex — whether it is representative of a construction phase, a resource or a technical sub entity — maintains the hierarchical and functional connections necessary for the overall coherence of the model, preserving the continuity between levels and the integrity of the graph's structural dependencies.

- Does instantiating a vertex at a given level imply instantiating all vertices at contained levels?
  Yes. In the MLHG model applied to civil construction, the instantiation of a vertex at a given level necessarily implies the instantiation of the vertices of the hierarchically contained levels. This is because each vertex represents an entity or process that depends, directly or indirectly, on elements defined at lower levels — for example, a construction process that integrates tasks, resources, and technical components. Thus, to ensure the structural and relational consistency of the model, each instantiated vertex must have at least one associated instance at the subordinate levels.
- Instantiating a vertex at a given level is the same as instantiating the entire structure from that level up to the Sim level. In the context of the modelling of construction systems through MLHG, the instantiation of a vertex at a given level implies the activation of the entire corresponding hierarchical structure, from that level to the maximum level, ensuring the presence of at least one base vertex in each layer. For example, when instantiating the vertex corresponding to a construction phase — such as "Execution of the Concrete Structure" — the model automatically generates the instances associated with the subordinate levels, including sub-processes (such as formwork, reinforcement and concreting), human resources, equipment and materials. In this way, it is ensured that the model fully represents the technical-functional chain of the construction, maintaining coherence between the hierarchical dependencies and the operational flows of the project.

MLHG functional abstraction can be applied as a data model to represent and analyze complex construction systems at different levels of detail, serving as a conceptual basis for the implementation of a BDOG. The MLHG model thus constitutes a hierarchical representation structure that allows capturing the multiple interactions between the elements of a construction system — from physical entities (such as pillars, beams and foundations) to technical processes, human resources and logistics flows. Its main

advantage lies in its ability to decompose a project into several levels of abstraction, providing a detailed and structured view of the interdependencies that characterize the life cycle of a project.

The main computational input components of MLHG and their role in the representation of data in the constructive context are recapped now:

- Adjacency Matrix. It is one of the fundamental elements of MLHG. It represents, in matrix format, the connections between the entities of the system at each hierarchical level — for example, between construction processes, tasks and technical resources. Each cell of the matrix indicates the presence or absence of a relationship between two elements. This representation is essential to visualize the structure of interdependencies and understand how changes in a certain component (for example, the delay in a task) can propagate through the remaining levels of the work;
- Dictionary Vertex Labels. It works as a systematic index that assigns unique identifiers to each node of the graph. In the case of civil construction, these labels can correspond to codes of structural elements, planned tasks, equipment or work teams. This indexing facilitates the tracking and analysis of each component individually within the overall project structure;
- Dictionary of Label-Vertex. Associates the numerical labels of the nodes with corresponding semantic descriptions, providing technical and functional context to each entity. For example, a vertex labeled "1.2.3" might refer to "Beam Reinforcement Execution." That semantic layer makes the model interpretable by both technical experts and automated planning and monitoring systems;
- Dictionary of Types of Relationships. It defines the different types of interaction between the vertices, reflecting the existing dependencies in the construction process. They may include relationships of precedence (order of execution), physical support (interconnected structural elements), resource dependency, or hierarchical supervision. The explicitness of these types of relationships is essential to correctly represent the operational dynamics and technical coherence of the work.

The articulation of these four components allows the construction of a BDOG with the ability to represent effectively the structure and behavior of a real work. The adjacency matrix defines the relational topology between processes, elements, and resources; node dictionaries and labels ensure semantic traceability and hierarchical control of entities; and the dictionary of relationship types ensures the logical coherence of interactions between different levels of abstraction. Combined, these elements make it

possible to instantiate a complete constructive graph, in which each vertex reflects a functional unit (such as an activity, a physical element, or a human agent), while the edges capture the dependencies and operational flows between these entities. This structure makes it possible to analyze the coherence between construction phases, to simulate execution scenarios and to detect inconsistencies in planning or resource allocation at an early stage – critical aspects for the efficiency of a construction project.

A key feature of the MLHG model is its ability to represent and analyze complex systems in a hierarchical and multiscale manner. The model starts with a global view, at level 1, which provides an integrated reading of the interactions between the main components of the system — in the case of civil construction, between the macro phases of the project, such as design, execution and control. As you go down the hierarchical levels, the model allows you to progressively deepen the detail, disaggregating each component into sub-processes, specific tasks, resources or associated physical elements. This controlled decomposition provides a gradual and contextualized understanding of the interdependencies between the various agents and operations of the work, allowing the identification of how decisions made at a strategic level have repercussions on operational levels and vice versa.

In the specific case of the application in the field of civil construction, an application in a generic civil construction contract, the MLHG, presented in the Figure 83 represents the schema to the graph-oriented database of the contract status.

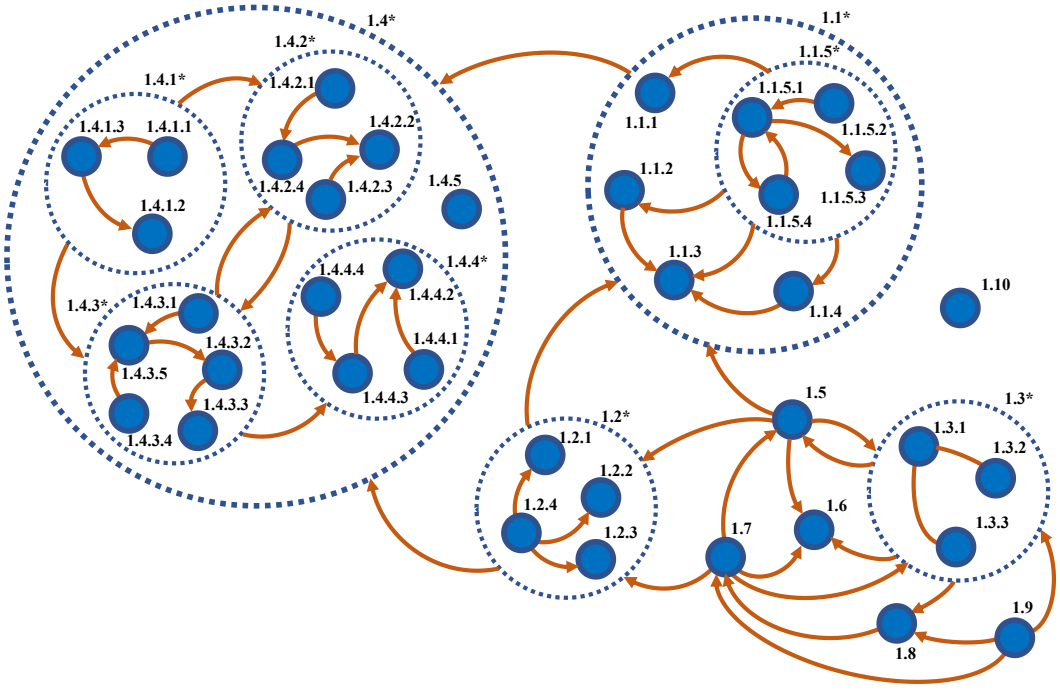


Figure 83 - Schema for BDOG of a civil construction contract.

The Figure 83 presents the conceptual schema for BDOG, derived from the application of the MLHG model to the context of a civil construction contract. In this schema, each node represents an entity of the construction process, while the edges indicate functional, technical or administrative relationships between these entities.

The virtual vertices (*) indicate encapsulated hierarchical substructures, corresponding to processes or groups of activities, which can be detailed at lower levels of the model. The graph thus allows for visualizing the relational architecture of the contract, highlighting the dependencies between resources, teams, structural elements and external entities, constituting the basis for the formal definition of the adjacency matrix and the respective dictionaries of labels and vertices.

Based on the adjacency matrix (matriz_adjacencia_G1), which formalizes the connections between the MLHG nodes at level 1, it is possible to identify the relational structure that supports the BDOG schema. Nodes at this level are designated by the labels defined in the dictionary dict_nodos_rotulos_G1 — {0: '1.1*', 1: '1.2*', 2: '1.3*', 3: '1.4*', 4: '1.5', 5: '1.6', 6: '1.7', 7: '1.8', 8: '1.9', 9: '1.10'} — and associated, through the dict_rotulo_vertice_G1 dictionary, with the respective entities in the field of construction: {'1.1': 'Foundations', '1.2': 'Resources', '1.3': 'Technical team', '1.4': 'Superstructure – Group', '1.5': 'Site protection', '1.6': 'Construction site', '1.7': 'Owner', '1.8': 'Inspection', '1.9': 'External entities', '1.10': 'Licensing process'}. Each label represents, therefore, a functional or structural entity of the contract. For example, the vertex '1.1' corresponds to Foundations, which depend logistically on '1.2' – Resources and are supervised by '1.3' – Technical team. Similarly, '1.4' – Superstructure – Group interacts with '1.8' – Inspection and '1.9' – External entities, reflecting the regulatory control and validation relationships associated with the construction phases. In turn, relationships are defined by another dictionary of relationship dictionaries, dict_tipos_relacionamentos_G1 = { '1.1': { '1.4': ["ENDURE"] }, '1.2': { '1.1': ["LOCK"], '1.4': ["LOCK"]}, '1.3': {'1.5': ["ADVISES"], '1.6': ["COAT"], '1.8': ["VALIDATE INFORMATION"],},'1.5':{'1.1':["TECHNICAL RESPONSIBILITY"], '1.2': ["CONTROL"], '1.3': ["QUESTION"], '1.6': ["RESPONSIBILITY"]}, '1.7': {'1.2': ["FINANCE"], '1.3': ["COMMUNICATE"], '1.5': ["QUESTION"], '1.6': ["RESPONSIBILITY"]}, '1.8': { '1.7': ["ADVISES"] }, '1.9': { '1.3': ["ADVISES"], '1.7': ["COMMUNICATE"], '1.8': ["ADVISES"] }}, This dictionary establishes the interactions between the vertices, indicating the type of relationship that exists. For example, node '1.1' has a relationship of "ENDURE" to node '1.4', node '1.2' has relationships of "LOCK" to nodes '1.1' and '1.4', and so on. Thus, the adjacency matrix, together with the dictionaries of labels and vertices, formally defines the

relational topology of the contract, serving as a basis for its computational instantiation as an oriented graph.

As can be seen in Figure 84, in the first level (n = 1), there are ten elements — identified as (1.#) — that represent entities of the construction system, such as Foundations, Resources and Technical Team, among others. Each of these entities is broken down into lower hierarchical levels, where their components, dependencies and interaction flows are specified. These interactions are formally described through the corresponding adjacency matrix and the label and vertex dictionaries, which associate each node with its directed connections and its semantic meaning in the context of the project. Such structuring permits us to visualize and analyze, in a systematic way, the relational dynamics between the different areas of the project, making the global model more transparent and functional. As a result, the system becomes more accessible and provides a basis for strategic decision-making, underpinned by an accurate understanding of the interdependencies between its components.

With the data pre-processed, the logical structure trees are built based on the structure of the labels (#.#.#.#.#. #) using the functions described earlier. These structures capture the relationships and connections between data elements at different hierarchical levels in which, each node is identified by a numerical label and has a vertex label that describes the semantic function in the graph, where it is possible to represent in the form of a diagram (Figure 84), referring to level 1 of the MLHG. In this diagram, relationships are established through the connections between nodes and indicate the type of interaction between them.

Matrix_adjacence_G1 =

[0 0 0 1 0 0 0 0 0 0]
[1 0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 1 0 1 0 0]
[0 0 0 0 0 0 0 0 0 0]
[1 1 1 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 1 1 0 1 1 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 1 0 0 0 1 1 0 0]
[0 0 1 0 0 0 1 1 0 0]

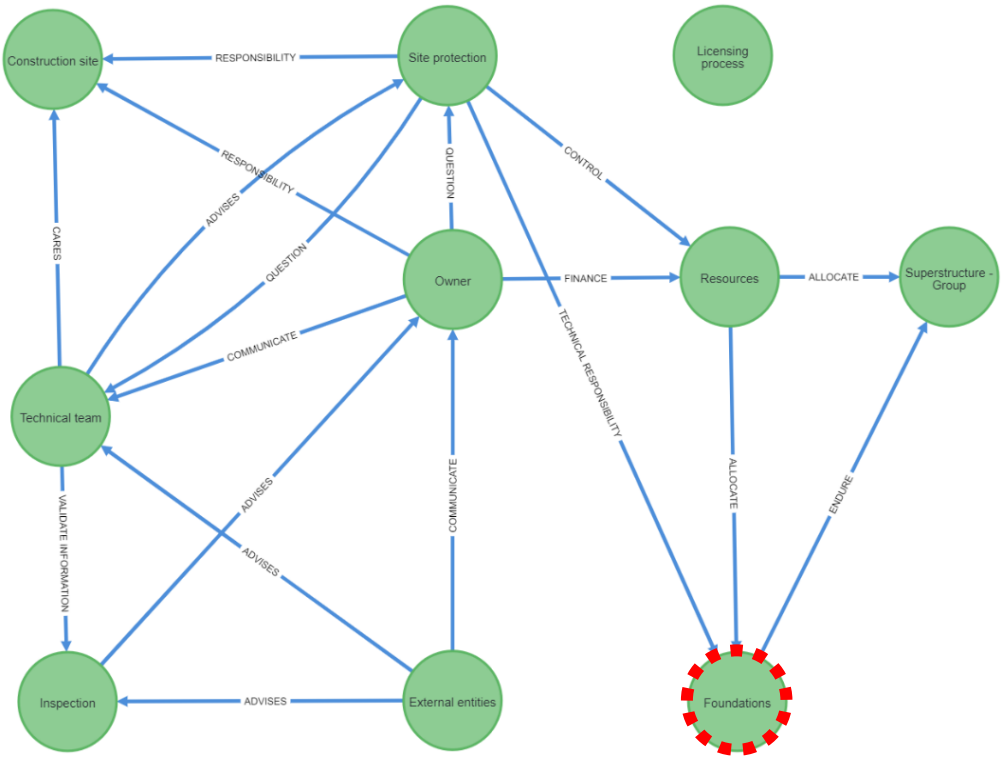


Figure 84 - Level 1 graph.

From the elements that make up the MLHG in level 1 and their formal definitions, we will then make a context description of the interactions between the entities, highlighting the relationships present in this level 1.

Let's look at the "Foundations" entity (node '1.1'). This is responsible for providing the basis for the project. It establishes a "ENDURE" relationship with the "Superstructure - Group" entity ('1.4'), indicating that the foundations support the structure of the superstructure group. The "Resources" entity ('1.2') plays a major role in the project, allocating the necessary resources for its development. It establishes an "ALLOCATE" relationship with both the "Foundations" entity ('1.1'), ensuring that resources are allocated to the foundations, and with the "Superstructure - Group" entity ('1.4'), ensuring that resources are allocated to the superstructure group. Still at this level, we find the entity "Technical team" ('1.3'). This entity (a team) performs several functions in the project. This entity establishes an "ADVISES" and "Site protection" type relationship (1.5), translating the advisory and guiding role of the technical team in the protection and maintenance actions of the construction site. It also maintains a "CARES" relationship with "Construction site" (1.6), reflecting the team's direct responsibility for the organization and operational safety of the shipyard. Finally, it has a "VALIDATE INFORMATION"

relationship with the "Inspection" entity (1.8), ensuring the technical verification and validation of data from inspection and quality control activities.

It is important to mention the interactions between the various entities that directly affect the development of the project in question. Site protection (1.5) assumes dual responsibility, technically guarding the foundations (1.1) and carefully monitoring the indispensable resources (1.2). However, when necessary, he assertively questions the "Technical team" ('1.3') in search of clarification. Finally, it shares responsibilities with the technical team regarding the "Construction site" (node '1.6'). In a different way, "Owner" (n'1.7') establishes several connections with key entities. It finances the "Resources" ('1.2'), guaranteeing means for their allocation. In addition, it proactively communicates with the "Technical team" ('1.3') to ensure the needs of the project, as well as questions both the "Site protection" (node '1.5') and the technical team in the expectation of subsidies for correct decision-making. In addition, the relationship between the "Inspection" ('1.8') and the "Owner" ('1.7') is relevant. The former advises the latter based on assessments, offering valuable guidance for ensuring the expected quality and compliance. Finally, the "External entities" ('1.9') exert constructive influence on the process and advise the "Technical team" ('1.3'), contributing with important knowledge, communicate with the "Owner" ('1.7'), for fluid exchange of information and alignment of objectives, and advise the "Inspection" ('1.8'), in search of the continuous improvement of its methods. The interactions between entities, represented in the nodes of level 1, allow us to glimpse the connections and interdependencies that are fundamental to the efficiency, quality and success of the enterprise.

