Gradual Pattern Extraction from Property Graphs

Faaiz Hussain Shah

▶ **To cite this version:**

Faaiz Hussain Shah. Gradual Pattern Extraction from Property Graphs. Other [cs.OH]. Université Montpellier, 2019. English. NNT : 2019MONTS025 . tel-02446233

HAL Id: tel-02446233
https://theses.hal.science/tel-02446233

Submitted on 20 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

Gradual Pattern Extraction from Property Graphs

Présentée par Faaiz Hussain SHAH
Le 16 juillet 2019

Sous la direction de Prof. Anne LAURENT et Dr. Arnaud CASTELLTORT

Devant le jury composé de

Carmen Gervet, Professeur, Université de Montpellier	Présidente
Ricard Gavalda, Professeur, Universitat Politècnica de Catalunya	Rapporteur
Marie-jeanne Lesot, Maître de Conférence HDR, Sorbonne Université	Rapporteur
Claire Noy, Maître de Conférence, Université Paul Valéry	Examinatrice
Anne Laurent, Professeur, Université de Montpellier	Directrice
Arnaud Castelltort, Maître de Conférence, Université de Montpellier	Co-directeur

UNIVERSITÉ DE MONTPELLIER

"To my love....."

Acknowledgements

The journey of PhD thesis is a life changing experience, i must say *"a life transformation experience"*. It requires a lot of patience, persistence and perseverance.

To reach at this stage, i really want to render my highest gratitude to Prof. Anne Laurent to give me this opportunity to work under her supervision. She is a real mentor. She was always available through emails and meetings whenever i needed guidance. I would also like to thank Dr. Arnaud Castelltort for his guidance at various phases of thesis and in particular sharing his expertise in experiments part of thesis.

I would like to thank Prof. Carmen Gervet for presiding the jury and i present my especial gratitude to Prof. Ricard Gavalda and Dr. Marie-jeanne Lesot for their valuable and explicate feedback on thesis report.

Finally, I am very much thankful for my family for their support; my mother, my father, my wife and sweet kids.

Résumé

Les bases de données orientées graphes (NoSQL par exemple) permettent de gérer des données dans lesquelles les liens sont importants et des requêtes complexes sur ces données à l'aide d'un environnement dédié offrant un stockage et des traitements spécifiquement destinés à la structure de graphe. Un graphe de propriété dans un environnement NoSQL est alors vu comme un graphe orienté étiqueté dans lequel les étiquettes des nœuds et les relations sont des ensembles d'attributs (propriétés) de la forme *(clé:valeur)*. Cela facilite la représentation de données et de connaissances sous la forme de graphes. De nombreuses applications réelles de telles bases de données sont actuellement connues dans le monde des réseaux sociaux, mais aussi des systèmes de recommandation, de la détection de fraudes, du data-journalisme (pour les panama papers par exemple). De telles structures peuvent cependant être assimilées à des bases NoSQL semi-structurées dans lesquelles toutes les propriétés ne sont pas présentes partout, ce qui conduit à des valeurs non présentes de manière homogène, soit parce que la valeur n'est pas connue (l'âge d'une personne par exemple) ou parce qu'elle n'est pas applicable (l'année du service militaire d'une femme par exemple dans un pays et à une époque à laquelle les femmes ne le faisaient pas). Cela gêne alors les algorithmes d'extraction de connaissance qui ne sont pas tous robustes aux données manquantes. Des approches ont été proposées pour remplacer les données manquantes et permettre aux algorithmes d'être appliqués. Cependant, nous considérons que de telles approches ne sont pas satisfaisantes car elles introduisent un biais ou même des erreurs quand aucune valeur n'était applicable. Dans nos travaux, nous nous focalisons sur l'extraction de motifs graduels à partir de telles bases de données. Ces motifs permettent d'extraire automatiquement les informations corrélées. Une première contribution est alors de définir quels sont les motifs pouvant être extraits à partir de telles bases de données. Nous devons, dans un deuxième temps, étendre les travaux existant dans la littérature pour traiter les valeurs manquantes dans les bases de données graphe, comme décrit ci-dessus. L'application de telles méthodes est alors rendue difficile car les propriétés classiquement appliquées en fouille de données (anti-monotonie) ne sont plus valides. Nous proposons donc une nouvelle approche qui est testée sur des données réelles et synthétiques. Une première forme de motif est extrait à partir des propriétés des nœdus et est étendue pour prendre en compte les relations entre nœuds. Enfin, notre approche est étendue au cas des motifs graduels flous afin de mieux prendre en compte la nature imprécise des connaissances présentes et à extraire. Les expérimentations sur des bases synthétiques ont été menées grâce au développement d'un générateur de bases de données de graphes de propriétés synthétiques. Nous en montrons les résultats en termes de temps calcul et consommation mémoire ainsi qu'en nombre de motifs générés.

Abstract

Graph databases (NoSQL oriented graph databases) provide the ability to manage highly connected data and complex database queries along with the native graph-storage and processing. A property graph in a NoSQL graph engine is a labeled directed graph composed of nodes connected through relationships with a set of attributes or properties in the form of $(key : value)$ pairs. It facilitates to represent the data and knowledge that are in form of graphs. Practical applications of graph database systems have been seen in social networks, recommendation systems, fraud detection, and data journalism, as in the case for panama papers. Often, we face the issue of missing data in such kind of systems. In particular, these semi-structured NoSQL databases lead to a situation where some attributes (properties) are filled-in while other ones are not available, either because they exist but are missing (for instance the age of a person that is unknown) or because they are not applicable for a particular case (for instance the year of military service for a girl in countries where it is mandatory only for boys). Therefore, some keys can be provided for some nodes and not for other ones. In such a scenario, when we want to extract knowledge from these new generation database systems, we face the problem of missing data that arise need for analyzing them. Some approaches have been proposed to replace missing values so as to be able to apply data mining techniques. However, we argue that it is not relevant to consider such approaches so as not to introduce biases or errors. In our work, we focus on the extraction of gradual patterns from property graphs that provide end-users with tools for mining correlations in the data when there exist missing values. Our approach requires first to define gradual patterns in the context of NoSQL property graph and then to extend existing algorithms so as to treat the missing values, because anti-monotonicity of the support can not be considered anymore in a simple manner. Thus, we introduce a novel approach for mining gradual patterns in the presence of missing values and we test it on real and synthetic data. Further to this work, we present our approach for mining such graphs in order to extract frequent gradual patterns in the form of "the more/less A_1,..., the more/less A_n" where A_i are information from the graph, should it be from the nodes or from the relationships. In order to retrieve more valuable patterns, we consider fuzzy gradual patterns in the form of "The more/less the A_1 is F_1,...,the more/less the A_n is F_n" where A_i are attributes retrieved from the graph nodes or relationships and F_i are fuzzy descriptions. For this purpose, we introduce the definitions of such concepts, the corresponding method for extracting the patterns, and the experiments that we have led on synthetic graphs using a graph generator. We show the results in terms of time utilization, memory consumption and the number of patterns being generated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

With the provision of ever increasing data rates on Internet and open source technologies to the end users, it has become increasingly challenging for many enterprises to process large volumes of data in an efficient manner. In some cases, traditional database management systems may not be able to store, process, manage, and analyze data to get insight of data for efficient decision making.

Since the inception of artificial intelligence (AI) in 1956 at Dartmouth conference, it has become such an important field that its influence on our daily lives can hardly be overestimated [Lun+07]. The rapid change in AI technologies is transforming various aspects of our lives and human activities from healthcare to education, supply chain, manufacturing, entertainment, etc. It has evolved from simple rule-based systems to machine learning in 1980s to deep learning in recent days and it is continuously evolving into many more aspect of information technology. Thomson Reuters, a well know corporation that serves the decision makers in providing integrated and intelligent information on financial risks for businesses and professionals states *"experts predicts that spending on AI by companies will grow from* $8 *billion in 2016 to* $47 *in 2020, up almost* 600% *"* [Reu19]. The current AI practices and state of the art is shown in Figure 1.1.

Artificial intelligence, data mining, and machine learning techniques are being actively studied for last three decades. More and more applications have been developed these days at a scale to solve the problems in diversified fields including computer science (neural nets and deep learning), medical science (biological databases), chemistry (DNA sequencing and molecules properties analysis), physics (quantum machine learning and experimental physics e.g. Large Hadron Collider experiments at CERN), space technology (data streams or sensor readings) etc. One of the main purpose of all these developments is to emulate human intelligence and implement or apply decision making process for computers.

At the crossroads of big data and artificial intelligence, the objective of data mining can be precisely described as *"the non-trivial extraction of new, implicit, and actionable knowledge from large datasets"* [WHR98]. Formally, it can be defined as *"a process concerned with uncovering patterns, associations, anomalies, and statistically significant structures and events in data"* [Kam09b]. The process involves collecting raw data, cleaning, pre-processing, data transformation into desired formats and then extraction of patterns to gain useful insight from data. The key steps in the process of data mining are briefly shown in Figure 1.2.

As stated in [HPK11], data mining is a highly application-driven domain that has adopted many techniques from various domains such as *"statistics, machine learning, pattern recognition, database and data warehouse systems, information retrieval, visualization, algorithms, high-performance computing, etc.,"*. Figure 1.3 depicts the relationships of data mining with different domains, where each one of them is entirely as vast field of research. Figure 1.4 given in article [Hal+14] shows the Venn diagram representation of AI, data mining, machine learning, and particularly

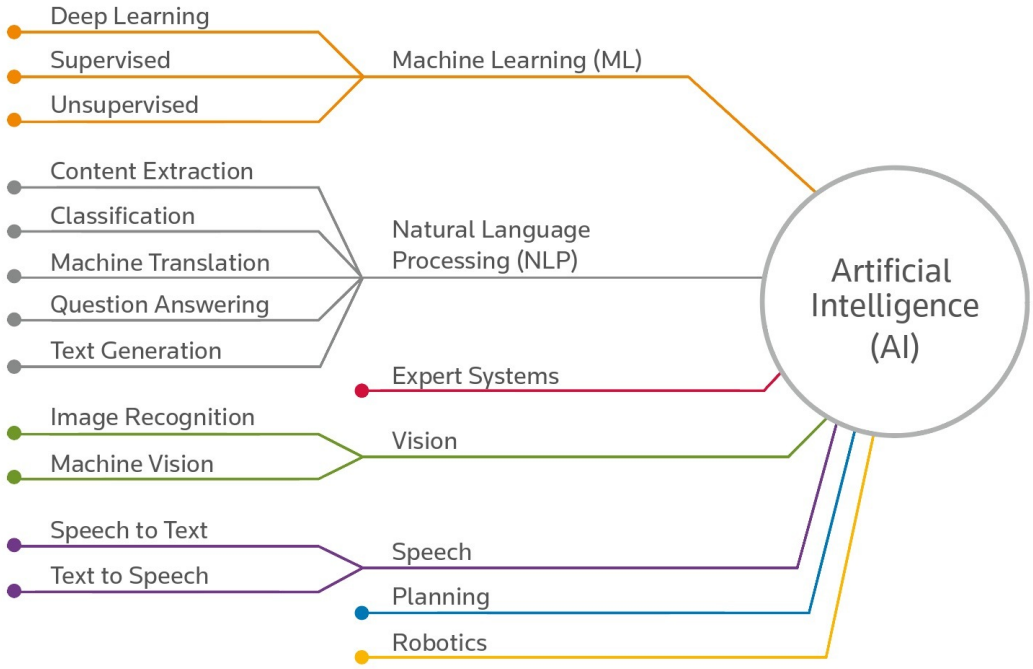

Figure 1.1 – *AI in Practice and Current Dimensions [Log16, Reu16, Ful16]*

data science domain covering databases, statistics, and pattern recognition as well as others. This clearly shows the versatility of data mining domain as well as the complexity involved in a particular mining task. The advantage of this intersection of data mining with several fields make it a source to deal with many open research problems.

The applications of data mining are often closely related to one of four main problems i.e., pattern mining, clustering, classification and outlier analysis [Agg15b]. The data mining process may vary depending on the type of data and the level of complexity for data analysis. It involves to find the relationships between attributes/columns and relationships between objects/rows. Pattern mining (aka association pattern mining) and classification are generally

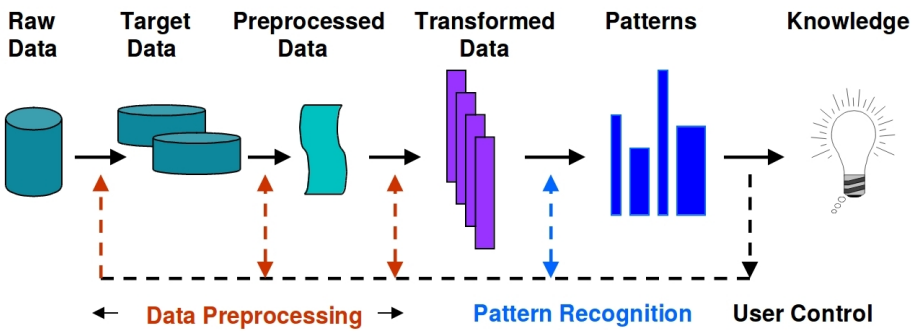

Figure 1.2 – *Data Mining Process [Kam09a]*

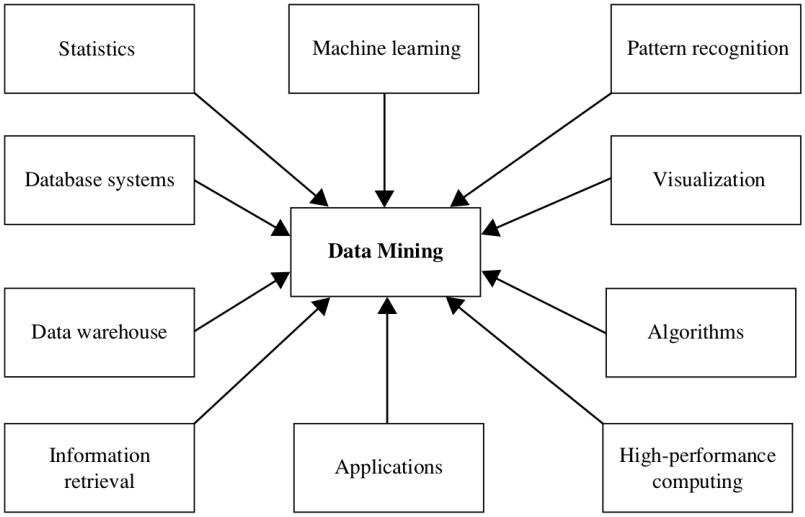

Figure 1.3 – *Data Mining Application Domains [HPK11]*

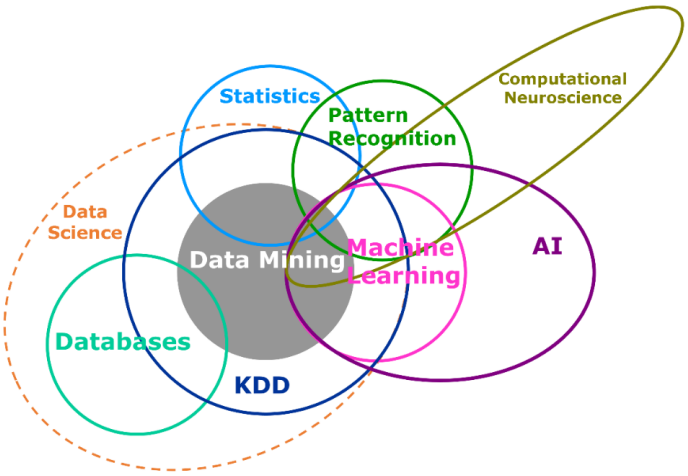

Figure 1.4 – *Data Mining Venn Diagram [Hal+14]*

used to find relationships between attributes, whereas clustering and outlier analysis find relationships between objects [Agg15b]. In data mining, the overall goal is to transform raw data into understandable structures that are presentable in such a way that they may be used for predictive analysis and strategic or capacity planning.

Pattern recognition is one the key step of data mining as shown in Figure 1.2. It involves considering various algorithms for an application to implement it in such a way to make the application more effective and computationally efficient [Kam09a]. A pattern is defined as *"an arrangement or an ordering in which some organization of underlying structure can be said to exist. Patterns in data are identified using measurable features or attributes that have been extracted from the data."* [Kam09a]

Pattern mining is generally referred as frequent itemsets (pattern) mining or frequent subsequence (sequential patterns) mining. It is used to find the items that appear often together and in some sequence in transactional dataset [HPK11]. The data classification mining determines the relationships of a special column with other columns of dataset and hence the process is referred to as supervised. In clustering the main objective is to determine how the values of a subset of rows are related to the values in corresponding columns. The outlier analysis refers to *"identify entries in the rows are different from the corresponding entries in other row"*, and it becomes interesting data or unusual data point [Agg15b].