Now, let us consider, at level 2, the "Foundations" entity, identified by the vertex label *1.1* at level 1. This entity describes, in semantic terms, the structural function of the foundations in the context of the building, being represented in the MLHG through the four elements that define its structure:

- the adjacency matrix, which represents the connections between the subgraph's internal nodes;
- the dictionary of knots-labels, which associates numerical indices with the identifiers of each node;
- the label-vertex dictionary, which translates identifiers into semantic descriptions;
- the dictionary of relationship types, which defines the nature of the relationships between the elements of the system.

The basic data corresponding to the "*Foundations"* entity are as follows:

Matrix_adjacence_G1_1_FUN=

[0 0 0 0 0]

[0 0 1 0 0]

[0 0 0 0 0]

[0 0 1 0 0]

[1 1 1 1 0]

```
dict_nodes_labels_G1_1_FUN = {
0: '1.1.1',
1: '1.1.2',
2: '1.1.3',
3: '1.1.4',
4: '1.1.5'
}
dict_vertex_label_G1_1_FUN = {
'1.1.1': 'Natural lighting',
'1.1.2': 'Foundation beam',
'1.1.3': 'Footings',
'1.1.4': 'Lintel beam',
'1.1.5': 'Site layout'
}
dict_relationship_types_G1_1_FUN = {
'1.1.2': {
'1.1.3': ['BALANCES']
},
'1.1.4': {
'1.1.3': ['LOCKS']
},
```

```
'1.1.5': {
'1.1.1': ['PRECEDES'],
'1.1.2': ['PRECEDES'],
'1.1.3': ['PRECEDES'],
'1.1.4': ['PRECEDES']
}
}
```

In this subgraph, the hierarchical structure of foundations is detailed into five main components:

- Foundation Slab (1.1.1) — represents the planning and orientation of the terrain to ensure adequate support conditions;
- Foundation beam (1.1.2) and Footings (1.1.3) — describe the structural elements that distribute the superstructure's loads;
- Lintel beam (1.1.4) — reinforces stability between support points by "locking" (LOCK) movements;
- Site layout (1.1.5) — acts as an organizing element of the process, preceding (PRECEDES) the execution of the remaining structural components.

This representation demonstrates how MLHG allows semantically decomposing the entity "Foundations" into interdependent components, preserving the hierarchical and functional coherence between the elements of the construction system. From the structural elements and with the integration of these governmental structures in the Neo4j database, the diagram presented in the Figure 85. In this diagram, the node of the foundations is represented in the form of a graph, whose context description clarifies the interactions between the entities, highlighting the relationships present in this entity belonging to level 2.

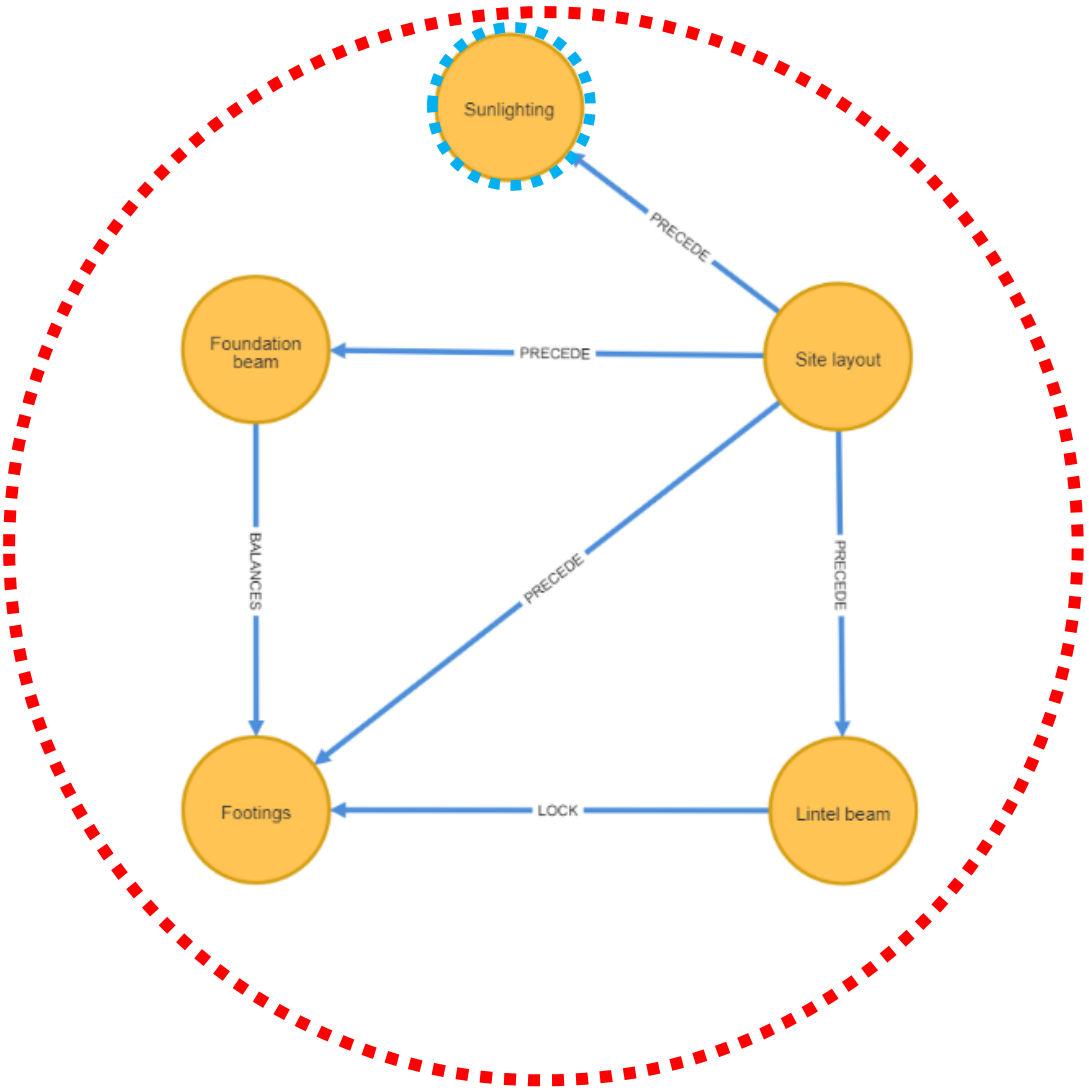


Figure 85 - Level 2 graph - "Foundations" (node '1.1*').

At the most detailed level of the foundations, it is possible to define the adjacency matrix matriz_adjacencia_G1_1_FUN, which represents the connections between the nodes of the structural system in this sublevel. It is a 5×5 matrix, which describes the interdependencies between the constituent elements of the foundations. Each node is identified by a unique label in the dict_nodos_rotulos_G1_1_FUN dictionary, associated with a semantic designation in the dict_rotulo_vertice_G1_1_FUN dictionary. For example, node 1.1.1 corresponds to "Foundation Slab", node 1.1.2 to "Foundation beam", and node 1.1.3 to "Footings", thus describing distinct structural components and their functions in the context of the work. The relationships between these nodes are formalized in the dict_tipos_relacionamentos_G1_1_FUN dictionary, which defines the type of interaction between the entities. For example, node 1.1.2 establishes a "BALANCES" relationship with node 1.1.3,

evidencing the structural balance between the foundation beam and the foundations. Similarly, node 1.1.4 (Lintel beam) maintains a "LOCK" relationship with node 1.1.3 (Footings), indicating the locking mechanism that reinforces the stability of the foundations.

Based on these links and associations, it is possible to identify the interactions between the entities. The Foundation Slab ('1.1.1') plays a key role in the governance of the construction site. It directly influences the entity "Foundation beam" ('1.1.2') through a "BALANCE" relationship, indicating a proper balance between the foundation beam and the foundation. The entity "Lintel beam" ('1.1.4') is related to the entity "Foundation beam" ('1.1.3') by a relation of "TRANCA", representing that the lintel beam locks the foundation. On the other hand, the "Site layout" entity ('1.1.5') has "PRECEDE" relations with nodes '1.1.1', '1.1.2', '1.1.3' and '1.1.4', indicating that the layout of the construction site is established before these specific elements.

Let us now consider the entity "Site Layout" ('1.1.5') of level 3, which is highlighted in Figure 85 and Figure 87, as a constituent of the "Foundations" node of level 2. As we can see in the graph of Figure 86, this structure is able to describe the semantics of the implantation of the building in graphs.

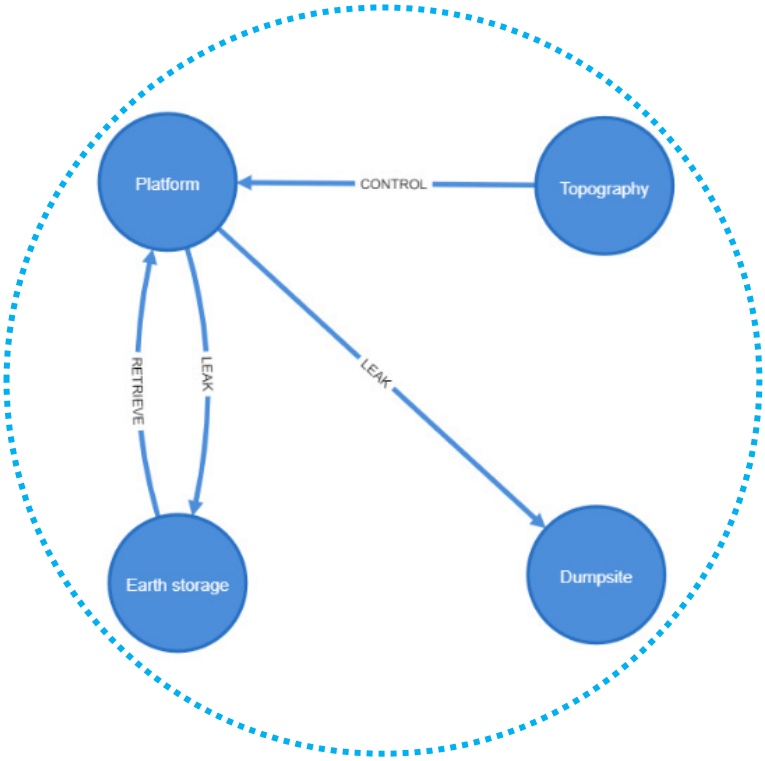


Figure 86 - Level 3 graph "Site Layout" (node '1.1.5*').

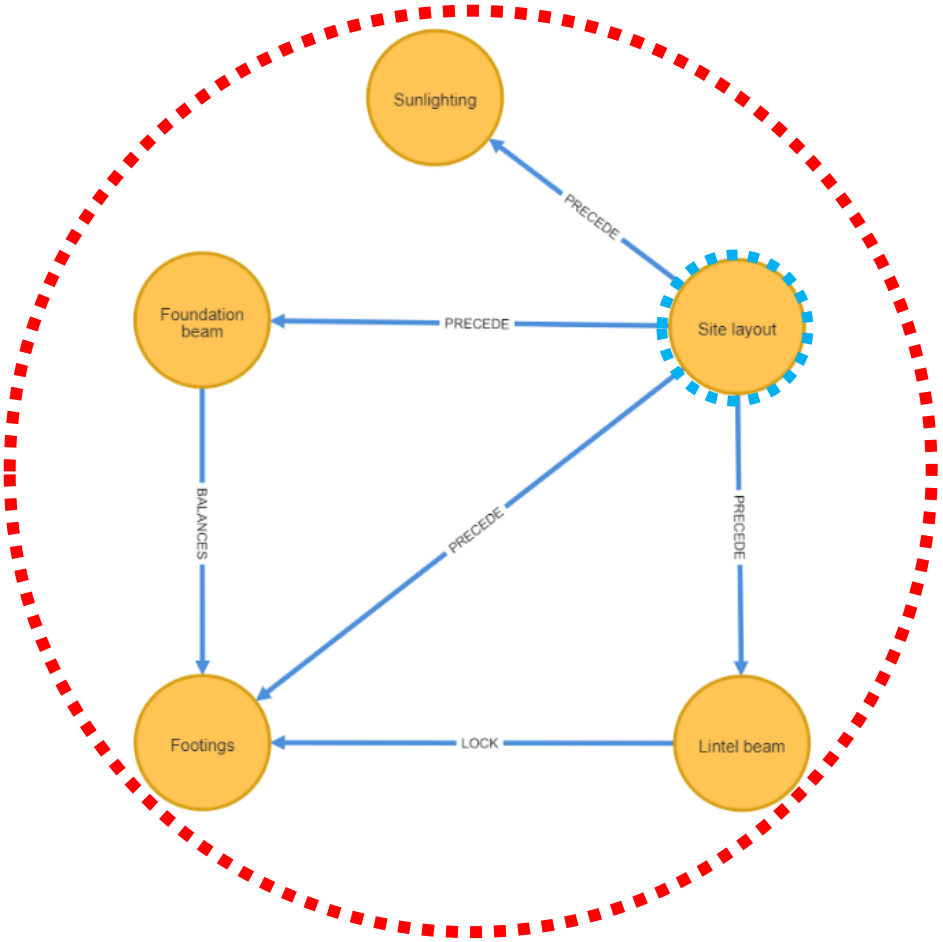


Figure 87 - Site Layout entity (node '1.1.5*') of level 3, as constituent of the foundation node of level 2.

For the "Site Layout" entity (node '1.1.5') of level 3, within the scope of the MLHG we will necessarily have the definition of structures in the form of the adjacency matrix (matriz_adjacencia_G1_1_5), the no-label dictionary, the label-vertex dictionary and the relationship type dictionary.

The structure of these elements is:

Matrix_adjacence__G1_1_5 =

[0 0 1 1]

[1 0 0 0]

[0 0 0 0]

[1 0 0 0]

```
dict_nodes_labels_G1_1_5 = {
    0: '1.1.5.1',
    1: '1.1.5.2',
    2: '1.1.5.3',
    3: '1.1.5.4'
}
dict_vertex_label_G1_1_5 = {
    '1.1.5.1': 'Platform',
    '1.1.5.2': 'Topography',
    '1.1.5.3': 'Dump site',
    '1.1.5.4': 'Earth storage'
}
dict_relationship_types_G1_1_5 = {
    '1.1.5.1': {
        '1.1.5.3': ['LEAKS'],
        '1.1.5.4': ['LEAKS']
    },
    '1.1.5.2': {
        '1.1.5.1': ['CONTROLS']
    },
    '1.1.5.4': {
        '1.1.5.1': ['RETRIEVES']
    }
}
```

After the integration of these structures in the Neo4j database, as illustrated in Figure 86, which presents the node corresponding to the implantation as a graph. The contextual description of the interactions at level 3 is provided, characterizing the relationships that define this entity within the foundation substructure at level 2.

Let's now look at the case of the "Site Layout" ('1.1.5') of level 3. For this node we have an adjacency matrix (matriz_adjacencia_G1_1_5) that represents the connections between the MLHG nodes. This matrix, of 4x4 dimension, shows the relationships between the elements of level 4, in which

each node is identified by a unique label in the dictionary "dict_nodos_rotulos_G1_1_5". For example, node '1.1.5.1' has the label "Platform", node '1.1.5.2' the label "Topography", and so on. Similarly, the relationships between nodes are defined in the dictionary "dict_tipos_relacionamentos_G1_1_5". These relationships establish the interactions between the entities and indicate the type of relationship that exists. For example, node '1.1.5.1' has a 'LEAK' relationship with nodes '1.1.5.3' and '1.1.5.4', indicating the lead to the dump and the land storage site. Node '1.1.5.2' has a 'CONTROL' relationship with node '1.1.5.1', suggesting that the topography exerts control over the platform. Node '1.1.5.4' has a relationship of "RETRIEVE" to node '1.1.5.1', indicating that material retrieval occurs on the platform. From these relationships, it is possible to understand the interactions between the entities of level 3, where the entity "Platform" ('1.1.5.1') plays an important role as a support structure. This structure is associated with leaks, establishing "LEAK" relationships with the entities "Dumpsite" ('1.1.5.3') and "Earth storage" ('1.1.5.4'). On the other hand, the entity "Topography" ('1.1.5.2') exerts control over the platform through a "CONTROL" relationship with the entity "Platform" ('1.1.5.1'). In turn, the entity "Earth storage" ('1.1.5.4') is related to the entity "Platform" ('1.1.5.1'), through a "RETRIEVE" relationship, indicating the recovery of materials on the platform.

Having arrived here, and after explaining the structures of some entities contained in the three hierarchical levels, which complement each other, we now illustrate the graph resulting from the total intra-level integration of the node "Foundations" (Figure 88), which has the vertex label ('1.1') of level 1, to better understand the potential of the proposed MLHG model.