Frequent pattern mining i.e., mining the data patterns having more occurrences than a predefined threshold is a major domain in data mining. Frequent pattern mining has rapidly extended from transactional databases analysis to the analysis of complex structures having numerical attributes such as sequences, trees or graphs. Therefore, frequent gradual pattern mining poses a new challenge to design efficient algorithms capable of scaling up on huge and complex databases [DLT09].

Gradual pattern extraction is the process of discovering knowledge from databases as comparable attributes of co-variations. These can be increasing variations or decreasing variations. In linguistic expression, it may be represented as, *"the more/less the value of* X_i*,..., the more/less the value of* X_n*"*, where $i = 1, 2, 3...n$, and X_1,X_2,X_3, ...to X_n are numerical ordinal attributes [DLT09]. For instance, a gradual pattern is considered interesting if it occurs frequently i.e., the support of that pattern is greater than the given threshold (minimum support).

Efficient mining of gradual patterns from large numerical databases is a non-trivial task, in particular when considering the scalability issues due to ever increasing volume of data in enterprises. The application of gradual patterns mining can be found in various fields ranging from applications for analyzing client databases for marketing purposes, analyzing patient databases in medical studies, analysis of climate and environment change. The existing gradual pattern mining techniques presented in [DLT09, LLR09, Do+15] are mainly for tabular databases while some other types are emerging, as for instance property graphs.

The concept of graph is studied since late 19^{th} century, however, in last few decades in the field of computer science, the research in applied graph has grown due to prevalence of social networks and networked-based data [RN12]. *"A graph G = (V,E) is a data structure composed of a set of vertices (V) and edges (E)"* or more commonly called as a set of nodes and the relationships that connect them [RN10]. The authors in [RN12, Rod16] present various types of graphs by describing their relevant definition as shown in Figure 1.5.

Graphs provide us better understanding of diversified datasets in the fields such as science, government, and business [FG17, Muñ+17]. As stated in [RWE15], *"The expressive nature of graph structures allows us to model almost all kind of scenarios for computing like construction of a space rocket, system of roads, supply chain, medical history for populations, etc"*. This expressive nature of graph structures helps to model a vast number of real scenarios of governments and business applications [Neo19e, Neo19c].

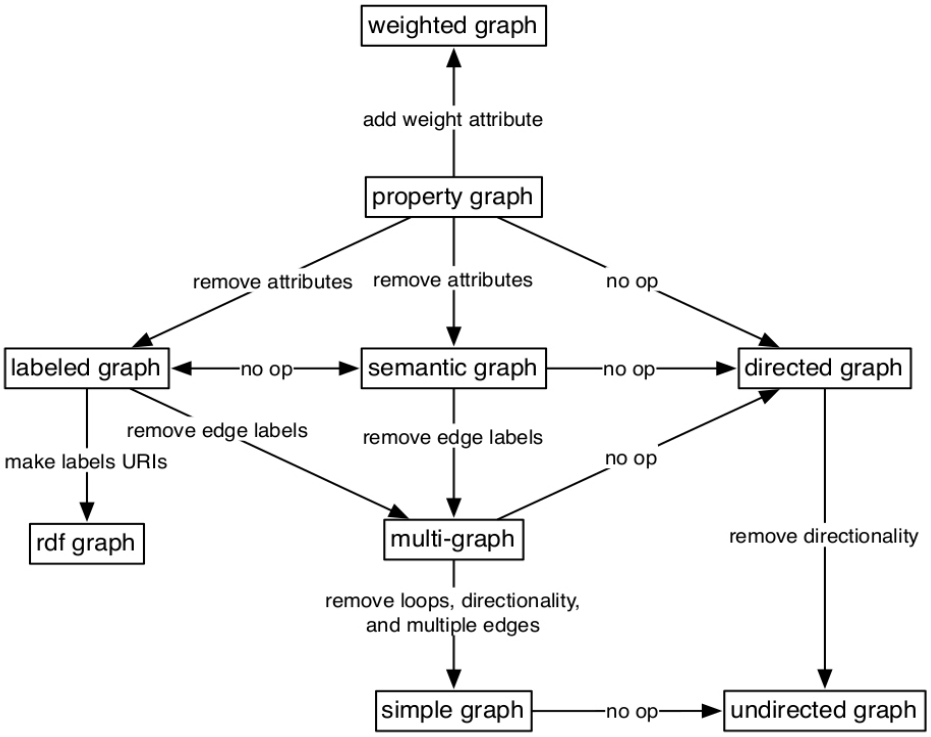

Figure 1.5 – *Graph Type Morphisms [RN10]*

Graphs have been studied from many years [Tru94] but they have only been integrated in database engines in recent years with the so called name as "graph databases". In contrast with relational database, there is no concept of "joins" in a graph database and relationships are treated as first class citizens [RWE15]. When considering the graph database technologies there are two main properties of a graph database are *(i) underlying storage* i.e., the way in which graphs are stored and manged and *(ii) graph processing engine* i.e, the way in which graph queries are processed, typically as an index-free node traversal [RN12, RWE15]. The Figure 1.6 shows an overview of graph database present with respect to these two properties.

Graph modeling helps to identify people's interactions, influences, and exchange of ideas on these social networks has helped to better understand global demographics, political movements and products commercialization [Van14]. Graph modeling is generally an iterative process. A property graph data model enables to represent the data in a natural way with the flexibility to incorporate schema changes as and when required. It does not require a fixed schema prior to the database creation. Hence, it is also referred as semi-structured data. In graph theory, these types of graphs are called *"dynamic"* or *"time-evolving"* graphs, where changes occur in vertices and edges over the time [KBB17, Dem+10, Liu+18]. For building a property graph model, first we identify requirements for the node labels and relevant properties that are to be assigned to nodes. Then we identify and assign the relationships between these nodes and their properties. This process becomes and evolutionary process and nodes and relationships keeps on adding as and when required.

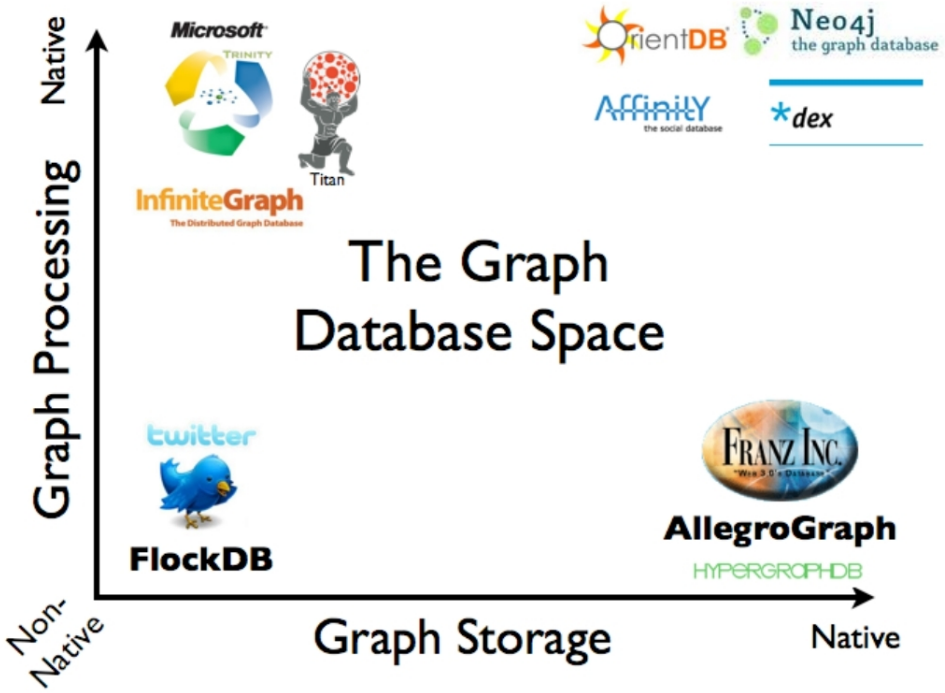

Figure 1.6 – *Graph Databases Overview [RWE15]*

Graph data can be embeded within NoSQL graph oriented databases. The other categories of NoSQL databases include Key-Value store, Column-Family stores, and Document stores respectively. Many companies have developed their in house implementations of graph database systems such as, Facebook's Open Graph, Google's Knowledge Graph, FlockDB by Twitter, and many more [Mil13]. The other open source graph database solutions are Dex, InfiniteGraph, and OrientDB having different levels of maturity. NoSQL graph engines are purpose built systems to store nodes and relationships natively. Nodes or vertices (data entities) are created and linked through relationships (edges), that results in much faster query response for this linked data. The increasing need for such systems has been observed in use cases including *"social networks, recommendation engines, knowledge graphs, fraud detection, network and IT operations, and life sciences"*[Beb+18].

A property graph is a way to *represent data* as graphs. *"In the parlance of graphs, a property graph is a directed, edge-labeled, attributed multi-graph"*[RN12]. A property graph is typically defined as a data model in a graph structure containing nodes and relationships with properties/attributes in form of (key:value) pairs. Nodes are represented as entities and relationships are the edges that connect these entities. A node can have one or more *Labels* that define the role of a node. Relationships between two nodes have one *Type*; they are directional and may contain (key:value) properties as nodes as shown in Figure 1.7. Several relationships can be defined between two nodes, which is different from regular graphs.

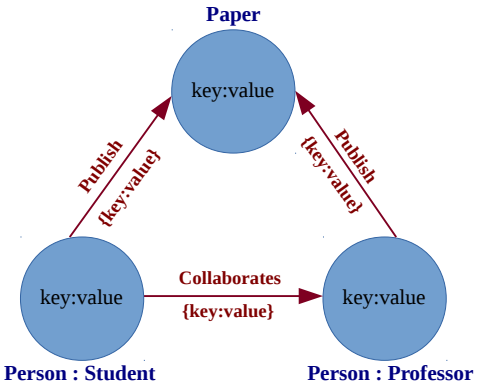

Figure 1.7 – *Property Graph Representation*

1.2 Problem Statement

In recent years, graph data management and mining is gaining a lot of interest in database research community due to its pervasiveness in the fields such as, social networks, knowledge graphs, genome and scientific databases, medical and government record [KBB17]. Also, the research is being actively pursed for last one decade in gradual pattern mining from large numerical tabular datasets along with addressing the scalability issues [DLT09, LLR09, Ayo+10, QLP11, Do+15, AY14, Lau+12, Ngo+18]. This dissertation argues that in case of property graphs, it is not possible to apply directly existing methods for gradual patterns extraction. There are two main reasons:

1. Graphs are not same as tabular data in their meaning and application;
2. Graphs are semi-structure nature of data.

The existing techniques were primarily for small or large single dataset of tabular data without involving relationships. Also, the existing techniques did not address the issue of missing values in datasets, if they appear in attributes. In property graphs, we have nodes with label(s), that have properties/attributes (missing and non-missing) and directed relationships. This arises many questions like how to manage different labeled nodes data for gradual pattern mining, what to do when nodes involve relationships between them and how to do transformation into tabular data, as well as treatment of missing data, if there exist any. To the best of our knowledge, the extraction of gradual patterns from property graphs is a novel idea and has not been studied as yet. The objective of this research work is to address following two main aspects:

1. Defining gradual patterns in the context of property graphs;
2. Mining gradual patterns automatically from property graphs.

In order to achieve these objectives, we address the issue of missing data that arises as a consequence when we are mining graph data. Furthermore, we also define and describe our method for mining fuzzy gradual patterns from property graphs.

We will investigate in detail each of the above mentioned aspect in the coming chapters including extending existing gradual pattern mining algorithms, their implementation results on real and synthetic graph data and discussions to highlight use and application of property graphs.

1.3 Thesis Outline

The rest of the thesis document is organized as follows.

Chapter 2 describes the preliminary concepts and definitions about property graphs, gradual patterns, mining fuzzy gradual patterns, and various types of missing data types and the techniques for handling missing data. All these concepts are discussed as a basis to for our proposed approach presented in chapter 3.

In chapter 3, we describe an overall mining process for gradual patterns in the context of property graphs. A property graph is a data structure containing nodes and relationships with properties in *(key:value)* format. One of our contributions is to propose a formal definition of property graphs as the literature often proposes definitions that do not embed all particularities that we would like to include. We then introduce different forms of gradual patterns in the particular context of such property graphs. Five types are proposed that can be seen as protoforms in the sense of fuzzy summaries. Finally, we propose an extension to fuzzy gradual patterns in order to be able to extract more understandable knowledge. Such patterns can be like *the more the age is "almost 40", the more the experience is "almost high"*.

In chapter 4, we investigate in detail the mining of gradual patterns. Such a process implies to deal with the presence of missing values as it is in the context of property graphs. We discuss the support computation process and the algorithms with related explanations. This chapter is concluded by a section discussing how our work can be integrated in a graph database engine with by a proof-of-concept example.

In chapter 5, we show the experiments and results for our proposed approach. It starts with describing synthetic graph generator tool in detail including its implementation and how it can be used to generate property graphs in NoSQL engine like Neo4j. This GUI-based tool facilitates to generate property graphs for our experiment. Then we describe the setup environment and program execution protocol. We show the details for 8 different datasets on which the experiments are led along with briefly describing them. For all these datasets the results are provided in terms of time utilization, memory consumption and the number of generated patters in the process of mining gradual pattern. The chapter ends with a discussion on the

results.

In chapter 6, we present the conclusion of the entire work covering of all the chapters. In perspectives, we discuss the possible dimensions of the current work by briefly describing the conceptual visualization that could possibly lead to complete research projects.

Chapter 2

Related Work

2.1 Introduction

This work deals with property graphs, gradual patterns, extending fuzzy gradual patterns and missing data management. In this chapter, we will first review existing works related to these topics. In section 1.1, we briefly described about the graphs, graph database, property graphs in Neo4j graph database and the purpose to choose it to perform gradual pattern mining. Furthermore, we describe and define property graphs, state of the art about gradual pattern extraction algorithms and gradual pattern mining from property graphs. We also describe mining fuzzy gradual patterns from property graphs and treatment of missing data in property graphs so as to mine gradual patterns.

2.2 Graph Databases

Graph databases have given a new way of modeling and traversing interconnected data and have applications in social graphs, recommendation systems, and bioinformatics [Mil13]. A comparison between the relational databases and graph databases focusing on the aspects of data structures, data models, query facilities and limitations of is presented in [Mil13]. The decision to choose between the relational database and graph database is primarily based on the requirements of systems that may be utilizing these database.

A NoSQL database engine is a system specifically designed to store, manage, and process such kind of graph-like data. Such kind of systems are gaining attention as mentioned in [SA11] "*increasing usage of graph data structure for representing data in different domains such as: chemical compounds, multimedia databases, social networks, protein networks and semantic web.*" Today, the well known NoSQL database systems are: Amazon's Neptune [Ama19], Microsoft's Cosmos [Mic19], Titan [Aur19], and Neo4j [Neo19a]. DB engine ranking [DBE19] provides the current ranking of graph database engines in the market according to the popularity and is updated monthly. The chart and list depiction of April 2019 are shown in Figures 2.1 and 2.2 respectively. This clearly shows that Neo4j is a leader in Graph databases for quite some time. Therefore, based on research survey, popularity and more importantly of being open source, we choose Neo4j as graph database engine to create property graphs and apply gradual pattern mining algorithms to analyze the patterns and evaluate the results.

2.2.1 Neo4j Graph Database

Neo4j is an open source NoSQL graph database implementation that is highly scalable and leverages data as well as relationships. Neo4j stores data as connected data that provides the flexibility of adding a new node or a new relationship without compromising or migration of existing data. Relationships are considered as much important as the nodes. Relationships are used to perform the node traversing thereby eliminating the need of complex join operations.