The structure of the MLHG is constituted by:

Matrix_adjacence_FUN_All =

```
[1  0  0  0  0  0  0  0]
[1  0  1  1  1  1  0  0]
[1  0  0  0  0  0  0  0]
[1  0  1  1  1  1  0  0]
[1  0  0  0  0  0  0  0]
[1  0  1  1  1  1  0  0]
[1  0  0  0  0  0  0  0]
[1  1  1  1  1  1  1  0]
```

```
dict_nodes_labels_FUN_All = {
    0: '1.1.1',
    1: '1.1.2',
    2: '1.1.3',
    3: '1.1.4',
    4: '1.1.5.1',
    5: '1.1.5.2',
    6: '1.1.5.3',
    7: '1.1.5.4'
}

dict_vertex_label_FUN_All = {
    '1.1.1': 'Natural lighting',
    '1.1.2': 'Foundation beam',
    '1.1.3': 'Footings',
    '1.1.4': 'Lintel beam',
    '1.1.5.1': 'Platform',
    '1.1.5.2': 'Topography',
    '1.1.5.3': 'Dump site',
    '1.1.5.4': 'Earth storage'
}

dict_relationship_types_FUN_All = {
    '1.1.2': {
        '1.1.3': ['BALANCES']
    },
    '1.1.4': {
        '1.1.3': ['LOCKS']
    },
    '1.1.5': {
        '1.1.1': ['PRECEDES'],
```

```
        '1.1.2': ['PRECEDES'],
        '1.1.3': ['PRECEDES'],
        '1.1.4': ['PRECEDES']
    },
    '1.1.5.1': {
        '1.1.5.3': ['LEAKS'],
        '1.1.5.4': ['LEAKS']
    },
    '1.1.5.2': {
        '1.1.5.1': ['CONTROLS']
    },
    '1.1.5.4': {
        '1.1.5.1': ['RETRIEVES']
    }
}
```

The adjacency matrix of the "Foundations" node, Matrix_adjacence_FUN_All, and its dictionaries obtained through the generate_adjacency_matrix_and_dict function, provide a global view of the practical case, covering the nodes present at levels 1, 2 and 3 and their real relationships. This representation now allows us to understand the connections resulting from the interactions between the elements at different levels of detail, which allows us to have a more comprehensive view of the project.

In the second level of iteration, we find the nodes that represent some fundamental elements, such as "Sunlighting" (1.1.1), "Foundation beam" (1.1.2), "Footings" (1.1.3) and "Lintel beam" (1.1.4). These elements play critical roles in the structure and support of the project. For example, the "BALANCES" relationship between the "Foundation beam" (1.1.2) and the "Footings" (1.1.3) highlights the importance of the balance between these components to ensure the stability of the construction. When we move to the next higher level, we see that the nodes represent elements related to "Platform" (1.1.5.1), "Topography" (1.1.5.2), "Dumpsite" (1.1.5.3) and "Earth storage" (1.1.5.4). These elements are linked to the infrastructure and logistics of the project. For example, the "LEAK" relationship between the "Platform" (1.1.5.1) and the "Dumpsite" (1.1.5.3) suggests the need for material delivery management measures at the dump site.

The analysis of the connections between elements of different levels reveals the importance of integration and coordination between the different aspects of the project. For example, "Platform" (1.1.5.1) controls "Topography" (1.1.5.2), directly influencing the characteristics and configurations of the terrain. This interaction reveals the need to consider the platform as a central point for the adaptation and confirmation of the construction environment.

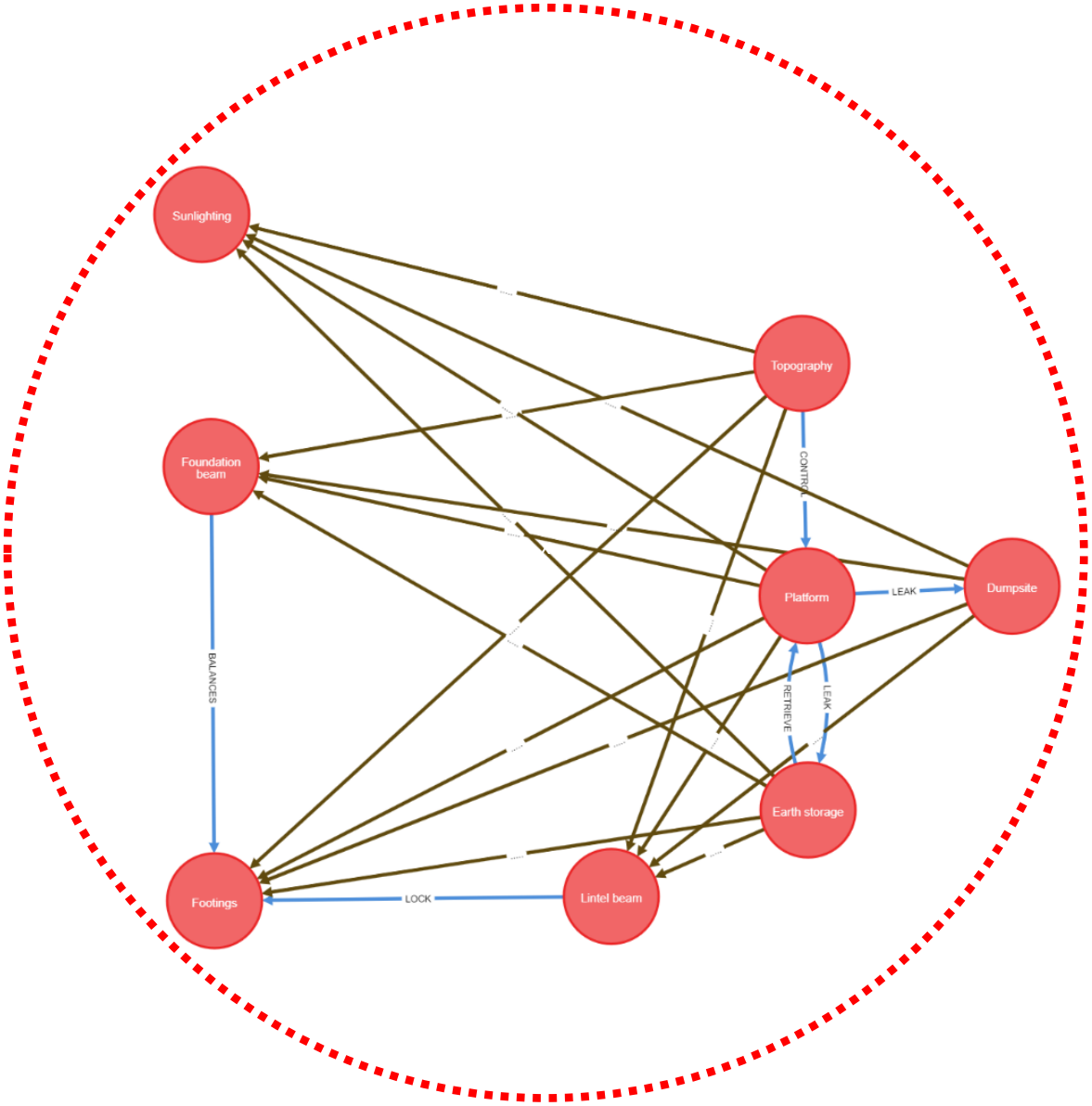


Figure 88 - Graph resulting from the total integration Intra levels of the "Foundations" node.

In the overall context of this case study, the adjacency matrix and the relationships defined in the dict_relationship_types_FUN_All dictionary allow a systemic and integrated view of the connections between the constituent elements of the project, making explicit the dependencies and mutual influences

between them. This holistic perspective facilitates the understanding of the structural and functional complexity of the contract, supporting strategic decision-making based on relational analysis between levels and components. For a more detailed reading of the structure, it is important to observe how the MLHG model is distributed along the different levels of abstraction:

- At level 1, entities such as Foundations, Resources, and Technical teams are represented as master nodes, while their relationships—defined in the adjacency matrix—reflect functional interactions such as "allocate to," "advise about," and "validate." These relationships are essential to represent the operational and informational interdependencies between the fundamental components of the system;
- At level 2, the "Foundations" entity is decomposed into its structural sub-elements, such as "Sunlighting", "Foundation beam", "Footings" and "Lintel beam", each of them with specific relationships that delimit the functional and physical interactions within the set;
- In turn, at level 3, the "Site layout" sub-element is detailed in even more granular components, such as "Platform", "Topography", "Dumpsite" and "Earth storage", whose relationships — for example, "LEAK" and "CONTROL" — represent dependencies on execution, supervision and environmental impact between the elements.

In summary, the MLHG model, when applied as a conceptual schema of a BDOG, evidences a hierarchical, relational and integrative approach, capable of accurately representing complex construction systems at multiple levels of detail. Figure 89, presents a complete instance of MLHG, converted to a graph from the BDOG schema represented in the Figure 83. In this visualization, four levels of abstraction are integrated, evidencing a densely interconnected structural network, whose relationships were computed by the MLHG support algorithms — connections that would hardly be identifiable through conventional modeling methods. Two virtual vertices are also highlighted, $V_{1.1}$* (corresponding to Foundations) and $V_{1.1.5}$* (Deployment), which behave as hyperedges at different levels of abstraction of a generic hypergraph, reinforcing the model's ability to represent high-order relations and multi-scalar hierarchical dependencies within the construction system.

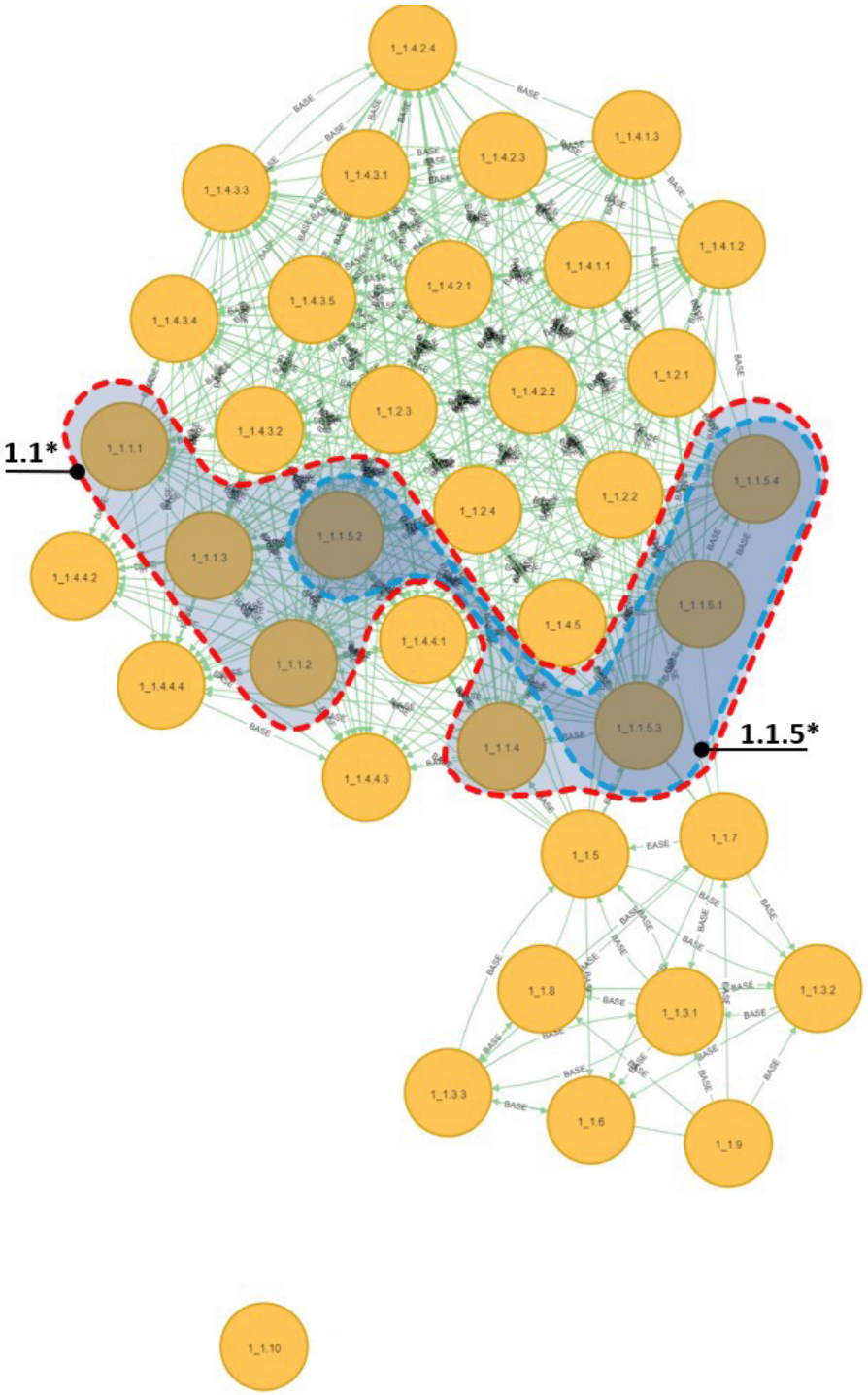


Figure 89 – The structure of an MLHG instance transformed into a graph.

Figure 90 explicitly presents the aggregation phases that describe the progression of a generic instance of the MLHG, in the context of the BDOG schema applied to civil construction. Four phases of aggregation are represented, distributed over four levels of abstraction, in which each level reflects an

increasing degree of detail and integrates the elements corresponding to the ten phases of model development.

In each phase, the individual nodes correspond to elementary entities (such as processes, resources, or physical components), which progressively aggregate to form groups of interconnected entities, representing more complex functional or constructive units. These aggregation steps constitute the evolutionary stages of MLHG, in which the basic components combine to perform integrated and wide-ranging functions within the modeled system. This hierarchical and cumulative structure allows for a controlled and staggered analysis of the contract, offering a coherent view of the evolution of relationships and interdependencies throughout the distinct phases of design.

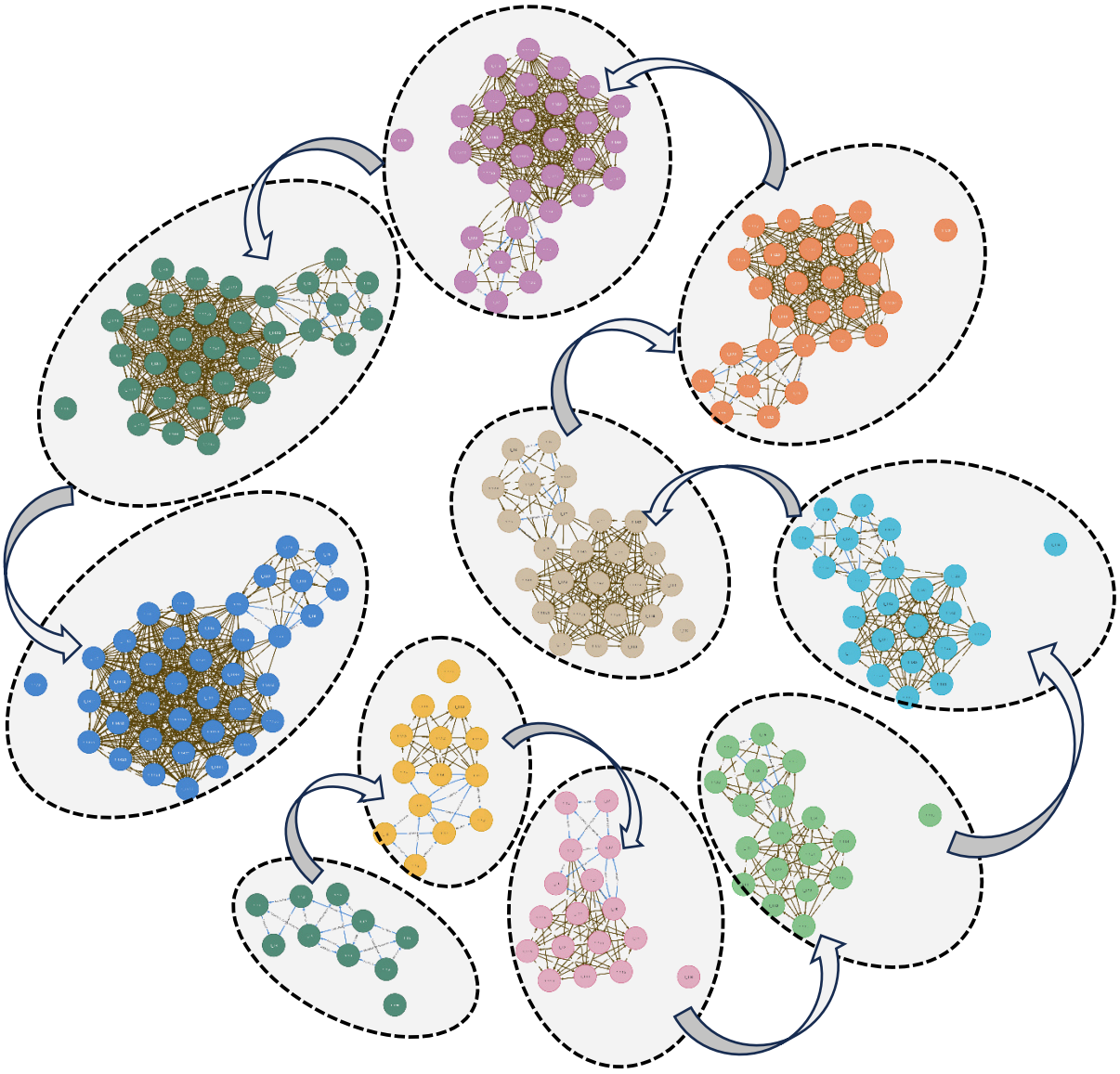


Figure 90 - Evolution of the complex structure of an MLHG.