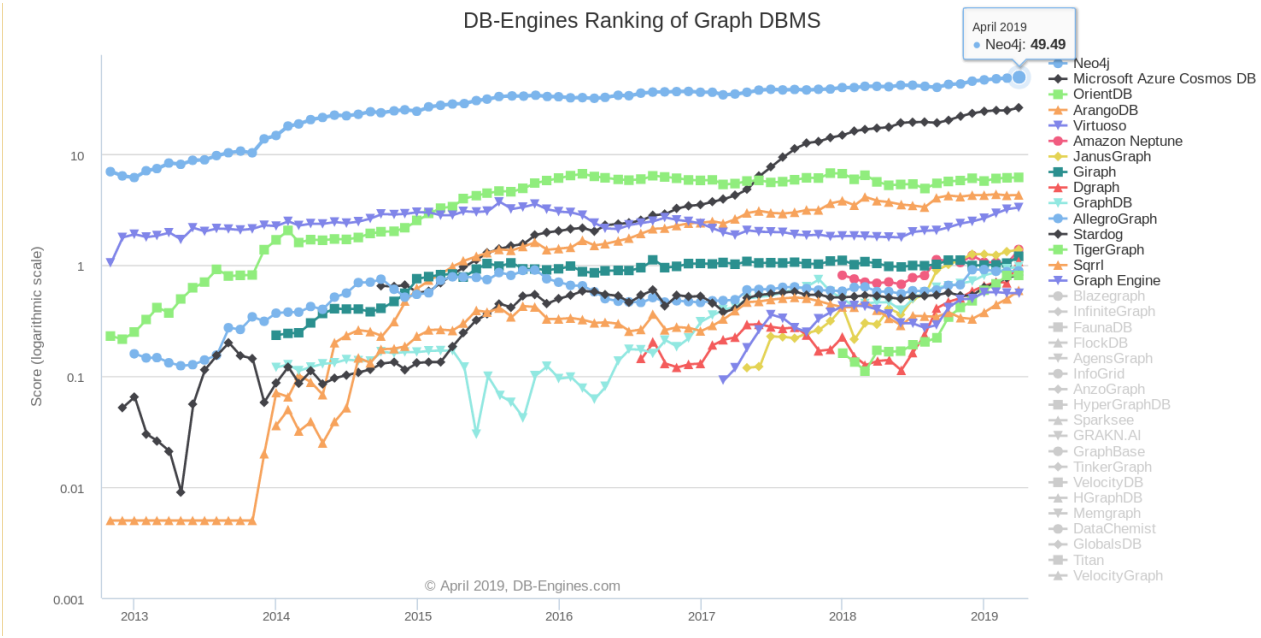

Figure 2.1 – *DB-Engines Ranking of Graph DBMS -Trend Chart April 2019*

Rank Apr 2019	Rank Mar 2019	Rank Apr 2018	DBMS	Database Model	Score Apr 2019	Score Mar 2019	Score Apr 2018
1.	1.	1.	Neo4j	Graph	49.49	+0.91	+8.59
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	26.28	+1.45	+9.09
3.	3.	3.	OrientDB	Multi-model	6.19	+0.06	+0.55
4.	4.	4.	ArangoDB	Multi-model	4.29	+0.03	+0.49
5.	5.	5.	Virtuoso	Multi-model	3.31	+0.12	+1.51
6.	↑ 8.	↑ 7.	Amazon Neptune	Multi-model	1.39	+0.36	+0.70
7.	↓ 6.	↑ 13.	JanusGraph	Graph	1.38	+0.06	+1.09
8.	↓ 7.	↓ 6.	Giraph	Graph	1.20	+0.16	+0.16
9.	↑ 13.	↑ 17.	Dgraph	Graph	1.08	+0.39	+0.94
10.	↓ 9.	10.	GraphDB	Multi-model	0.97	+0.04	+0.51
11.	↓ 10.	↓ 8.	AllegroGraph	Multi-model	0.89	+0.02	+0.31
12.	12.	↓ 9.	Stardog	Multi-model	0.81	+0.04	+0.29
13.	↓ 11.	↑ 15.	TigerGraph	Graph	0.81	+0.00	+0.64
14.	↑ 17.	↓ 12.	Sqrrl	Multi-model	0.59	+0.10	+0.20
15.	↓ 14.	↓ 11.	Graph Engine	Multi-model	0.56	0.00	+0.16
16.	↓ 15.	↑ 19.	Blazegraph	Multi-model	0.56	+0.01	+0.43
17.	↑ 18.	↑ 22.	InfiniteGraph	Graph	0.40	+0.01	+0.29
18.	↓ 16.	↑ 21.	FaunaDB	Multi-model	0.37	-0.15	+0.26
19.	↑ 20.	↓ 18.	FlockDB	Graph	0.27	+0.01	+0.14
20.	↑ 23.	↑ 25.	AgensGraph	Multi-model	0.23	+0.06	+0.20
21.	↓ 19.	↓ 20.	InfoGrid	Graph	0.22	-0.05	+0.11
22.	22.		AnzoGraph	Graph, Multi-model	0.22	+0.04	
23.	↓ 21.	↓ 16.	HyperGraphDB	Graph	0.21	+0.03	+0.06
24.	24.	↓ 14.	Sparksee	Graph	0.15	+0.03	-0.06
25.	↑ 27.	↑ 27.	GRAKN.AI	Multi-model	0.11	+0.01	+0.09
26.	↑ 28.	26.	GraphBase	Graph	0.11	+0.03	+0.08
27.	↓ 25.	↓ 24.	TinkerGraph	Graph	0.10	+0.00	+0.06
28.	↓ 26.	↓ 23.	VelocityDB	Multi-model	0.08	-0.02	+0.01
29.	↑ 30.	29.	HGraphDB	Graph	0.05	+0.04	+0.03
30.	↓ 29.		Memgraph	Graph	0.05	+0.00	
31.	31.		DataChemist	Graph, Multi-model	0.02	+0.02	

Figure 2.2 – *DB-Engines Ranking of Graph DBMS -Tabular April 2019*

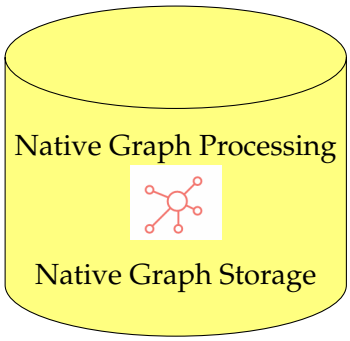

Figure 2.3 – *Graph Native Storage and Processing*

Neo4j's native graph storage as shown in Figure 2.3 is used to store the data as graphs and optimized for managing graphs. *"The graph processing engine is used to provide basic graph operations and algorithms to deliver constant, real-time performance, helping enterprises to build intelligent applications to meet today's evolving data challenge"* [RWE15]. Another main performance characteristic of a graph database is the graph traversal or pattern matching query independent of the data set size. This is achieved primarily due to native storage of data as graphs.

Graph traversal across the nodes using relationships is one of the key features of graph databases. Neo4j does not require a join operation because every node in graph has an explicit relationship joined between two nodes and it only requires traversal across the graph to match the required pattern. This feature results in efficient query performance and better response time. In the case of relational database systems, we need to create a separate link table in which we store foreign keys of two tables to link them together. Another advantage of property graph model as compared to relational model is that it does not require creating a schema prior the creation of database. It allows more flexibility to incorporate schema changes as and when required.

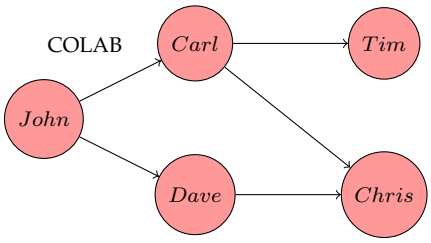

Figure 2.4 – *Explicit relationships*

Neo4j helps to model dynamic complex relationships in a maneuverable form of connected data that can be easily understandable like white board model. Neo4j is gaining more interest in the field of data science, specifically for defining complex join-intensive and path traversal queries over the graph [RWE15].

2.2.2 Cypher Query Language

Neo4j is a NoSQL graph database that uses *"Cypher"*, a declarative query language that uses ASCII-art syntax. The queries written in declarative query language allow to declare a required pattern to retrieve data from the database as opposed to imperative query language such as SQL where we have to specifically tell the database what to do in order to retrieve the required data. Cypher query language is composed of specific clauses to perform queries from graph database.

Table 2.1 shows the different clauses used to query the graph data in Noe4j. Cypher uses these clauses to change or update the graph data by adding or removing the nodes, relationships and properties of a graph. Cypher also provides several aggregate functions to calculate aggregate data analogous to 'GROUP BY' in SQL. These include but are not limited to COUNT, COLLECT, AVG, MAX, MIN, SUM, etc.

Clause	*Purpose*
MATCH	To match the required pattern for result
RETURN	To return the matched data
WHERE	Provides criteria for filtering pattern matching results
CREATE	Create nodes and relationships
DELETE	Removes nodes, relationships, and properties
SET	Sets property values
UNION	Merges results from two or more queries
REMOVE	Remove a label or property from node
CALL	Standalone call to built-in procedures
LOAD	To import external files in graph database and create nodes
ORDER BY	To the result of return result in ascending or descending order

Table 2.1 – *Most Used Neo4j Cypher Clauses*

In the following section we describe property graphs using Neo4j graph database engine to explain the relavent concepts including the running example created in Neo4j.

2.3 Property Graphs

A *Property Graph (PG)* refers to a data model in which data has *(key:value)* pairs. A property graph data model enables us to represent the data in natural way in form of graph structure of vertices (nodes) and edges (relationships), as shown in Figure 2.5. We use property graphs to represent data. Entities are represented as nodes or objects having one or more labels that describe the type or class of nodes and a node can have one or more (key:value) properties, hence the graph is called as a labeled property graph model. Each node can have one or more relationships that connect the nodes. Relationships have a single type and they can also store (key:value) properties. Relationships have a direction.