The edges that connect the entities explicitly represent the functional relationships and structural dependencies between the different components of the system. Unlike the linear connections observed in conventional planar graph models, the edges of MLHG capture the topological complexity and multidimensional nature of interactions, reflecting not only the direction of the relationship, but also its intensity, hierarchy, and functional semantics. Figure 90 it exemplifies, therefore, an advanced modeling approach, in which both the topology and the typology of the edges are determinant to understand how entities relate, influence and condition each other within the construction system. It is important to highlight that, as the levels of abstraction deepen, there is an exponential multiplication of interconnections, configuring a densely intertwined network that is difficult to perceive immediately. However, this apparent entanglement is a direct consequence of RI, according to which higher-level entities derive, are governed by, or structurally shaped by lower-level entities. This characteristic gives to a MLHG a hierarchical depth, allowing it to rigorously represent complex construction processes, in which local and global dependencies coexist and influence each other.

A closer look at civil engineering highlights the relevance of MLHG in the representation of complex systems marked by multiple levels and rigidly conditioned by technical, logistical, economic and legal aspects. The possibility of simultaneously modeling structural interdependencies, flows between teams and suppliers, and contractual and institutional relationships, demonstrates the robustness of the method in dealing with different degrees of abstraction, without losing coherence. Just as in a regatta, in which collective success depends on the harmonious articulation between various agents and resources, also in civil construction the viability of an enterprise is based on an interconnected network of actors, decision-making processes and construction stages. However, unlike these two domains — sport and construction — in which organizational links tend to be relatively stable and delimited by clear rules, there are contexts in which complexity arises mainly from the strategic and competitive dynamics between the parties involved. This is precisely the scenario of dynamic electronic bids that take place in a digital auction, in which interactions are not structured by the materiality of the works or the predictability of organizational routines, but rather by adaptive decisions, real-time dispute, and organizational patterns, which emerge spontaneously and volatilely. This third case study closes the validation cycle of MLHG, extending its application to a domain in which complexity results directly from the strategic interaction between multiple decision-makers under conditions of high uncertainty. We will look at this case in more detail in the next section.

5.3 Bidding in Digital Auctions

The civil construction model allowed to demonstrate how an MLHG model is capable of dealing with complex systems, in which technical and institutional interdependencies are articulated around stable standards. However, not all organizational contexts present this predictability. In some domains, complexity depends less on the materiality of structures and the linearity of processes, and more on the volatility of strategic interactions between multiple actors. Knowing this, we selected the third application case: bidding in digital auctions. In this case, the organizational logic is replaced by a real-time competitive dynamic, in which each decision (e.g., an offer, a withdrawal, or a price change) can immediately alter the state of the system. Unlike sports or construction, where the structure and rules are established in advance, digital tenders are characterized by having associated accelerated and unpredictable information flows, and by emerging patterns that are redefined with each interaction.

The increasing digitalization of auction processes implies the management of their information in real time, which poses serious challenges in the formal representation and processing of the data itself. This reality requires the adoption of innovative methods that can accommodate the diversity, speed and complexity of the elements involved in these processes. Currently, most auctions take place, in fact, in digital environments, marked by high competitiveness and procedural dynamics. Auction houses face the challenge of managing a vast and heterogeneous set of data — ranging from the goods to be auctioned, to participants and detailed bid records. This complexity justifies the existence of applications to support the processes related to the digital management of auctions, which can deal with the new demands of the market and contemporary information systems in this domain (Hendricks, Sorensen, and Wiseman 2024).

The digital paradigm has exponentially increased the multiplicity and speed of interactions between the elements of the auction process, giving rise to true ecosystems of complex data. MLHG can provide an effective response by offering a hierarchical and structured representation, which allows you to store the data and represent the dynamics underlying a digital auction. An approach such as that of MLHG is particularly relevant in a sector characterized by high volatility and constant evolution of information, such as digital auctions. The multilevel representation facilitates both the global (macro) analysis of processes and the specific (micro) analysis of individual elements as outlined by (Deng and Zhou 2025). An MLHG naturally adapts to these two types of representation, providing adequate support for consultations and operations of a digital auction.

The definition and construction of the hierarchy of an MLHG that was designed to model the functioning of an auction house requires a methodological approach, which carefully considers the specific nature of the entities involved in the auction process, as well as the various interactions that occur between them. Table 13 presents the set of entities integrated in the proposed model, organized according to the levels of abstraction identified as relevant (Level 1 and Level 2), The table also included two subsets of entities, designated as input sublevel 1 and input sublevel 2, whose structure is formally supported by two hyperedges, that is, two virtual vertices.

Table 13 - Characterization of an MLHG hierarchy.

Level 1	**Entry**	**Level 2**	**Sub level 2**	**Level 3**	**Sub level 3**
Auctioneer	V_1	Enforcement Solicitor	$V_{1.1}$		
		Auctioneer	$V_{1.2}$		
		Seller	$V_{1.3}$		
		Auction	$V_{1.4}$	Participant	$V_{1.4.1}$
				Lot	$V_{1.4.2}$
				Spear	$V_{1.4.3}$
		Goods	$V_{1.5}$		
		Documentary Process	$V_{1.6}$	Appraiser	$V_{1.6.1}$
				Designer	$V_{1.6.2}$
				Lawyer	$V_{1.6.3}$
				Expert	$V_{1.6.4}$
				Requester	$V_{1.6.5}$
		Buyer	$V_{1.7}$		
		Process Manager	$V_{1.8}$		

To start the construction of the hierarchy that integrates all the data elements involved in an auction process, it is essential to define and characterize each vertex (entity) and the relationships it establishes with the other vertices of the system. This characterization should consider the different levels of abstraction and the way each entity can be organized through virtual vertices, usually responsible for structuring the existing complex interactions. The resulting hierarchical structure should offer a

comprehensive, coherent and faithful representation of the auction house's functional ecosystem, supporting a model capable of supporting dynamic and in-depth analysis of the information processed throughout the auctions.

For the practical application case, we have developed the hierarchy presented in Figure 91, in the form of a tree. This structure reflects the functional dimensions and stakeholders identified within a specific requirements analysis process, conducted in the real-world context of the application. In this hierarchical configuration, two virtual vertices stand out, "Auction_$V_{1.4}$" and "Documentary_Process_$V_{1.6}$", which assume structuring functions in the MLHG model, by organizing and interconnecting the vertices that represent the auctions and the associated document processes.

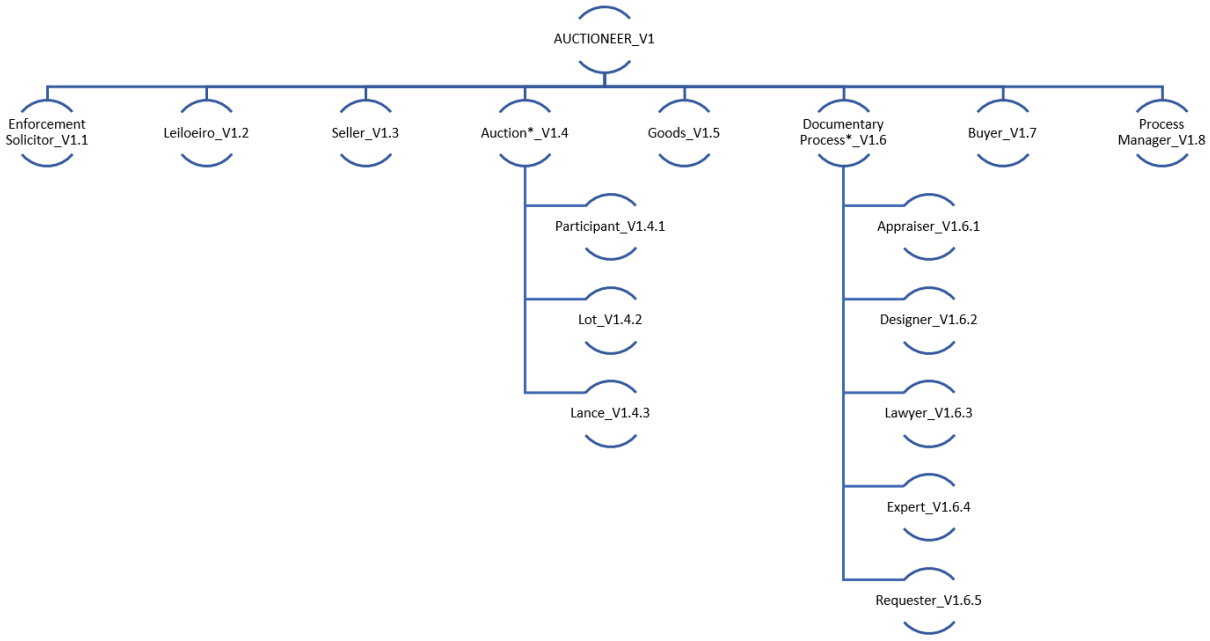


Figure 91 - The level tree (Ai,n) of the auctioneer application.

The adjacency matrix of the MLHG model, presented in Table 14, reveals how the virtual vertices interact with the various regular vertices, outlining the flow of information and the structural dependencies between the represented entities. Understanding these relationships contributes significantly to the operational efficiency of auction houses, as it allows modeling, with a high degree of detail, all the data elements involved in the process.

Table 14 - The Adjacency Matrix of the MLHG.

Vertices	**Auction*_$V_{1.4}$**	**Documentary_Process*_$V_{1.6}$**
Auctioneer_$V_{1.2}$	0	0
Seller_$V_{1.3}$	0	0
Participant_$V_{1.4.1}$	1	0
Lot_$V_{1.4.2}$	1	0
Bids_$V_{1.4.3}$	1	0
Assets_$V_{1.5}$	0	0
Evaluator_$V_{1.6.1}$	0	0
Designer_$V_{1.6.2}$	0	1
Lawyer_$V_{1.6.3}$	0	1
Expert_$V_{1.6.4}$	0	1
Requester_$V_{1.6.5}$	0	1
Buyer_$V_{1.7}$	0	1
Process Manager_$V_{1.8}$	0	0

According to the MLHG of Figure 92, the virtual vertex "Auction*_$V_{1.4}$" represents the overall context of the auction, aggregating the essential entities that participate and exert direct influence on the dynamics of the process, which are, namely:

- "Participant_$V_{1.4.1}$", which represents individuals or organizations with an active interest in participating in the auction, either through the submission of bids or for the purpose of acquiring goods. Each participant brings dynamism and competitiveness to the auction session;
- "Lot_$V_{1.4.2}$", which represents a specific grouping of goods, auctioned as a unit. Each lot can contain different types of items, whose association is designed to maximize the interest of participants and enhance the value of joint selling;
- "Bids_$V_{1.4.3}$", which receives the bids submitted by the participants with a view to the acquisition of the goods at auction. This entity is intrinsically linked to the auction itself, reflecting the degree of competition and the competitive behavior of the participants.

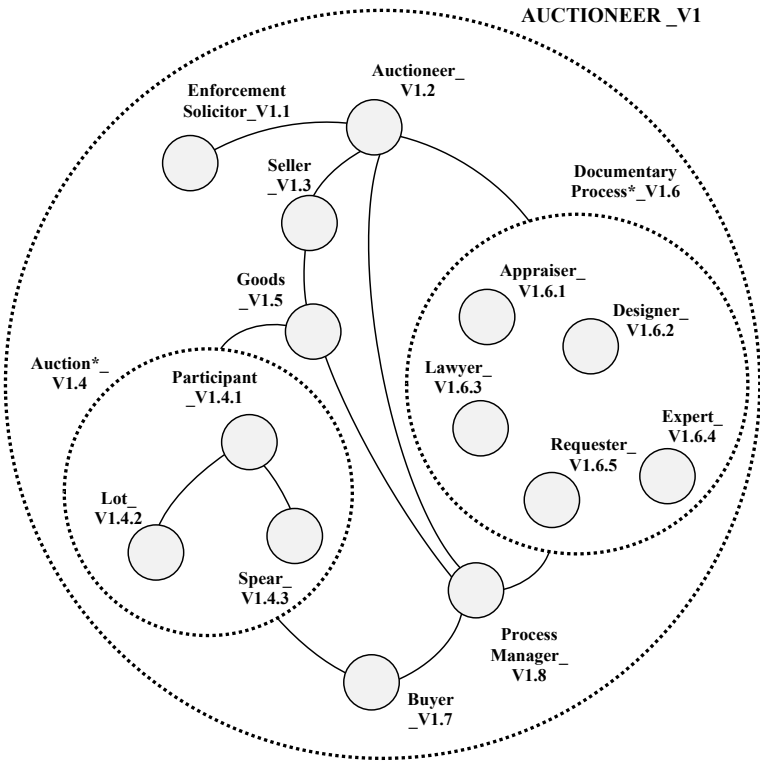


Figure 92 - The MLHG of the auctioneer.

The virtual vertex "Auction*_$V_{1.4}$" plays a central role in shaping the auction, by bringing together the participants, the lots, the bids and, by association, the auctioned goods. This vertex acts as an instance of interaction between the various elements involved in the process, reflecting its functional articulation and directly conditioning the success of the transition process, in which the participants become buyers. The bids submitted ("Bids_$V_{1.4.3}$") are directly related to specific goods at auction ("Assets_$V_{1.5}$"), establishing an essential link between the intention to acquire and the specific objects of the transaction. This dynamic creates a competitive cycle where participants compete for ownership of the desired goods, becoming buyers ("Buyer_$V_{1.7}$") whenever they purchase a lot. This relationship shows the passage from a state of participation to a state of effective purchase, which is a fundamental element in the logic of the operation of an auction house. In turn, the virtual vertex "Documentary_Process*_$V_{1.6}$" plays a structuring role in the organization and legal compliance of the auction, covering activities of evaluation, design, legal validation and coordination of administrative processes. The articulation with the auctioneer and the process manager is decisive to ensure the rigor and effectiveness of the operation, involving the following entities:

- "Evaluator_$V_{1.6.1}$", which represents a specialized professional responsible for the valuation of the goods before they are put up for auction. Its estimate directly influences the market value of the items and serves as a reference for the bids of the participants;
- "Designer_$V_{1.6.2}$", which is a technical collaborator in charge of the design, presentation or visual planning aspects of the goods. Its intervention contributes to the aesthetic attractiveness and the perception of value of the lots;
- "Lawyer_$V_{1.6.3}$", which represents a jurist in charge of ensuring legal compliance with the entire auction process, ensuring compliance with applicable contractual, regulatory and regulatory requirements;
- "Expert_$V_{1.6.4}$", which is a technical expert in specific areas of knowledge, whose function is to provide opinions or validations that add technical credibility to the process and the characteristics of the auctioned goods;
- "Requester_$V_{1.6.5}$", which models an employee responsible for the documentary and logistical preparation of the goods at auction, acting in support of the seller and ensuring the correct presentation and framing of each item.

Coordinated interaction between these agents and the process manager ("Process_Manager_$V_{1.8}$") is vital to ensure the fluidity, legality and overall efficiency of the auction. The MLHG structure adopted allows us to represent this scalable relational complexity, enhancing detailed operational analysis and better-informed decisions. The interaction between the auctioneer ("Auctioneer_$V_{1.1}$") and the documentary specialists is essential to ensure both legal compliance and the operational efficiency of the auction process. In this context, the process manager ("Process_Manager_$V_{1.8}$") assumes a central role in the supervision and global coordination of activities, articulating directly with specialists in the legal, technical and documentary areas. Its performance ensures that all stages take place in a structured, efficient manner and in strict compliance with legal and operational requirements, functioning as an integrating link between the various stakeholders and instances of the process. This coordination is particularly critical in a dynamic digital environment, where procedural robustness and traceability of decisions are decisive factors for the credibility of the auction platform.

Figure 93 presents the final configuration of the MLHG model after the application of the RI process. In this diagram, the various data elements are represented through vertices — regular and virtual — as well as their respective structural and functional relationships. The organization of the model shows the

interdependence between the various components of the auction ecosystem, such as participants, lots, bids, document-related processes, and other entities involved. This visualization reinforces the relevance of the MLHG model as a relational modeling solution for complex digital environments, highlighting its ability to dynamically structure, represent and adapt the information flows characteristic of digital auctions. The resulting topology concretely illustrates the gains obtained from the application of hierarchical and inheritance mechanisms in the formal representation of real systems, because it allows for the articulation of different levels of detail under a coherent structure, where the relationships between entities are preserved and propagated automatically throughout the hierarchy. Thus, inheritance ensures semantic consistency and functional continuity between levels, while hierarchical organization reduces redundancies and reinforces traceability between processes. Together, these mechanisms make the model more expressive, scalable and adaptive, capable of accurately representing the dynamic complexity of real systems, such as digital auctions.

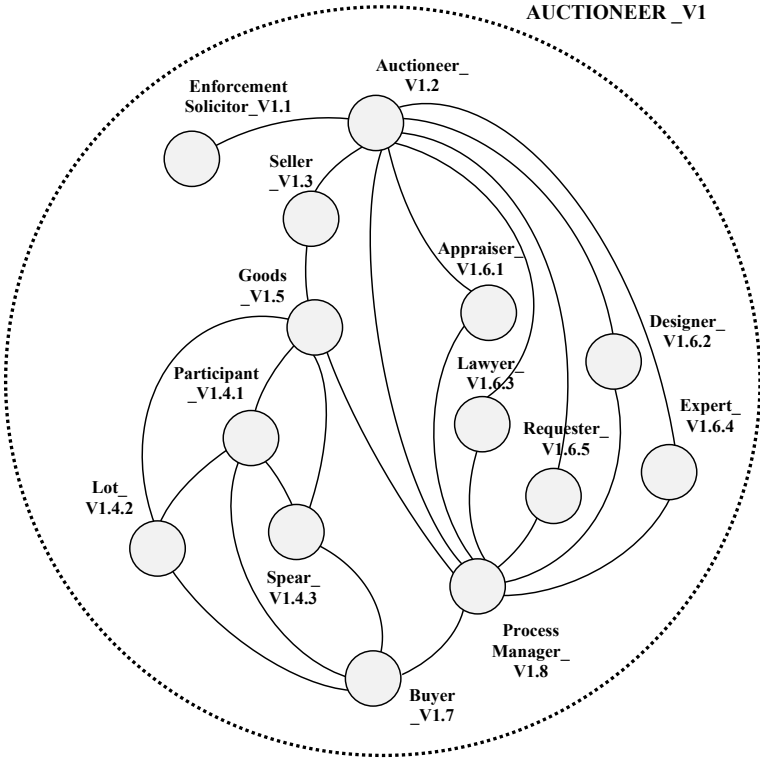


Figure 93 - The auctioneer MLHG configuration after the application of the heritage process.

Based on the developed model, we conclude that MLHG allows the representation of data elements and their interrelationships through a multi-level hierarchical structure, enabling the explicit definition of

dependencies between abstract layers and the coherent aggregation of differentiated components under a common structural framework. In the context of digital auctions, characterized by intricate and interdependent relationships between multiple actors and functional entities, MLHG-based modelling has proven to be particularly effective. Its multilevel architecture, combined with the use of virtual vertices as aggregating entities, gives the model a superior semantic expressiveness, allowing it to represent both the transactional relationships — between participants, lots and bids — as well as the documentary, legal and operational dimensions that support the integrity of the process. This capability results from the fact that MLHG simultaneously integrates several layers of abstraction, rather than isolating entities or processes. The mechanisms of RI and semantic coherence ensure that each level inherits structural and semantic properties from the top level, preserving the overall consistency of the system. In parallel, the combination of the adjacency matrix with the label and relationship dictionaries enables dynamic queries, simulations, and predictive analysis, allowing us to understand and anticipate behavior patterns in digital ecosystems subject to continuous data updating.

When compared to the case of civil construction, the conceptual plasticity of MLHG stands out. While in the field of engineering, the emphasis is on structural stability and regulatory traceability, in digital auctions, adaptability, agility and the management of unpredictable information flows are privileged. This duality proves the versatility of the model, which is suitable for both physical and regulated systems and complex and dynamic digital ecosystems. So, this third application case confirms that MLHG is more than a formal representation model: it is a useful instrument for managing and analyzing complex systems, offering a robust methodological basis for the integration of different domains under a coherent schema of abstraction, inference and semantic interoperability.

5.4 MLHG Evaluation and Impact

The evaluation of the MLHG model, in combination with the logic of BDOG, was conducted from three case studies that cover distinct domains, but convergent in their high structural, semantic and organizational complexity: sport (with a focus on rowing), civil construction (through the modeling of a reinforced concrete structure) and dynamic electronic auctions. Each of these contexts was selected for its ability to test the model in different analytical dimensions — hierarchical relationships, procedural flows, temporal dynamics and variability of entities — allowing to evaluate the robustness, generality and expressiveness of MLHG in the face of systems of a diverse and interdependent nature.

In the field of sport, the choice of rowing was not fortuitous: it is a sport that symbolizes in an illustrative way the functional interdependence between human, material and institutional agents. The performance of a crew results not only from the sum of individual physical capacities, but from the temporal coordination and collective synchrony between the athletes, the helmsman, the equipment (boat, oars, clamping mechanisms) and the operational strategies defined by coaches and technical directors. MLHG proved to be particularly effective because it allowed the simultaneous modeling of local and global relationships — such as the synchronization between two rowers (micro level) and the competitive position of a club in the championship (macro level) — while maintaining semantic consistency between hierarchical levels. This is achieved through inter-vertex RI, which propagates properties and states from a lower level (individual athlete) to a higher level (team or club), without loss of structural coherence.

In addition, the model allows navigation between layers of abstraction through graph queries (e.g., path traversal between vertices), which enables upward and downward analysis—from collective performance to individual contribution, and vice versa. This operational bidirectionality demonstrates how MLHG integrates detail and wholeness seamlessly, capturing the cooperative dynamics that characterize high-performance sport.

Thus, by providing an integrated view of the relationships between the nodes individual, team, and institution, MLHG demonstrated its ability to represent multilevel organizational systems, in which overall performance results from the articulation between individual effort, collective coordination and institutional structure.

In the second case study, which focuses on civil construction, the construction model was analyzed as a representative technical and organizational system conditioned by physical, normative, and logistical factors. The graph-driven approach proved to be more appropriate than the classical relational model, whose rigidity tends to fragment composite entities and generate artificial dependencies. In MLHG, entities are represented directly, relating to each other through edges and hyperedges that accurately express functional and constructive dependencies. In this way, elements such as footings, columns, beams, and slabs can be modeled as interdependent components of a construction network, whose cohesion depends on the assembly sequence, the materials used and the structural function associated with each component. The inclusion of temporal and operational properties — such as execution time and construction group — allows describing not only the structural state but also simulating the temporal and

financial evolution of the work. It is also possible to correlate the accumulated cost with the order of assembly of the elements, identifying planning patterns and critical points of the construction process.

The third case study introduced an additional dimension of temporal and informational complexity. Unlike previous cases — characterized by stable entities and constraints — digital auctions require dealing with constantly changing data. MLHG made it possible to simultaneously represent permanent entities (products, bidders, rules) and transitory entities (bids, phases, contractual links) that transform throughout the process. The use of vertices and top-level edges made it possible to integrate multiple parameters into a single relational structure, avoiding the fragmentation typical of relational approaches. The comparative analysis of the three case studies showed the structural consistency and adaptability of the model. In sport, complexity comes from the articulation between human and institutional agents; in construction, of the physical and normative interdependence of structural elements; and in digital auctions, of the informational dynamics in real time. In all contexts, the MLHG adjusted to the specificities of each domain without the need for structural reconfiguration, supported by its hierarchical organization and the contextual interpretation of relationships.

The evaluation carried out allows us to conclude that the strength of MLHG lies not only in its topology, but in the ability to attribute contextual properties to relations and to operate in a coherent way between different levels of abstraction. This distinguishes it from relational approaches and other graph-based models, consolidating MLHG as a formal representation framework for systems in which entities and relations evolve simultaneously in time and space. In conclusion, the systematic evaluation conducted in the three domains confirms the validity and generality of the model. In rowing, MLHG captured the interplay between individual performance, collective cooperation, and organizational framework; in construction, he modeled the structural and temporal interdependence of physical elements; and in electronic auctions, it represented dynamic processes of high relational complexity. This convergence of results demonstrates that the model is methodologically consistent and conceptually extensible, offering a solid basis for future interdisciplinary applications and theoretical extensions, namely in the field of automatic inference, semantic annotation and integration with decision support systems.

6 Conclusions and Future Work

The realization of this thesis allowed carrying out an in-depth examination of the theoretical foundations and practical applications of the representation models of complex systems, with emphasis on MLD and MLHG. After a detailed exposition of the basic concepts, mathematical bases and methodologies proposed, now we can evaluate the results achieved and formulate a reflection that articulates theory, experimentation and practical implications. Far from being just a review or repetition of the contents presented throughout the thesis, this chapter presents a critical synthesis. Its objective is to articulate the contributions developed, relating to theoretical formalism and the identified application domains. This is a final reflection on the path taken, which highlights the transition of MLD and MLHG models from the conceptual sphere to use in real contexts, such as the management of civil construction projects, multi-scale coordination in sport (rowing) and the dynamic mechanisms of electronic auctions.

This section is organized to encompass the main actions undertaken throughout this research. We begin with a synthesis of the investigation, highlighting the key findings. Next, we discuss the application cases examined, which enabled us to demonstrate the capability of the MLHG model in representing heterogeneous domains. We also analyze the limitations identified, the challenges encountered, and the scientific and technical contributions resulting from this work. Subsequently, we present the practical implications and illustrate how the developed models were applied. We then outline directions for future research. Finally, we provide concluding reflections that close the chapter, emphasizing the overall relevance and impact of this thesis for both academic and professional practice.

6.1 A Summary of the Contributions

The work of this thesis was motivated by the conviction that computing, engineering and information management increasingly face great challenges in modeling systems whose complexity exceeds the capabilities of traditional structures. This complexity results from the multifaceted nature of contemporary systems, in which physical, digital, social, and organizational dimensions interact interdependently. Throughout this work, it was argued that the answer to this challenge was through significant conceptual evolution in the field of formal representation, so that structures such as MLD and MLHG could be developed. The fundamental theoretical contribution of this research lies in the formulation of these two

models, as hierarchical extensions of graphs and hypergraphs, which can translate complex interdependencies at different levels of abstraction. MLD introduces a layer-based model, in which each vertex and edges can be understood not as an isolated entity, but as part of a chain of interdependent relationships, structured at multiple levels of granularity. This approach allows, for example, the same entity (such as a pillar in a reinforced concrete structure) to be represented simultaneously at the local scale, associated with a restricted set of relationships, and at the global scale, where it is integrated into a broader system of interactions.

The MLHG proposal advances the hypergraph paradigm by incorporating hierarchical organization into the representation of higher-order relationships among multiple entities. This integration results in a formal structure capable of representing complex relationships at different levels of abstraction while preserving semantic coherence across layers. The explanation of non-binary relations distributed by different levels and the demonstration that they can maintain logical and formal consistency are, therefore, a central theoretical contribution of this thesis. While a traditional graph captures only binary relationships between pairs of vertices, an MLHG allows modeling compound hyperlinks, involving multiple entities simultaneously, and framing them at a higher hierarchical level, where they articulate with broader dependencies. This conceptual advance is part of the recent scientific debate on the need for models capable of dealing with high-order interactions, a topic widely discussed in areas of complex systems, neural networks, or computational social sciences (Battiston et al., 2020; Lambiotte et al., 2019). In these works, it is argued that classical graphs are no longer sufficient to represent phenomena whose structure is not exhausted in individual pairs. By formalizing the MLHG, this thesis contributes to this line of research, offering a mathematical structure oriented to the hierarchical representation of hyperlinks, which responds to the limitations identified in traditional modeling.

Another significant theoretical contribution is the interplay between completeness, coherence, and transitivity across the multilevel structures of MLD and MLHG. The proposed models were designed so that the generated representations not only capture entities and relationships but do so in a verifiable and semantically consistent way. In the case of MLD, it was demonstrated that vertical transitivity allows tracking dependencies between different levels of abstraction, without loss of meaning. Thus, a dependency established at a local level is automatically preserved when the system is observed at a more global level. On the other hand, in an MLHG, relational coherence comes from the possibility of structuring hyperlinks without generating redundancy or contradiction, a relevant aspect when the system involves many interdependent entities. The relevance of this theoretical contribution is also manifested in the

proposal of a topological representation of MLD and MLHG, developed in this work as a complementary instrument to graph representations. By establishing a spatial and ordinal arrangement of relationships, this approach allows formalizing functional dependencies and relationships, supporting the mathematical analysis of properties such as connectivity, density, or hierarchical depth. Such formalization brings MLD and MLHG closer to the status of classical graphs and hypergraphs as objects of mathematical and computational study, consolidating them as structures suitable for algorithmic implementation.

It is important to underline that MLD and MLHG are not presented as exclusive alternatives to existing models, but as necessary extensions to address scenarios in which traditional graphs and hypergraphs are insufficient. These remain adequate for well-defined problems of low relational complexity. However, in systems characterized by strong heterogeneity, multiple scales of observation and interdependent information layers, MLD and MLHG proved to be conceptually more adjusted, precisely because of their ability to preserve semantic consistency in contexts of high structural complexity.

6.2 Methodological Integration

This work extended the theoretical formulation of MLD and MLHG by also exploring the practical implementation of these models. The interdisciplinary approach adopted was essential to ensure that the proposed ideas did not remain confined to a vague conceptual notion, but that they could be materialized, manipulated and examined in concrete computational environments, to meet the current challenges of big data analysis and systemic modelling. A central element of this integration was the use of Neo4j, one of the most widely used graph database platforms in academic and industrial contexts. The choice for Neo4j was mainly due to three reasons: (i) the robustness of its engine for storing and interrogating data stored in graphs; (ii) its language, Cypher, declarative and intuitive for graph manipulation; and (iii) the extensive community of users and the available documentation, which ensured the necessary technical support for the exploitation of the MLHG that were developed (Robinson, Webber and Eifrem 2015). Within the scope of this work, Neo4j played a key role, as it allowed MLD and MLHG to be translated into concrete digital structures, which could be archived, consulted and visualized in an interactive way. However, Neo4j, as a graph-oriented database, has revealed some limitations to deal directly with hypergraphs and their multilevel extensions. To fill this gap, it has become essential to use Python libraries, especially *NetworkX* and *HyperNetX*, widely recognized in the literature for enabling flexible

manipulation of graphs and hypergraphs in a dynamic way. *NetworkX*, as a consolidated library for Python graphs, provided a versatile framework to create the prototypes we developed for MLD, allowing us to program insertion, removal and analysis operations at different hierarchical levels in a fluid way. *HyperNetX*, being a more recent library, but specialized in the manipulation of hypergraphs, was fundamental for the implementation of higher-order relationships, sustaining the transition from MLD to MLHG. The bridge between the theoretical formalization of MLHG and its operationalization in graph databases was possible using the *py2neo library*, which acts as middleware between the Python language and the Neo4j system. Allowing functions and objects developed in Python to be automatically translated into nodes and edges in Neo4j, to preserve the coherence between the conceptual formulation and the computational representation. In this way, MLHG could be manipulated in Python, taking advantage of the flexibility of a scientific prototyping ecosystem and its visualization and analysis in Neo4j.

Another relevant methodological dimension was to consider matrix representations to complement the graph approaches we performed. While the libraries mentioned above allowed the manipulation of MLD and MLHG in terms of vertices, edges and hyperarcs, the matrix formulation offered an additional formalism. Thus, the incidence and adjacency matrices, adapted for multilevel and higher-order relationships, were implemented using the *NumPy* and *Pandas* libraries in Python, which supported the quantitative analysis of connectivity, density, transitivity, and hierarchy. This step proved essential to demonstrate that MLD and MLHG are not just visual or narrative concepts, but formal structures. The methodological integration also included visualization, a relevant aspect when dealing with complex systems. The *Matplotlib* and *Plotly* libraries were chosen to provide the mechanisms for interactive representation of multilevel graphs and hypergraphs. Visualization fulfilled two central roles: (i) to support the validation of the internal coherence of MLD and MLHG, as it allowed to inspect higher-order hierarchical relationships immediately; and (ii) enable more intuitive communication of results, facilitating dialogue with specialists from different areas. As Keim et al., n.d. pointed out, visualization is not only a presentation tool, but also a method of exploratory analysis that reveals patterns not evident in structures of great complexity.

Another important point to highlight is that the proposed methodology was not limited to the isolated application of each tool but also promoted a synergy between them. For example, while Neo4j was used to make persistent storage and support declarative queries, Python served as a space for prototyping and algorithmic manipulation, and matrix representations as support for performing quantitative analysis. This integration created an integrated architecture of complementary tools, in which

each layer reinforced the others, aligning with the very logic of MLD and MLHG as hierarchical and interconnected structures. Another relevant challenge is related to the visualization of higher-order structures. The topological complexity of MLHG requires more robust and expressive visual representation methods, capable of making non-binary relationships organized at multiple levels of abstraction intelligible, something that exceeds the capabilities of traditional graph visualization tools.