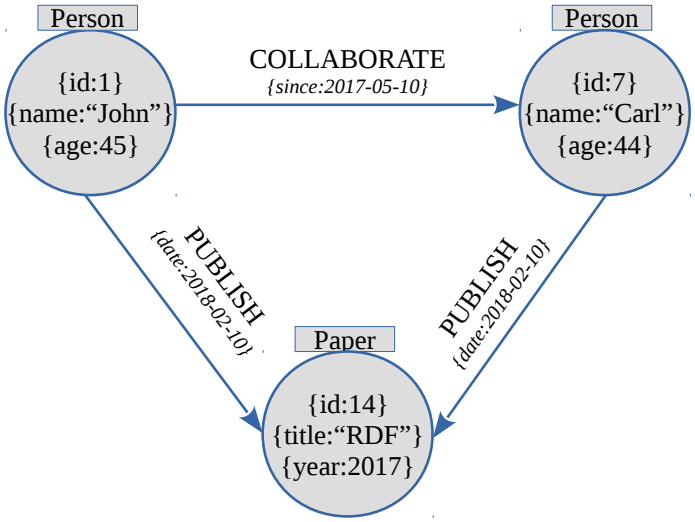

Figure 2.5 – *Property Graph with data as (key:value) pairs*

Property graph modeling is generally an iterative process that provides a flexibility to change the graph schema without altering the existing graph structure. This feature is very useful particularly for agile application development. For building a property graph data model it is important to identify the nodes and what properties are relevant to the nodes that are to be assigned. This is generally termed as *user requirement gathering* for developing the initial graph structure. We identify the uniquely identifiable able attributes to define the indexes and constraints accordingly. Then we identify the relationships between these nodes and possible properties these relationships can have between each node. This process is repeated whenever needed.

To present the definitions, we create a running example property graph in Neo4j. The graph schema of the property graphs is shown in Figure 2.6 showing three labeled nodes with relationship types. The overall property graph visualization for this graph schema with nodes and relationships is shown in Figure 2.7. The property graph shows 14 nodes with 3 labels namely person, paper and projects. There are 30 relationships with 3 relationship types namely collaborate, publish and receive_grant.

2.3.1 Schema-less Nature of Property Graphs

The interesting thing is the property graph is that there is no fixed schema and any node(s) or relationship(s) can be removed without affecting the entire graph. Only the links between associated nodes will diminish and rest of the structure will be intact. Similarly, the addition of nodes and relationships can be done on the fly whenever required. For example, in case of Person label, assume that you have a table named "Person" in a relational database and you need to add an attribute, say "Phone_Number". To do so, you need to define the attribute first

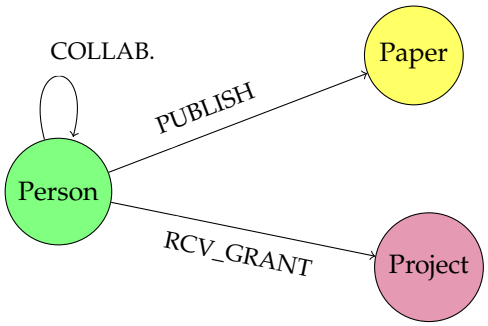

Figure 2.6 – *Running Example Graph Database Schema*

by using ALTER table and then you can add the value for the respective record. In case of property graph, there is no need to define something before for adding an attribute/property for a node (record). You just add the property right away independent of any structure as well as without affecting relationships. That is why property graphs are often referred as schema less graphs and semi-structured data graphs.

We use Neo4j *cypher-shell* utility to query the list of nodes for each label. The *Cypher* query along with the output result for label "Person" is shown in Listing 2.1. In the query we write the names of each property to make the output more presentable. Similarly, the query and output for label "Paper" is shown in Listing 2.2 and for label "Project" it is shown in Listing 2.3 respectively.

```
neo4j> MATCH (n:Person)
       RETURN ID(n) AS ID , n.name AS NAME, n.age AS AGE,
       n.desig AS DESIGNATION, n.expr AS EXPERIENCE,
       n.sal AS SALARY;
+-------------------------------------------------------------------------+
| ID | NAME    | AGE  | DESIGNATION | EXPERIENCE | SALARY |
+-------------------------------------------------------------------------+
| 1  | "John"  | 45   | "PR"        | 13         | 3500   |
| 2  | "Dave"  | NULL | "MCF"       | 7          | 2500   |
| 3  | "Emily" | 30   | "ETU"       | 5          | NULL   |
| 6  | "Tim"   | NULL | "MCF"       | 10         | 2800   |
| 7  | "Carl"  | 44   | "PR"        | 14         | 3800   |
| 9  | "Chris" | 38   | "MCF"       | 9          | 2600   |
| 10 | "David" | 32   | "ETU"       | 5          | 1400   |
+-------------------------------------------------------------------------+
7 rows available after 10 ms, consumed after another 1 ms
```

Listing 2.1 – *Person Node Details*

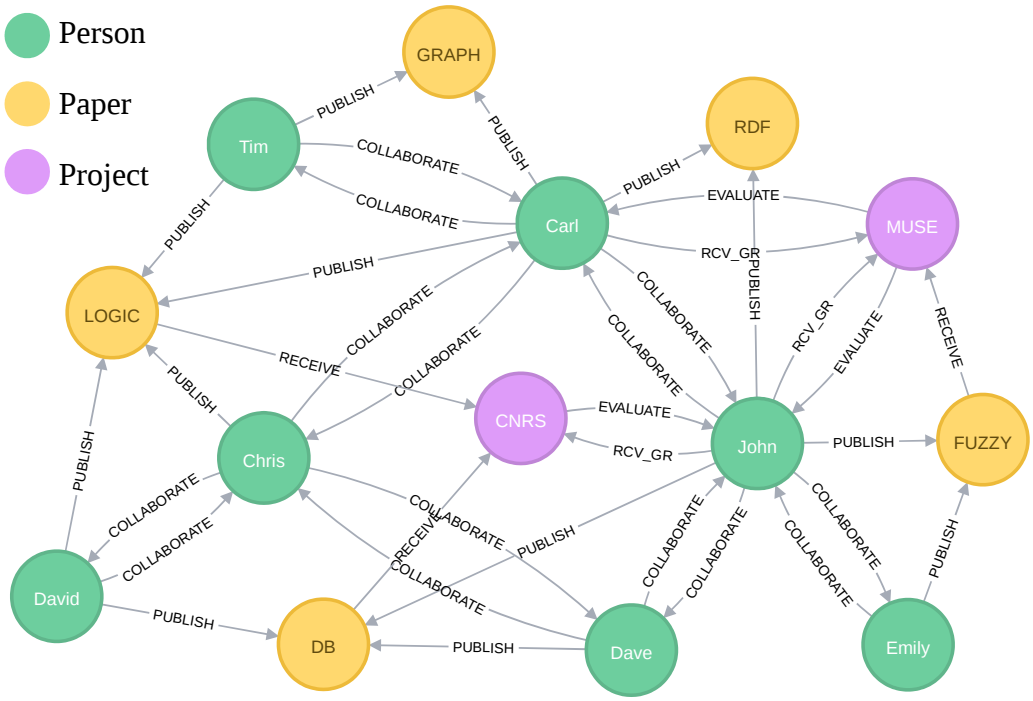

Figure 2.7 – *Running Example Property Graph Visualization*

```
neo4j> MATCH (n:Paper)
       RETURN ID(n) AS ID , n.pr_title AS TITLE,
       n.type AS TYPE, n.year AS YEAR, n.Citations AS CITATIONS,
       n.Impact_Factor AS IMPACT_FACTOR;
+-----------------------------------------------------------------------------+
| ID | TITLE     | TYPE           | YEAR     | CITATIONS | IMPACT_FACTOR |
+-----------------------------------------------------------------------------+
| 11 | "GRAPH"   | "conference"   | "2015"   | 10        | NULL          |
| 12 | "LOGIC"   | "journal"      | "2016"   | 30        | 4.5           |
| 13 | "FUZZY"   | "conference"   | "2015"   | 5         | NULL          |
| 14 | "RDF"     | "conference"   | "2017"   | NULL      | NULL          |
| 15 | "DB"      | "journal"      | "2016"   | 50        | 2             |
+-----------------------------------------------------------------------------+
5 rows available after 1 ms, consumed after another 1 ms
```