Despite these limitations, the methodological integration carried out in this thesis constitutes a substantive methodological contribution. It was demonstrated that it is possible to operationalize MLD and MLHG through an infrastructure composed of consolidated (Neo4j, NetworkX, NumPy) and emerging (HyperNetX, py2neo) tools, resulting in a robust and reproducible experimental environment. This integration not only reinforces the practical validity of the models but also creates conditions for future research to explore their application in large-scale contexts, such as social networks, biological systems, and critical infrastructures. In fact, the methodological conjunction of tools and technologies demonstrated that deep neural networks and generalized hypergraphs are formally solid and computationally viable structures, being able to move from theoretical space to applied practice. The merit of this approach lies precisely in establishing a bridge between two worlds: that of mathematical abstraction and that of technological implementation. Such a bridge not only validates the proposed models but also paves the way for such models to be adopted in real environments, contributing to the analysis and management of complex systems in multiple spheres of knowledge.

6.3 Discussion of Application Cases

The validation of the MLHG model, carried out through three different application case studies areas — rowing, civil construction and digital auctions — enabled a cross-sectional analysis of the robustness, generality and practical applicability of the proposed approach. The diversity of these contexts proved to be particularly relevant, as it allowed us to verify the performance of the model in areas of a structural, functional and procedural nature, each marked by specific and varied technical requirements.

In the first application case, dedicated to competitive rowing, we used MLHG in the modeling of entities and relationships characteristic of this sport, including athletes, boats, events, courses and environmental conditions. This context presented a peculiar challenge: the coexistence of human entities (athletes), technical artifacts (boats and equipment) and contextual factors (wind conditions, water and competitive regulation), which required a sophisticated model to represent its complexity. The social

nature of rowing sport, in which individual performance is articulated with the collective coordination of the crew, and with environmental variables, has become an appropriate scenario to test the ability of MLHG to simultaneously represent multiple levels of abstraction. Thus, it was possible to model, for example, the interdependencies between the training of certain athletes and the performance of the boat in different phases of the competition, inferring relationships that would be difficult to represent in traditional relational models. The flexibility of the MLHG proved to be fundamental to capturing both the static dimension (team structure, boat configuration) and the dynamic dimension (performance variations as a function of external and contextual conditions), approaching the requirements of contemporary sports analysis (C. C. Chen et al. 2023).

The second application case, focused on civil construction, explored the planning of a construction, involving various types of elements, such as fundraising or the use of materials in construction. This area has revealed the potential of holistic planning to represent complex socio-environmental systems, characterized by interdependent relationships and clearly defined synergistic functions. Unlike conventional projects, in which technical variables introduce rigidity, the logic of construction management operates with more adaptable entities, whose complexity comes from the multiplicity of interactions and the continuous dynamics of the site over time. The attribution of properties such as function, user group, costs involved or dimensional aspects to each element allowed a rich and contextualized modeling, in which the interrelationships could be analyzed directly.

Finally, the application case, digital auctions, presented a highly dynamic scenario marked by the volatility of entities. In this context, data is not only generated in real time but can also quickly lose validity. The multiplicity of elements involved - products, bidders, operating rules, phases, bids and contractual ties - required a representation capable of dealing with complex relationships and frequent changes in the configuration of the system. In this case, the MLHG model was adapted to capture intricate interactions, such as the record that "bidder A offered X units of product P in phase 2 at price Y", simultaneously encapsulated entities and properties in hyperedges. The multifaceted representation also made it possible to segment the processes into pre-auction, active auction and post-auction, ensuring semantic coherence and analytical support. These results are in line with the recent literature in network science applied to the digital economy, which highlights the need for models capable of dealing with fast transactions, multiple agents, and contextual interdependencies (Abdelkafi et al. 2019a).

When comparing the three studies of the application cases, we found some convergences and divergences that reinforce the general validity of the model. In both rowing and civil construction, the

central entities (athletes and boats; pillars and beams) are relatively stable, although on different temporal scales. In digital auctions, instability and mutability are dominant, requiring greater flexibility. MLHG proved to be able to deal with both extremes: the structural stability of construction and the transactional volatility of auctions, as well as the hybrid balance of rowing, in which human and environmental variables coexist. This versatility suggests that the model has the potential to become a cross-cutting reference in different areas, from engineering to applied social sciences. Another common aspect that we can discuss is the importance of temporality. In rowing, time is associated with performance in training and competitions; in civil construction, it is related to the execution and planning phases; In online auctions, it defines crucial deadlines for participation and validity of proposals. In all cases, the ability to assign temporal properties to vertices and edges demonstrated that an MLHG model is not just a static model, but a structure capable of dealing with evolutionary processes. This brings it closer to research in dynamic networks over time, a booming field in data science (Holme and Saramäki 2012) . On the other hand, some practical limitations were also highlighted. In rowing, complexity comes mainly from the integration between qualitative and quantitative variables (such as team synchronization, individual power, and environmental conditions). In construction, the challenge is visual scalability: as the building grows, the representation can become overly dense. In online auctions, the main obstacle is the computational load associated with real-time updating, especially in scenarios of high competitiveness and high bidding volume. These limitations, however, do not nullify the validity of the proposal, but rather point to clear directions of evolution, such as the adoption of visual aggregation algorithms and integration with parallel or distributed processing technologies.

In epistemological language, the debate over these three studies confirms that the strength of the multidimensional network lies in the combination of structural topology and semantic richness. The model not only represents entities and relationships but also allows the incorporation of contextual attributes (temporal, quantitative, qualitative) that confer greater analytical expressiveness. By providing a coherent framework for systems as diverse as sports organizations, construction project management, and digital auction environments, the multidimensional network represents itself as a versatile and innovative tool, with relevant contributions to the advancement of information science and the modeling of complex systems.

6.4 Limitations and challenges identified

Although the MLHG model has shown potential in the three case studies carried out, it is important to recognize that every methodological proposal faces some limitations and practical obstacles, which influence its large-scale adoption. The critical identification of these aspects does not diminish the validity of the scientific contribution but is an essential step to guide future research and applications. This section discusses the main constraints encountered, organized into four major areas: computational complexity, scalability and visualization, integration with existing systems, and validation. One of the main challenges is related to the complex manipulation of MLHG. Although modeling offers semantic expressiveness and the ability to represent higher-order relationships, this richness brings an increase in computational costs, both in storage and in consultation. In scenarios such as digital auctions, where data is highly dynamic and subject to real-time updating, performing complex operations on hypergraphs can entail significant computational overhead in large-scale situations, compromising performance. This limitation has already been pointed out in literature specialized in complex network analysis, where it was observed that the transition from simple graphs to hypergraphs implies a leap in complexity that requires new algorithmic approaches. Additionally, the issue of scalability and visualization appears. While MLHG has been shown to be effective in hierarchically representing structures such as a reinforced concrete building, the density of vertices and hyperedges grows rapidly as the size of the system increases. In real civil engineering contexts, which involve hundreds or thousands of structural elements, the graphical representation can become dense and difficult to interpret. The problem of readability is not only aesthetic, but also cognitive, insofar as interpretation by engineers or project managers depends on the clarity of the visualization. In the literature we can also find this limitation, emphasizing that representations of very dense networks tend to lose informative capacity, requiring simplification or aggregation techniques.

Another set of limitations is related to the integration of MLHG with already established systems and tools. In the construction sector, for example, *Building Information Modelling* (BIM) has become the international standard for the representation and management of construction information. Therefore, the adoption of MLHG will require interoperability mechanisms with these systems, ensuring that the proposed semantic richness does not imply loss of compatibility with platforms in current use. Similarly, in the economic field, integration with already implemented electronic auction systems requires the development of interfaces and data flows capable of translating hypergraph structures into formats used

in practice (JSON[24], XML[25], SQL). Without this bridge, the model runs the risk of remaining in the experimental domain, with restricted applicability.

From an epistemological point of view, another limitation is the lack of standardized metrics to evaluate MLHG. While traditional graphs have consolidated measures such as centrality and modularity, hypergraphs lack widely recognized and normalized indicators, which makes it difficult to make objective comparisons between different implementations and to construct shared parameters. Recent works (Abdelkafi et al. 2019b) have developed some specific metrics, but the field still requires maturation. Regarding usability, one of the main challenges is related to the learning curve associated with the practical use of the model. Researchers and practitioners accustomed to relational models or traditional graph structures may experience some difficulty in understanding the functioning of nodes, edges, and hierarchies distributed at various levels, which may limit the initial adoption of MLHG. For this reason, the existence of intuitive modeling tools, interactive graphical interfaces, and clear documentation is essential to lower entry barriers and facilitate the application of the model in different contexts. The creation of applied examples and user guides will also contribute to a greater familiarity with the methodology, promoting its gradual integration into academic and professional environments.

Finally, it is important to discuss the issue of real-time updating. While in civil engineering and shipbuilding, for example, data can be updated at relatively stable intervals, in online auctions or in other economic systems of the digital age the need for continuous updating presents challenges for synchronization between successive states of the network. This limitation suggests the need to develop hybrid strategies, in which the MLHG structure is complemented by streaming mechanisms and incremental data processing, in line with recent research in big data and temporal networks (Abdelkafi et al. 2019b). In short, the limitations identified point to two paths of evolution: (i) technical improvement, centered on the optimization of algorithms, scalability, visualization, and integration with consolidated systems; and (ii) epistemological improvement, which involves the construction of metrics, validation methodologies and pedagogical dissemination of the approach. Far from compromising the relevance of MLHG, these challenges reinforce the need for continuity of research, ensuring that the model is consolidated as a generalizable proposal with an impact on multiple domains of science and professional practice.

[24] JavaScript Object Notation

[25] Extensible Markup Language

6.5 Scientific and technological contributions

The contributions were reflected in the four scientific articles that integrate and support the present doctoral work, in which the theoretical and experimental stages of the development of the MLHG models are documented. Each article addresses a specific stage of the research process, progressively refining the proposed model from its theoretical foundations to its empirical validation in applied domains (J. Ribas and Belo 2025c, 2025b, 2025a).

Article 1, "Multilevel Hypergraphs – An Alternative Approach to Modeling Complex Systems", presents the initial formulation of the model, exploring its ability to represent higher-order relations and hierarchical structures in complex engineering systems. Building upon this initial theoretical formulation, the second article focuses on the application of the model to a concrete and technically demanding domain.

Article 2, "Multilevel Hypergraphs in Construction Project Management: A Proposal for a Hierarchical Model", applies these concepts to the civil construction domain, proposing a formal framework to deal with technical, temporal and logistical interdependencies in structural projects. Following the validation of the model within the construction management context, the third article extends the discussion toward computational aspects related to data management and system dynamics.

Article 3, "Dynamic Data Management in Complex Systems: The Integration of Multilevel Hypergraphs (MLHG) and their CRUD Operations", extends the discussion to the field of Computer Science, proposing the integration between MLHG and dynamic data management mechanisms, in order to optimize real-time updating and inference processes.

Finally, the fourth article provides an empirical evaluation of the proposed approach in a highly dynamic and data-intensive environment. Article 4, "Modelling Auction House Processes with Multilevel Hypergraphs", empirically demonstrates the application of the model in the context of digital auctions, evidencing gains in efficiency, semantic consistency and ability to adapt to highly volatile data flows.

The convergence of these works confirms the relevance and coherence of the theoretical proposal, articulating knowledge from Engineering, Civil Construction and Computer Science. The thesis, by integrating these strands, establishes a bridge between theory and practice, offering an interdisciplinary approach capable of responding to contemporary problems of high complexity. This integration shows that the contribution of the MLHG model transcends mere mathematical formalization, positioning itself as a conceptual and operational tool with transformative potential in several scientific and applied domains.

We think that we should highlight the identification of the limitations and challenges of this work, without intending to diminish the value of the research, since they pave the way for new fronts of study and improvement. Among the critical aspects pointed out are the need for a more in-depth mathematical formalization, the creation of mechanisms that allow its scalability in environments with a large volume of data, and the integration with explainable artificial intelligence platforms. The transparent recognition of these limitations reinforces the scientific character of the investigation, since it admits the current stage of the proposal without omitting its inherent weaknesses at this initial moment (Finzel 2025).

From a technological point of view, the new hierarchical model showed potential as the basis for the next generation of SBDs. This paradigm can sustain engines capable of dealing with highly complex entities and relationships, maintaining semantic coherence and structural integrity even under circumstances of dynamic change. The practical implications discussed in the previous chapter reinforce that the application of this new model goes beyond academia, extending to industrial and social sectors in which complexity is a decisive factor. The paradigm can be exploited to optimize construction processes, monitor global logistics chains, structure digital market ecosystems or even support the development of public policies based on interdependent systems. Thus, this thesis is not limited to a theoretical contribution but opens concrete perspectives for applied innovation in multiple domains.

On the other hand, the research carried out allowed us to draw a broader epistemological lesson: data science and the modeling of complex systems require structures capable of integrating mathematical formalism, semantic expressiveness, and computational flexibility. Models that are too rigid lose adaptability, while overly flexible models sacrifice formal precision. The new paradigm, by combining the multilevel hierarchy with the relational richness of hypergraphs, approaches a balance between these two poles, constituting a methodologically technically promising proposal.

This PhD work brought some scientific and technological contributions to the field of complex data modeling and to the development of computational systems with the capacity to deal with contexts of high interdependence. The main contribution lies in the formalization and application of the MLHG model, whose design goes beyond traditional approaches based on graphs or classical relational models. From a scientific point of view, MLHG constitutes a conceptual advance by introducing a structure capable of articulating higher-order relations, multiple hierarchies, and contextual properties in a single space of representation. Although property graph models allow enriching vertices and edges with attributes, they remain constrained by the binary logic of the relation. MLHG, on the other hand, expands this possibility by integrating hyperedges through virtual vertices, giving the model a higher expressiveness, allowing it

to represent simultaneously horizontal (between entities of the same level) and vertical (between distinct levels of abstraction). This characteristic brings graph modeling closer to contemporary theories of complex systems, offering a formal approach closer to the reality of domains in which structural stability coexists with high dynamicity.

In the methodological domain, this research contributed through the definition of a replicable procedure for the construction of MLHG, including the identification of abstraction levels, the introduction of temporal properties and the articulation between static and dynamic entities. The application of the procedure in different areas has shown that this is not just a conceptual model, but a method capable of guiding the construction of databases and information systems in real contexts. This methodological contribution is particularly relevant in an era when data science requires not only theoretically sound models, but also practical implementation tools. The contributions of the work conducted reinforce this methodological dimension.

From a technological perspective, MLHG extends the functionality of graph-oriented databases. Systems such as Neo4j still rely on indirect solutions to represent multiple hierarchical levels, such as using multiple labels or excessively fragmenting entities. By proposing native mechanisms of hierarchy, temporality, and contextual properties, MLHG pave the way for the development of extensions or new engines capable of dealing directly and efficiently with such demands. In addition, studies have shown that MLHG can be integrated into decision-making support systems, whether in civil engineering, to associate constructive progress with accumulated costs, or in digital markets, to predictively simulate bidder behaviors.

Finally, it is important to highlight the interdisciplinary contribution to this work. The possibility of applying MLHG to such diverse domains proves its suitability as a unifying conceptual framework, capable of promoting interoperability between heterogeneous systems and fostering communication between traditionally distant areas. This cross-cutting character directly addresses the emerging needs of contemporary data science, which favors holistic and interdisciplinary approaches. In summary, the scientific and technological contributions of this research can be summarized in four fundamental points:

- the theoretical formalization of MLHG as a robust model of representation;
- the development of a methodological procedure that can be replicated;
- empirical validation in different contexts, confirming the versatility of the model;

- the technological expansion of BDOGs capabilities, making room for new computational implementations.

These contributions show that MLHG is not only a conceptual innovation, but also a scientifically rigorous, technologically feasible and interdisciplinary applicable proposal, capable of responding to contemporary challenges of the representation of complex systems.

6.6 Practical Implications and Applicability

By proposing and validating the application of the MLHG model in different contexts, this work is not restricted to an exercise of theoretical conceptualization but presents concrete contributions that translate into relevant practical implications. The application of MLHG manifests itself in several domains in which complex, dynamic and hierarchically structured information management is fundamental for efficiency, transparency and decision-making. One of the main practical implications is in civil engineering and construction. By introducing MLHG in construction modeling, it is possible to develop information systems that not only represent the geometry and physical elements of the building, but also incorporate execution schedules, building dependencies and financial estimates. In practice, this integration translates into efficiency gains in planning, in the management of the work, reducing inconsistencies between architectural, structural and budgetary projects. In addition, the ability to model multilevel relationships between components, for example, between a footing, the column that rests on it, and the beam that that column supports, provides greater clarity in managing structural risks and identifying potential failures. This application is particularly relevant in large-scale or highly technically complex projects, where any coordination errors can generate exponential costs.