Listing 2.2 – *Paper Node Details*

```
neo4j> MATCH (n:Project)
       RETURN ID(n) AS ID , n.prg_title AS PROJECT_TITLE,
       n.amount AS AMOUNT, n.year AS YEAR;
+-----------------------------------------+
```

```
| ID | PROJECT_TITLE | AMOUNT | YEAR    |
+-------------------------------------------+
| 16 | "MUSE"        | 30000  | "2017" |
| 17 | "CNRS"        | 50000  | "2016" |
+-------------------------------------------+
2 rows available after 11 ms, consumed after another 1 ms
```

Listing 2.3 – *Project Node Details*

2.3.2 Graph Pattern Matching

Pattern matching e.g., "()-[]->()" is also one of the main feature of cypher to match the desired graph structures and to retrieve the desired information pertaining to nodes and relationships in a graph. An example cypher query to find a node labeled Person, whose name property value is 'John' and he publishes a journal paper can be written as shown in Listing 2.4. The output of this query from Neo4j console is shown in Figure 2.8.

```
MATCH (p:Person) - [v:PUBLISH] -> (q:Paper)
WHERE p.name ="John" AND q.type ="journal"
RETURN p AS PersonName, q AS Paper
```

Listing 2.4 – *Example Cypher Pattern Query*

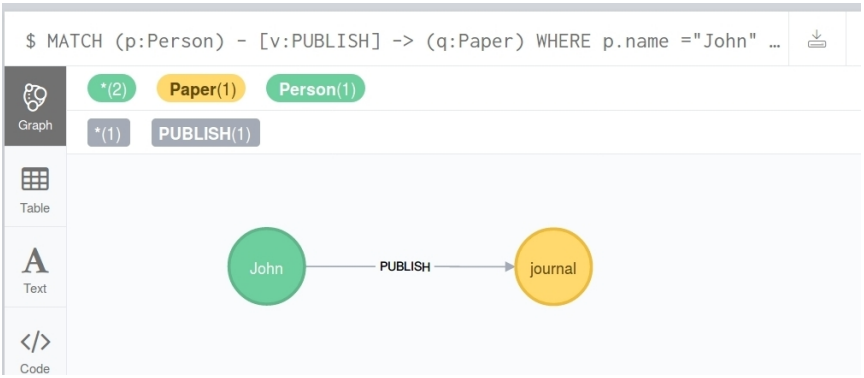

Figure 2.8 – *Output Example Cypher Pattern Query*

Similarly, if we want to see all the persons who have published in a journal, the Cypher query can be written as shown in Listing 2.5. The output of the query is shown in Figure 2.9.

```
1 MATCH (p:Person) - [r:PUBLISH] -> (q:Paper)
2 WHERE q.type ="journal"
3 RETURN p AS PersonName, q AS Paper
```

Listing 2.5 – *All Persons who Published in Journal*

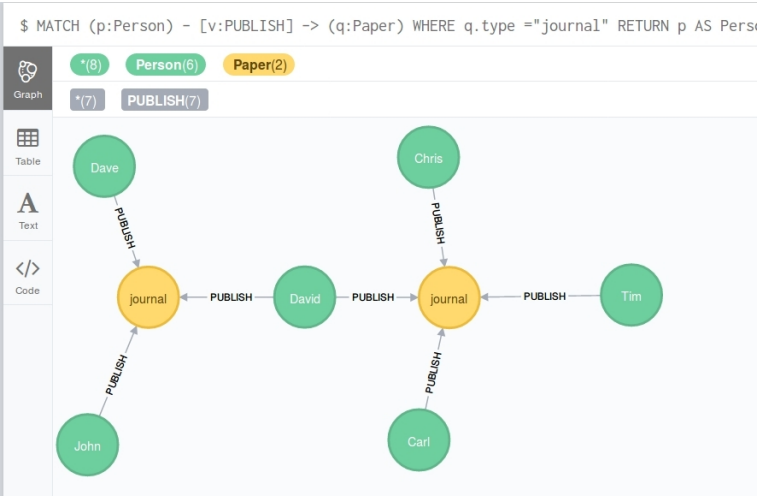

Figure 2.9 – *Cypher - All Persons who Publish Paper*

In the whole property graph, if we like to see who is connected with whom and in what frequency of relationships, then we can use the query shown in Listing 2.6. It can also be said as topology extraction in tabular form.

After discussing briefly about the property and pattern matching in property graphs, we proceed in the next section to describe the main concepts of gradual pattern mining, definitions and algorithms presented in the literature on the topic.

2.4 Gradual Pattern Mining

The knowledge discovery process was first presented in [FPS96] as shown in Figure 2.10. In the process of knowledge discovery from databases, data mining is one of the key steps that involves applying algorithms in order to extract patterns. Data mining tasks include clustering, classification, association rule mining, etc. The association rule mining involves frequent pattern mining/extraction. Following are some of the categories of frequent pattern mining:

- frequent Patterns,
- fuzzy Patterns,

```
1
2  neo4j> MATCH (n) -[ r ]-> (m)
3         RETURN labels(n) AS SrcLabel , type(r) AS Relationship,
4         labels(m) AS DstLabel, Count (*) AS Relationships_Count;
5  +-------------------------------------------------------------------------+
6  | SrcLabel     | Relationship   | DstLabel     | Relationships_Count |
7  +-------------------------------------------------------------------------+
8  | ["Person"]   | "COLLABORATE"  | ["Person"]   | 14                  |
9  | ["Project"]  | "EVALUATE"     | ["Person"]   | 3                   |
10 | ["Paper"]    | "RECEIVE"      | ["Project"]  | 3                   |
11 | ["Person"]   | "PUBLISH"      | ["Paper"]    | 13                  |
12 | ["Person"]   | "RCV_GR"       | ["Project"]  | 3                   |
13 +-------------------------------------------------------------------------+
14
15 5 rows available after 7 ms, consumed after another 0 ms
16 neo4j>
```

Listing 2.6 – *Graph Schema Summary with Relationships*

- gradual Patterns,
- sequential Patterns.

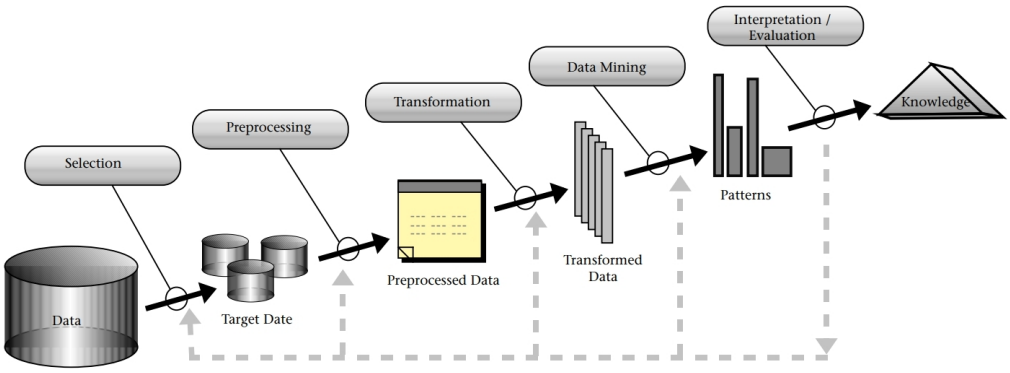

Figure 2.10 – *The KDD Process [FPS96]*

All these approaches for mining patterns differ in their application on the basis of the *type of data* from which the pattern extraction is performed.

Gradual pattern mining is an extension of frequent pattern mining. In this section, first we briefly describe frequent pattern mining and its related definitions as a basis for describing gradual patterns. Then we describe state of the art algorithms for gradual patterns mining. In particular, we explain the two existing algorithms namely GRITE and GRAANK that were primarily used on tabular data to extract gradual patterns. The GRadual ITemset Extraction (GRITE) [DLT09] is based on precedence graph, whereas GRAANK [LLR09] is employing the

approach based on rank correlation and the concept of concordant/discordant pairs. We also describe their related definitions and support computation method for these algorithms.

In data mining process from the raw data to the patterns and knowledge discovery, the pattern mining/extraction poses several questions such as [Kam09a]:

- *"Is it possible to modify existing algorithms, or design new ones, that are scalable, robust, accurate, and interpretable?"*
- *"Can these existing algorithms be applied effectively and efficiently to complex data?"*

To address these questions, research is being continuously carried out to present more optimized solutions with the evolution of technology.

2.4.1 Frequent Pattern Mining

Frequent pattern mining (*aka frequent itemset mining*) is defined as a process to find the items that appear often together in some sequence in transactional database [ABA14]. Frequent pattern mining was first introduced in 1993 in [AIS93] and since then it is *"one of the most intensively investigated problems in terms of computational and algorithmic development"* [ABA14].