In the world of high-level sport, the application of MLHG lies in its ability to represent complex collective performance systems, in which multiple variables interact at different scales. In rowing, studied in this work, the model allowed to model the relationship between athletes, boats, physiological metrics and environmental conditions in detail. This structure facilitates the development of training support systems capable of analyzing not only individual performance, but also collective synergy, an essential factor in sports where coordination is important for success. From a practical point of view, this applicability can be extended to real-time monitoring systems, integrating biometric and environmental sensors and allowing immediate adjustments in competition strategies.

In the field of digital commerce and online markets, the application of MLHG is especially relevant in view of the growing digitalization and dynamism of economic interactions. Traditional auction management systems tend to organize data in a linear and rigid fashion, making it difficult to analyze emerging relationships between participants, products, auction phases, and subsequent contracts. An MLHG, by enabling the multifaceted representation of these entities, offers digital platforms a framework that supports predictive queries, such as identifying patterns of participant behavior at different stages or simulating award scenarios based on contractual rules. From a practical point of view, this capability translates into competitive advantages for auction houses and digital marketplaces, by enabling faster, more transparent and informed decisions, increasing the confidence of participants and the efficiency of transactions.

The opportunity to make data-driven decisions is one of the main implications of the application domains of the hierarchical multilevel hypergraph model. Hierarchically structuring data and the ability to integrate temporal and contextual properties provides a foundation for predictive analytics and simulations. This characteristic is particularly useful in organizational environments with high uncertainty since the ability to explore diverse scenarios in a coherent and consistent semantic and structural manner generates an obvious practical advantage.

Another relevant implication is the interoperability between heterogeneous systems. Today, many organizations rely on multiple information systems, which are often disconnected or have incompatible data structures. The model, through a hierarchical, contextual and flexible representational approach, can act as an intermediate layer of semantic integration, allowing communication and articulation between different systems. This characteristic expands the model's applicability in corporate, governmental, and scientific scenarios, in which the integration of fragmented data is a constant challenge (Rajaraman, Corp, and Ordille 2006).

In summary, the practical implications and the applicability of the model developed in this work are in three complementary levels: sectoral, by validating the MLHG in specific areas such as construction, sports and digital auctions; organizational, for its usefulness in integrating information, supporting decision-making and optimizing processes; and by opening new possibilities for the development of database mechanisms and intelligent systems. These three plans confirm the innovative, versatile and pragmatic nature of MLHG, reinforcing its importance not only as a scientific contribution, but also as a tool for technological transformation with direct consequences in strategic sectors of the economy and society.

6.7 Future Research

The work developed in this thesis represents an initial milestone in the formalization and validation of MLHG as an alternative paradigm for the representation and manipulation of complex information. However, as with any emerging scientific proposal, the consolidation and expansion of this approach require a continuous, critical, and systematic research program. In this section we will outline some of the main future research perspectives, distributed in different axes of development, ranging from theoretical foundation to practical application in interdisciplinary domains.

The first line of deepening lies in the theoretical-formal refinement of MLHG. Although this thesis has established the structural and semantic principles of the model, it is necessary to consolidate a more formalized mathematical base, which allows the definition of axioms, theorems and properties of completeness or consistency associated with MLHG. This more rigorous formalization will pave the way for systematic comparison with other mathematical structures, such as classical hypergraphs, formal ontologies and complex network models, allowing to position MLHG clearly in the ecosystem of representation theories (Bretto 2013a, Handbook on Ontologies 2009, Newman 2003).

Another line of research concerns the integration of MLHG into database management systems and large-scale distributed environments. The experience with property graphs and hypergraph-oriented databases demonstrates the growing demand for solutions that go beyond the limits of classical relational structures. Thus, a promising line of research would be the design of a natively hypergraph-oriented multilevel DBMS, capable of executing complex queries in real time and supporting ACID or BASE transactions. The development of specific optimization algorithms for multilevel operations (such as aggregations, property inheritances and dynamic partitions) is also a promising field of study (Robinson, Webber, and Eifrem 2015a, Dan Pritchett 2008). In the field of Artificial Intelligence and Data Science, MLHG could play an important role as a support framework for automatic inference systems and explainable machine learning. The model's ability to represent complex relationships and incorporate contextual attributes into relationships suggests a natural integration with *neural network algorithms* (GNN). Future studies may explore how the use of MLHG as an input framework can enhance the performance of predictive algorithms, especially in complex domains, such as biomedical networks, global resource chains, or urban transport systems.

The fourth line of research focuses on the application of MLHG in interdisciplinary contexts and with direct social impact. For example, in digital health, multilevel mapping could help integrate information from diverse sources – electronic health records, continuous monitoring sensors, genomic

data, and lifestyles – into a coherent and navigable framework, enabling advances in personalized medicine. Similarly, in digital governance and public policy, MLHG can be exploited as a tool to map complex ecosystems of actors, institutions, and information flows, supporting evidence-based policymaking. An exploratory and interactive approach to visualizing multi-level data should also be considered. One of the biggest challenges of contemporary Data Science is not only to store and process large volumes of data, but to represent it in an understandable and meaningful way for end users. MLHG, due to its hierarchical nature, opens possibilities to design visual interfaces that allow switching between distinct levels of detail, promoting a clearer understanding of emerging patterns. Future studies may research scalable visualization methodologies that associate principles of graph design, information visualization, and augmented reality, expanding the analytical and pedagogical potential of the model.

Another perspective for future evolution could involve the systematic assessment of the scalability and robustness of MLHG in big data environments. Although the studies of this thesis have demonstrated its application in concrete scenarios, it is essential to evaluate the performance of the model in contexts of millions of vertices and hyperedges, characteristic of large-scale real-world applications. Issues such as parallelization of operations, efficiency in performing distributed queries and fault tolerance mechanisms are central topics to be explored in future work, in line with advances in high-performance computing (Dean and Ghemawat, 2004). In the future, several research possibilities promise to expand the current boundaries, in a comprehensive and multifaceted way. This includes deeper mathematical formalization, continuous technological advancement, synergetic integration with artificial intelligence, applied exploration in domains of societal relevance, visual interface design, large-scale scalability assessment, and transdisciplinary scholarly diffusion.

6.8 Final considerations

This PhD work sought to address one relevant contemporary challenge of information science: the need for formal models capable of representing, manipulating, and analyzing systems of high structural, semantic and temporal complexity, in a comprehensive way. MLHG is a response to this challenge, introducing an innovative paradigm that combines the expressiveness of hypergraphs with the hierarchical capacity of segmentation at various levels. From a scientific point of view, this research contributed to consolidating the notion that complex reality cannot be adequately captured by linear and rigid structures, such as those provided by classical relational models. The formalization of MLHG has shown that it is

possible to represent higher-order relations and contextual properties intrinsically, without the need for artificial decomposition or joining mechanisms. These characteristic positions the model as a differentiating proposal in relation to other contemporary approaches, such as property graphs or complex networks. (Baxendale and Codd 1970, Robinson, Webber, and Eifrem 2015a, Newman 2003)

Methodologically, this research demonstrated that the adoption of a practical approach, based on real cases, is essential to validate the applicability and robustness of models intended for the representation of complex systems. For this purpose, three domains of different nature were selected — team sport (rowing), reinforced concrete structures in civil engineering and digital auctions in economic-computational environments — which served as conceptual laboratories to test the behavior and generality of the proposed models. Each of these contexts revealed dimensions of systemic complexity: rhythmic and interdependent coordination in rowing, normative and technical rigidity in civil construction, and strategic volatility in digital markets. This diversity was decisive in assessing the limits and potentialities of the model, evidencing its ability to adapt to different realities.

In terms of prospects, this work does not stand as an endpoint but rather paves the way for an ongoing research program. Several avenues for future investigation were identified, including the development of more robust mathematical foundations, the integration of the model with GNN-based algorithms, the design of multilevel hypergraph-oriented database engines, the creation of interactive visualization interfaces, and the evaluation of scalability in large-scale environments. These research directions constitute the natural progression of this thesis and simultaneously form a research agenda open to interdisciplinary collaboration. This work achieved its central objective: demonstrating the validity, relevance, and applicability of MLHG as a formal model for the representation of complex systems. At the same time, it opened perspectives for theoretical and technological advances with potential impact in both academic and industrial domains, as well as within broader social practice. The trajectory outlined here does not mark a definitive endpoint but rather a point of departure for a new stage of research and innovation, in which MLHG may increasingly establish itself as a reference framework within data science, formal modelling, and complex systems engineering.

References

Abdelkafi, Nizar, Christina Raasch, Angela Roth, and R. Srinivasan. 2019a. "Multi-Sided Platforms." *Electronic Markets* 29(4): 553–59. doi:10.1007/s12525-019-00385-4.

Abdelkafi, Nizar, Christina Raasch, Angela Roth, and R. Srinivasan. 2019b. "Multi-Sided Platforms." *Electronic Markets* 29(4): 553–59. doi:10.1007/s12525-019-00385-4.

Abiteboul, Serge, Dallan Quass, Jason Mchugh, Jennifer Widom, and Janet L. Wiener. 1997. "The Lorel Query Language for Semistructured Data." *International Journal on Digital Libraries* 1(1). doi:10.1007/s007990050005.

Allen, Frances E. 1970. "Control Flow Analysis." *ACM SIGPLAN Notices* 5(7): 1–19. doi:10.1145/390013.808479.

Angles, Renzo. 2008. *The Property Graph Database Model*.

Angles, Renzo, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2018. "Foundations of Modern Query Languages for Graph Databases." *ACM Computing Surveys* 50(5). doi:10.1145/3104031.

Angles, Renzo, and Claudio Gutierrez. 2008a. "Survey of Graph Database Models." *ACM Computing Surveys* 40(1). doi:10.1145/1322432.1322433.

"Applications of Network Optimization R. Ahuja, T. L. Magnanti, J. B. Orlin and M. R. Reddy OR 300-94 June 1994." 1994. *Engineering* (June).

Ausiello, G., A. D'Atri, and D. Sacca. 1986. "MINIMAL REPRESENTATION OF DIRECTED HYPERGRAPHS." *SIAM Journal on Computing* 15(2): 418–31. doi:10.1137/0215029.

Ausiello, Giorgio, and Luigi Laura. 2017. "Directed Hypergraphs: Introduction and Fundamental Algorithms—A Survey." *Theoretical Computer Science* 658. doi:10.1016/j.tcs.2016.03.016.

Battiston, Federico, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean Gabriel Young, and Giovanni Petri. 2020. "Networks beyond Pairwise Interactions: Structure and Dynamics." *Physics Reports* 874: 1–92. doi:10.1016/j.physrep.2020.05.004.

Baxendale, P, and E F Codd. 1970. *Information Retrieval A Relational Model of Data Large Shared Data Banks*.

Berge, C. 1989. *Hypergraphs: Combinatorics of Finite Sets*. ed. Elsevier.

Bonifati, Angela, – 869x, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. *Schema Validation and Evolution for Graph Databases*. https://www.gqlstandards.org/.

Bretto, Alain. 2013a. *Hypergraph Theory*. http://www.springer.com/series/8445.

Bretto, Alain. 2013b. "Hypergraph Theory: An Introduction." *Mathematical Engineering* 11. doi:10.1007/978-3-319-00080-0.

Bulgaru, Elena, and Vitaly I. Voloshin. 1997. "Mixed Interval Hypergraphs." *Discrete Applied Mathematics* 77(1): 29–41. doi:10.1016/S0166-218X(97)89209-8.

Buneman, O. P., S. B. Davidson, and A. Watters. 1991. "A Semantics for Complex Objects and Approximate Answers." *Journal of Computer and System Sciences* 43(1). doi:10.1016/0022-0000(91)90035-4.

Cai, Derun, Moxian Song, Chenxi Sun, Baofeng Zhang, Shenda Hong, and Hongyan Li. 2022. *Hypergraph Structure Learning for Hypergraph Neural Networks*.

Cambini, Riccardo, Giorgio Gallo, and Maria Grazia Scutellà. 1997. "Flows on Hypergraphs." *Mathematical Programming, Series B* 78(2): 195–217. doi:10.1007/BF02614371.

Çatalyürek, Ümit, Karen Devine, Marcelo Faraj, Lars Gottesbüren, Tobias Heuer, Henning Meyerhenke, Peter Sanders, et al. 2023. "More Recent Advances in (Hyper)Graph Partitioning." *ACM Computing Surveys* 55(12): 1–38. doi:10.1145/3571808.

Cattell, Rick. 2010. *Scalable SQL and NoSQL Data Stores*.

Chathurika, P T C, M Vivitharani, L H V Chamikara, N T B Dias, and M A Siriwardhene. 2023. "Research Article." 4(2): 11–19.

Chehreghani, Morteza Haghir, Hassan Abolhassani, and Mostafa Haghir Chehreghani. 2008. "Improving Density-Based Methods for Hierarchical Clustering of Web Pages." *Data and Knowledge Engineering* 67(1): 30–50. doi:10.1016/j.datak.2008.06.006.

Chen, Chien Chang, Cheng Shian Lin, Yen Ting Chen, Wen Her Chen, Chien Hua Chen, and I. Cheng Chen. 2023. "Intelligent Performance Evaluation in Rowing Sport Using a Graph-Matching Network." *Journal of Imaging* 9(9). doi:10.3390/jimaging9090181.

Chen, Peter Pin Shan. 1976. "The Entity-Relationship Model—toward a Unified View of Data." *ACM Transactions on Database Systems (TODS)* 1(1). doi:10.1145/320434.320440.

Codd, E. F. 1970a. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 13(6). doi:10.1145/362384.362685.

Codd, E. F. 1970b. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 13(6). doi:10.1145/362384.362685.

Codd, E. F. 1981. “Data Models in Database Management.” *ACM SIGMOD Record* 11(2): 112–14. doi:10.1145/960126.806891.

Cole, Sam, and Yizhe Zhu. 2020. “Exact Recovery in the Hypergraph Stochastic Block Model: A Spectral Algorithm.” *Linear Algebra and Its Applications* 593: 45–73. doi:10.1016/j.laa.2020.01.039.

Dai, Qionghai, and Yue Gao. 2023. *Hypergraph Computation Paradigms*. doi:10.1007/978-981-99-0185-2_3.

DAN PRITCHETT, EBAY. 2008. *ACM QUEUE*. www.acmqueue.com.

Dean, Jeffrey, and Sanjay Ghemawat. 2004. *MapReduce: Simplified Data Processing on Large Clusters*.

Demir, Engin, and Cevdet Aykanat. 2010. “Efficient Successor Retrieval Operations for Aggregate Query Processing on Clustered Road Networks.” *Information Sciences* 180(14): 2743–62. doi:10.1016/j.ins.2010.03.015.

Deng, Shanglyu, and Qiyao Zhou. 2025. “Recurring Auctions with Costly Entry: Theory and Evidence.” http://arxiv.org/abs/2306.17355.

Diestel, Reinhard. 2018. Graph Theory (on Demand Printing Of 02787) *Graph Theory (on Demand Printing Of 02787)*. doi:10.1201/9780429493768.

De Domenico, Manlio, Albert Solé-Ribalta, Emanuele Cozzo, Mikko Kivelä, Yamir Moreno, Mason A. Porter, Sergio Gómez, and Alex Arenas. 2013. “Mathematical Formulation of Multilayer Networks.” *Physical Review X* 3(4). doi:10.1103/PhysRevX.3.041022.

Eppstein, David, and Vijay V. Vazirani. 2019. “NC Algorithms for Computing a Perfect Matching, the Number of Perfect Matchings, and a Maximum Flow in One-Crossing-Minor-Free Graphs.” *Annual ACM Symposium on Parallelism in Algorithms and Architectures*: 23–30. doi:10.1145/3323165.3323206.

Fan, Wenfei, Yinghui Wu, and Jingbo Xu. 2016. “Functional Dependencies for Graphs.” In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, 1843–57. doi:10.1145/2882903.2915232.

Faria, Luerbio, André L.P. Guedes, and Lilian Markenzon. 2021. “On Feedback Vertex Set in Reducible Flow Hypergraphs.” *Procedia Computer Science* 195: 212–20. doi:10.1016/j.procs.2021.11.027.

Feng, Song, Emily Heath, Brett Jefferson, Cliff Joslyn, Henry Kvinge, Hugh D. Mitchell, Brenda Praggastis, et al. 2020. "Hypergraph Models of Biological Networks to Identify Genes Critical to Pathogenic Viral Response." http://arxiv.org/abs/2010.03068.

Finzel, Bettina. 2025. "Current Methods in Explainable Artificial Intelligence and Future Prospects for Integrative Physiology." *Pflugers Archiv European Journal of Physiology* 477(4): 513–29. doi:10.1007/s00424-025-03067-7.

Francis, Nadime, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, et al. 2018. "Cypher: An Evolving Query Language for Property Graphs." In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, 1433–45. doi:10.1145/3183713.3190657.

Gallo, Giorgio, Giustino Longo, Stefano Pallottino, and Sang Nguyen. 1993. "Directed Hypergraphs and Applications." *Discrete Applied Mathematics* 42(2–3): 177–201. doi:10.1016/0166-218X(93)90045-P.

Gao, Xiang, Fan Zhou, Kedi Xu, Xiang Tian, and Yaowu Chen. 2022. "A Parallel Algorithm for Maximal Cliques Enumeration to Improve Hypergraph Construction." *Journal of Computational Science* 65(June): 101905. doi:10.1016/j.jocs.2022.101905.

Geng, Yuxia, Jiaoyan Chen, Zhiquan Ye, Zonggang Yuan, Wei Zhang, and Huajun Chen. 2021. "Explainable Zero-Shot Learning via Attentive Graph Convolutional Network and Knowledge Graphs." *Semantic Web* 12(5): 741–65. doi:10.3233/SW-210435.

Ghrab, Amine, Oscar Romero, Sabri Skhiri, Alejandro Vaisman, and Esteban Zimányi. 2016. "GRAD: On Graph Database Modeling." http://arxiv.org/abs/1602.00503.

Graham M. Winch. 2021. *Managing Construction Projects An Information Processing Approach*.

Gregor, Shirley, and David Jones. 2007. "The Anatomy of a Design Theory." *Journal of the Association for Information Systems* 8(5): 312–35. doi:10.17705/1jais.00129.

Guedes, A. L.P., L. Markenzon, and L. Faria. 2011. "Flow Hypergraph Reducibility." *Discrete Applied Mathematics* 159(16): 1775–85. doi:10.1016/j.dam.2011.02.006.

Gueting, R. 1994. "GraphDB: Modeling and Querying Graphs in Databases." *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES* 1(20).

Haerder, Theo, and Andreas Reuter. 1983. "Principles of Transaction-Oriented Database Recovery." *ACM Computing Surveys (CSUR)* 15(4): 287–317. doi:10.1145/289.291.

Handbook on Ontologies. 2009. Handbook on Ontologies Springer Berlin Heidelberg. doi:10.1007/978-3-540-92673-3.

Healy, Patrick, and Nikola S Nikolov. *Hierarchical Drawing Algorithms*.

Hendricks, Kenneth, Alan Sorensen, and Thomas Wiseman. 2024. *Dynamics and Efficiency in Decentralized Online Auction Markets*.

Heo, Yu-Jung, Eun-Sol Kim, Woo Suk Choi, and Byoung-Tak Zhang. 2022. 1 *Hypergraph Transformer: Weakly-Supervised Multi-Hop Reasoning for Knowledge-Based Visual Question Answering*. Long Papers. https://github.com/yujungheo/.

Hevner, Alan R, Salvatore T March, Jinsoo Park, and Sudha Ram. 2004. 28 Design Science in IS Research MIS Quarterly *DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH 1*.

Holme, Petter, and Jari Saramäki. 2012. "Temporal Networks." *Physics Reports* 519(3): 97–125. doi:10.1016/j.physrep.2012.03.001.

Huseynli, Alisettar, and M. Ali Akcayol. 2021. "Bloom Filter Based Graph Database CRUD Optimization for Stream Data." *Proceedings of the 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021* 2: 1056–61. doi:10.1109/IDAACS53288.2021.9661054.

Johnson, Michael, and Robert Rosebrugh. 2014. *Spans of Lenses*.

Karypis, George, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1999. "Multilevel Hypergraph Partitioning: Applications in VLSI Domain." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7(1): 69–79. doi:10.1109/92.748202.

Karypis, George, and Vipin Kumar. 2005. "Parallel Multilevel Series K-Way Partitioning Schema for Irregular Graphs." *SIAM Review* 41(2): 278–300. doi:10.1137/s0036144598334138.

Keim, Daniel A, Florian Mansmann, Jörn Schneidewind, Jim Thomas, and Hartmut Ziegler. *Visual Analytics: Scope and Challenges*. http://nvac.pnl.gov.

Kim, Won. 1990. "Object-Oriented Databases: Definition and Research Directions." *IEEE Transactions on Knowledge and Data Engineering* 2(3). doi:10.1109/69.60796.

Kivelä, Mikko, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. 2014. "Multilayer Networks." *Journal of Complex Networks* 2(3): 203–71. doi:10.1093/comnet/cnu016.

Klamt, Steffen, Utz Uwe Haus, and Fabian Theis. 2009. "Hypergraphs and Cellular Networks." *PLoS Computational Biology* 5(5). doi:10.1371/journal.pcbi.1000385.

Landry, Nicholas W., and Juan G. Restrepo. 2022. “Hypergraph Assortativity: A Dynamical Systems Perspective.” *Chaos* 32(5). doi:10.1063/5.0086905.

de Lara, Juan, Hans Vangheluwe, and Manuel Alfonseca. 2004. “Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM3.” *Software & Systems Modeling* 3(3): 194–209. doi:10.1007/s10270-003-0047-5.

LaSalle, Dominique, and George Karypis. 2015. “Technical Report. A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning.” *University of Minnesota*.

Lee, John Boaz, Ryan A. Rossi, Sungchul Kim, Nesreen K. Ahmed, and Eunyee Koh. 2019. “Attention Models in Graphs: A Survey.” *ACM Transactions on Knowledge Discovery from Data* 13(6). doi:10.1145/3363574.

Li, Huan, Ruisheng Zhang, Zhili Zhao, and Xin Liu. 2021. “Lpa-Mni: An Improved Label Propagation Algorithm Based on Modularity and Node Importance for Community Detection.” *Entropy* 23(5). doi:10.3390/e23050497.

Liu, Meng, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. 2021. “Strongly Local Hypergraph Diffusions for Clustering and Semi-Supervised Learning.” In *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, Association for Computing Machinery, Inc, 2092–2103. doi:10.1145/3442381.3449887.

Lotito, Quintino Francesco, Federico Musciotto, Federico Battiston, and Alberto Montresor. 2024. “Exact and Sampling Methods for Mining Higher-Order Motifs in Large Hypergraphs.” *Computing* 106(2): 475–94. doi:10.1007/s00607-023-01230-5.

Luiz, André, Pires Guedes, and Lilian Markenzon. 2004. *DIRECTED HYPERGRAPH PLANARITY*.

Mao, Mingsong, Jie Lu, Jialin Han, and Guangquan Zhang. 2019. “Multiobjective E-Commerce Recommendations Based on Hypergraph Ranking.” *Information Sciences* 471: 269–87. doi:10.1016/j.ins.2018.07.029.

March, J.G. and Simon, H.A. 1958. *Organizations*. New York, NY: Wiley.

March, Salvatore T, and Gerald F Smith. 1995. 15 Decision Support Systems *Design and Natural Science Research on Information Technology*.

Marx, Arnim, and Markus König. 2013. *MODELING AND SIMULATING SPATIAL REQUIREMENTS OF CONSTRUCTION ACTIVITIES*.

Mathew, Bibin, Sunil Jacob John, and Harish Garg. 2020. “Vertex Rough Graphs.” *Complex and Intelligent Systems* 6(2): 347–53. doi:10.1007/s40747-020-00133-8.

Misiorek, Pawel, and Szymon Janowski. 2023. “Hypergraph-Based Importance Assessment for Binary Classification Data.” *Knowledge and Information Systems* 65(4): 1657–83. doi:10.1007/s10115-022-01786-2.

Moretti, C O, T N Bittencourt, J C André, and L F Martha. *Algoritmos Automáticos de Partição de Domínio*.

Navathe, Shamkant B. 1992. “Evolution of Data Modeling for Databases.” *Communications of the ACM* 35(9). doi:10.1145/130994.131001.

Newman, M E J. 2003. 45 Society for Industrial and Applied Mathematics *The Structure and Function of Complex Networks **. http://www.siam.org/journals/sirev/45-2/42480.html.

Odedairo, Babatunde Omoniyi. 2024. “Assessing the Influence of Various Work Breakdown Structures on Project Completion Time.” *Engineering, Technology and Applied Science Research* 14(2): 13773–79. doi:10.48084/etasr.7023.

Oguz Selvitopi, R., Ata Turk, and Cevdet Aykanat. 2012. “Replicated Partitioning for Undirected Hypergraphs.” *Journal of Parallel and Distributed Computing* 72(4): 547–63. doi:10.1016/j.jpdc.2012.01.004.

Ouvrard, Xavier. 2020. “Hypergraphs: An Introduction and Review.” http://arxiv.org/abs/2002.05014.

Paredaens, Jan, Peter Peelman, and Letizia Tanca. 1995. “G-Log: A Graph-Based Query Language.” *IEEE Transactions on Knowledge and Data Engineering* 7(3). doi:10.1109/69.390249.

Poulovassilis, Alexandra, and Mark Levene. 1994. “A Nested-Graph Model for the Representation and Manipulation of Complex Objects.” *ACM Transactions on Information Systems (TOIS)* 12(1). doi:10.1145/174608.174610.

Praggastis, Brenda, Sinan Aksoy, Dustin Arendt, Mark Bonicillo, Cliff Joslyn, Emilie Purvine, Madelyn Shapiro, and Ji Young Yun. 2024a. “HyperNetX: A Python Package for Modeling Complex Network Data as Hypergraphs.” *Journal of Open Source Software* 9(95): 6016. doi:10.21105/joss.06016.

Praggastis, Brenda, Sinan Aksoy, Dustin Arendt, Mark Bonicillo, Cliff Joslyn, Emilie Purvine, Madelyn Shapiro, and Ji Young Yun. 2024b. “HyperNetX: A Python Package for Modeling Complex Network Data as Hypergraphs.” *Journal of Open Source Software* 9(95): 6016. doi:10.21105/joss.06016.

Pritchett, Dan. 2008. “Base an Acid Alternative.” *Queue* 6(3): 48–55. doi:10.1145/1394127.1394128.

Rajaraman, Anand, Kosmix Corp, and Joann Ordille. 2006. *Data Integration: The Teenage Years*.

Ribas, J., and Belo O. 2025a. “An Alternative Approach for Managing Construction Engineering Projects Using Multilevel Hypergraphs.” In *Proceedings of the 3rd International Conference on Construction, Energy, Environment and Sustainability (CEES 2025)*, Lecture Notes in Civil Engineering, eds. Umberto Berardi, Julieta António, and Nuno Simões. Bari, Italy: Springer.

Ribas, J., and Belo O. 2025b. “Multilevel Hypergraph Databases Conceptual Modelling: Approaching CRUD Operations.” In *Model and Data Engineering (MEDI 2024)*, Lecture Notes in Computer Science, eds. Carlos Ordonez, Giancarlo Sperlì, Elio Masciari, and Ladjel Bellatreche. Cham: Springer, 175–82. doi:10.1007/978-3-031-87719-3_13.

Ribas, J., and Belo O. 2025c. “Multilevel Hypergraphs: A Conceptual Approach for Complex System Database Modelling.” In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) – Volume 1*, Porto, Portugal: SciTePress, 87–98. doi:10.5220/0013356900003929.

Ribas, J., and Belo O. 2026. “Modelling Auction House Processes with Multilevel Hypergraphs”. *In Proceedings of the 14th World Conference on Information Systems and Technologies (WorldCist’26).* Springer. Full paper, peer-reviewed, accepted.

Ribas, J. (2015). *Planeamento e gestão da construção sob a ótica da sustentabilidade* (Tese de doutoramento, Universidade de Aveiro). Repositório da Universidade de Aveiro. http://hdl.handle.net/10773/15142

Robinson, Ian, Jim Webber, and Emil Eifrem. 2015. *Graph Databases*.

Rodriguez, Marko A., and Peter Neubauer. 2010. “Constructions from Dots and Lines.” *Bulletin of the American Society for Information Science and Technology* 36(6): 35–41. doi:10.1002/bult.2010.1720360610.

Sandell, Johan, Einar Asplund, Workneh Yilma Ayele, and Martin Duneld. 2024. *Performance Comparison Analysis of ArangoDB, MySQL, and Neo4j: An Experimental Study of Querying Connected Data*. https://hdl.handle.net/10125/107319.

Schermann, Michael, Holmer Hemsen, Christoph Buchmüller, Till Bitter, Helmut Krcmar, Volker Markl, and Thomas Hoeren. 2014. “An Interdisciplinary Opportunity for Information Systems Research.” *Business and Information Systems Engineering* 6(5): 261–66. doi:10.1007/s12599-014-0345-1.

Selvakkumaran, Navaratnasothie, and George Karypis. 2005. XX IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN *Multi-Objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization*.

Sluzalec, Tomasz, Rafal Grzeszczuk, Sergio Rojas, Witold Dzwinel, and Maciej Paszynski. 2022. "Quasi-Optimal hp-Finite Element Refinements towards Singularities via Deep Neural Network Prediction." 142(April): 157–74. doi:10.1016/j.camwa.2023.04.023.

Song, Sicheng, Juntong Chen, Chenhui Li, and Changbo Wang. 2023. "GVQA: Learning to Answer Questions about Graphs with Visualizations via Knowledge Base." *Conference on Human Factors in Computing Systems - Proceedings*. doi:10.1145/3544548.3581067.

Srivillibhuthur, Manasa, Bailey N Warder, H Toke, Pooja P Shah, Qiang Feng, Nan Gao, Edward M Bonder, and Michael P Verzi. 2018. Developmental Biology *Author ' s Accepted Manuscript*. https://doi.org/10.1016/j.ydbio.2018.04.015.

Stanley, Richard P. 1973. "Acyclic Orientations of Graphs." *Discrete Mathematics* 5(2): 171–78. doi:10.1016/0012-365X(73)90108-8.

Stonebraker, Michael, Daniel Abadi, David J. Dewitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. 2010. "MapReduce and Parallel DBMSs: Friends or Foes?" *Communications of the ACM* 53(1): 64–71. doi:10.1145/1629175.1629197.

Tian, Ling, Xue Zhou, Yan Ping Wu, Wang Tao Zhou, Jin Hao Zhang, and Tian Shu Zhang. 2022. "Knowledge Graph and Knowledge Reasoning: A Systematic Review." *Journal of Electronic Science and Technology* 20(2): 100159. doi:10.1016/j.jnlest.2022.100159.

Vogels, Werner. 2009. "Eventually Consistent." *Communications of the ACM* 52(1): 40–44. doi:10.1145/1435417.1435432.

Wang, Zhihui, Jianrui Chen, Fernando E. Rosas, and Tingting Zhu. 2022. "A Hypergraph-Based Framework for Personalized Recommendations via User Preference and Dynamics Clustering." *Expert Systems with Applications* 204(December 2021): 117552. doi:10.1016/j.eswa.2022.117552.

Wu, Longcan, Daling Wang, Kaisong Song, Shi Feng, Yifei Zhang, and Ge Yu. 2021. "Dual-View Hypergraph Neural Networks for Attributed Graph Learning." *Knowledge-Based Systems* 227. doi:10.1016/j.knosys.2021.107185.

Xie, Ming, Xiu Xiu Zhan, Chuang Liu, and Zi Ke Zhang. 2023. "An Efficient Adaptive Degree-Based Heuristic Algorithm for Influence Maximization in Hypergraphs." *Information Processing and Management* 60(2): 103161. doi:10.1016/j.ipm.2022.103161.

Yang, Zhihao, Yingxin Bi, Linbing Wang, Dongwei Cao, Rongxu Li, and Qianqian Li. 2022. "Development and Application of a Field Knowledge Graph and Search Engine for Pavement Engineering." *Scientific Reports* 12(1): 1–12. doi:10.1038/s41598-022-11604-y.

Yin, Robert K. 2018. *Case Study Research and Applications: Design and Methods*.

Zhang, Hongliang, Student Member, Lingyang Song, Senior Member, Zhu Han, and Abstract Device-to-device Dd. 2016. "Radio Resource Allocation for Device-to-Device Underlay Communication Using Hypergraph Theory." *IEEE Transactions on Wireless Communications* 15(7): 4852–61. doi:10.1109/TWC.2016.2547862.

Zhang, Yuanzhao, Maxime Lucas, and Federico Battiston. 2023. "Higher-Order Interactions Shape Collective Dynamics Differently in Hypergraphs and Simplicial Complexes." *Nature Communications* 14(1). doi:10.1038/s41467-023-37190-9.

Zhao, Yusheng, Xiao Luo, Wei Ju, Chong Chen, Xian-Sheng Hua, and Ming Zhang. 2023. "Dynamic Hypergraph Structure Learning for Traffic Flow Forecasting." http://arxiv.org/abs/2309.12028.

Zhen, Yaoming, and Junhui Wang. 2021. "Community Detection in General Hypergraph via Graph Embedding." http://arxiv.org/abs/2103.15035.

Zhou, H, L Cao, Z Chen, D Zhou, J Z Huang, B Schölkopf, Y Zhao, et al. 2021. "Hypergraphs: Combinatorics of Finite Sets." *Information Sciences* 29(2): 1–256. doi:10.1016/j.physrep.2009.11.002.

Zykov, A. 1974. 29 Russian Mathematical Surveys *Math-Net.Ru All Russian Mathematical Portal*. https://www.mathnet.ru/eng/rm4449.