"The problem was originally proposed in the context of market basket data in order to find frequent group of items that are bought together" [Agg14].

Frequent patterns play an important role in many data mining tasks such as association rules, correlations, sequences, classifiers, clusters etc. [Goe03]. Association rule mining is generally referred as frequent itemsets mining and it is used to discover the association rules (Section 2.4.3) between items in the transactional data [AIS93].

An itemset is considered to be frequent if its *support* is at least equal to user defined *minimum support threshold*. Therefore, in frequent pattern mining the actual task is to determine itemsets that have requisite level of *support* [Agg15a]. The definition of *support* and its calculation is described as follows.

Definition of Support

As stated in [ABA14], let D be a transaction database which contains a set of n transactions as $D = \{t_1, t_2, ..., t_n\}$. Each $t_i \in D, \quad \forall_i = \{1....n\}$ consists of a set of items, say $t_i = \{x_1, x_2, x_3, ...x_m\}$. A set $I \subseteq t_i$, is called an itemset. A k-itemset is an itemset that contains exactly k items, where k is the cardinality of set of items.

tid	Set of Items	Binary Representation
t_1	a,b,e	110010
t_2	d,e,f	000111
t_3	a,c,d,e	101110
t_4	d,e,f	000111
t_5	c,e,f	001011

Table 2.2 – *Market Basket Dataset*

Definition 2.1 (Support)**.** The support of an itemset I is defined as the fraction of the transactions in the database $D = \{t_1, t_2, ..., t_n\}$ that contains I as a subset [Agg15a].

$$Support(I) = \frac{|Occurrences\ of\ I\ in\ D|}{|D|} \tag{2.1}$$

Support Calculation

Considering the Table 2.2 of market basket dataset, the attributes for binary representation are arranged in the order {a,b,c,d,e,f} and each item that is placed in basket is set to 1. Since there are 5 transactions in the database, the *support* for itemsets $I_x = \{a, e\}$ and $I_y = \{c, f\}$ is :

$$Support(I_x) = \frac{2}{5} = 0.4 \quad , \quad Support(I_y) = \frac{1}{5} = 0.2$$

How to choose *minimum support threshold* ?

Considering the above support calculation, if the required *minimum support threshold* value is 0.3, then the itemset I_y will not be considered as frequent because it does not fulfill the minimum support level. In this example, I_x is a frequent 2-itemset that satisfies the minimum support.

Therefore, the choice of *minimum support threshold* is a crucial aspect when discovering the frequent patterns. Because in case of a smaller value of minimum support threshold, a large number of patterns will be generated and if the minimum support threshold is very high then few or no patterns can be found.

Algorithms must be designed to extract all frequent itemsets from databases efficiently. For frequent pattern mining algorithms the search space for finding the frequent patterns becomes very larger (of the order of $2^{|t|}$). In the above toy dataset of market basket, the search space will be $2^6 = 64$. Therefore, to limit the search space, algorithms apply pruning based on the anti-monotonicity property of support that allows to achieve better computational efficiency.

2.4.2 Anti-monotonicity Property

The anti-monotonicity property of support states that, *"the support of every subset J of I is at least equal to that of the support of itemset I"* [Agg15a].

$$Support(J) \geq Support(I) \, \forall \, J \subseteq I \tag{2.2}$$

In other words, it can be said that support of an itemset never exceeds the support of its subset. The anti-monotonicity property of support is also referred to as *"downward closure property"* which implies that "if an itemset is frequent, then all of its subset will also be frequent, and if an itemset is infrequent then all of its superset will also be infrequent".

2.4.3 Association Rules and Gradual Dependencies

An association rule is an implication of the form shown in Equation 2.3, [AS+94].

$$X \implies Y, \, where \, X \subset t_i, \, Y \subset t_i, \, and \, X \cap Y = \emptyset \tag{2.3}$$

It says that a transaction that contains the set of items X is likely to contain the items Y as well. By using a measure know as *confidence*, frequent patters can be used to generate association rules [Agg15a]. It is defined as follow, .

Definition 2.2 (Confidence). *" Let X and Y be two set of items. The confidence conf(X ∪ Y) of the rules X ∪ Y is the conditional property of X ∪ Y occurring in a transaction, given that the transaction contains X. Therefore, the confidence conf(X ∪ Y) is defined as follows:"* [Agg15a].

$$conf(X \cup Y) = \frac{Support(X \cup Y)}{Support(X)} \tag{2.4}$$

Although the original techniques for mining association rules were designed for binary attributes but in practice a database does not contain only the binary attributes but also the quantitative attributes therefore, *"the algorithms can be extended to attributes with values ranging on (completely) ordered scales, e.g. cardinal or ordinal attributes"* [Hül02]. These quantitative attributes then lead *quantitative association rules*. Several works have been proposed in the literature to deal with such data and patterns, as for instance [SA96] for mining patterns like "(Age: 30..39) and (Married: Yes) → (NumCars: 2)". Such patterns have been extended to fuzzy intervals in order to discover patterns like "(Age: young) and (Married: Yes) → (NumCars: low)" [AYL11].

For mining quantitative data, [Hül02] proposes a new type of rules to express a kind of *"tendency" aka a gradual dependence between attributes*.

Gradual Dependency

In [Hül02], a first interpretation of *"gradual dependency"* is presented in which it is expressed as co-variation constraint such that *"the more A, the more B holds if an increase in A comes along with an increase in B"* [LLR09]. To extract such relationships, a linear regression analysis of two attributes is presented. Hence, a *"gradual dependency"* is defined as a pair of gradual itemsets on which a causality relationship is imposed [LLR09]. For instance, it takes the form *"the higher the experience, 'then' the more the salary"* meaning that an increase in experience implies a salary increase which as a result *"breaks the symmetry of the gradual itemsets in which all items play the same role"* [LLR09]. In the next section we discuss the definition of gradual item, gradual pattern, and the computation of support for gradual patterns.

2.4.4 Formal Definition of Gradual Pattern (Itemset)

The gradual pattern mining is a process to discover frequent co-variations of the form "the more/less the X_i, ..., the more/less theX_n" [DLT09]. These co-variation can be between two or more than two attributes such as: *"The higher the age, the higher the salary, the higher the tax"* or *"The more the intensive-diet, the less the physical-activity-hours, the more the weight"*.

A gradual item is defined as follows.

Definition 2.3 (Gradual Item)**.** Let τ be a set of items, $i \in \tau$ be an item and $\star \in \{\uparrow, \downarrow\}$ be a comparison operator. A gradual item $i\star$ is defined as an item i associated to an operator $\star$ [DLT09].

Consequently, a gradual pattern (aka gradual itemset) is defined as follows:

Definition 2.4 (Gradual Pattern)**.** A gradual pattern $P = (i_1\star^1, \ldots, i_k\star^k)$ is a non empty set of gradual items. A k-itemset is an itemset containing k gradual itemsets [DLT09].

Example 1. For example, let us consider the pattern *"the higher the age, the higher the number of publications"*, formalized by the itemsets from Figure 2.5 are:

$$P1 = (Age \uparrow\ ,\ Number of Publications \uparrow)$$

A gradual pattern is said to be frequent if its support calculation is greater than or equal to user defined *minimum support threshold*. In the following section we discuss different methods for computing the support.

2.4.5 Support Measure for Gradual Pattern

In the classical pattern mining the quality of patterns is assessed by the "minimum support" measure. Therefore, when dealing with gradual patterns, the definitions of the classical support measure must be extended.

In [Ber+07], and interpretation of gradual dependencies is presented as constraints imposed to the order induced by the attributes and not to their numerical values[LLR09]. Considering the attributes shown in Table 2.3, the order-based definition of gradual dependency defined in [Ber+07] as follows.

Definition 2.5 (Order-based gradual dependency)**.** A gradual dependency of the form *the more the Age, the more the Experience* holds if,

$$\forall (x, x') \in D,\ Age(x) < Age(x')\ implies\ Experience(x) < Experience(x') \tag{2.5}$$

where $Age(x)$ denotes the value taken by attribute Age for object x such that x proceeds x'.

It is clear from the above definition that it takes the implication relationship between itemsets and the authors propose to extract such gradual dependencies to formulate the association rules [LLR09]. And to mine these gradual rules, the authors in [Ber+07] use operators $\{<, >\}$ with attributes e.g., $\{Age <\}$. *The support is consequently expressed as the number of object pairs respecting the order divided by the total number of object pairs of the dataset* as given below [DLT09, LLR09].

$$Supp(A_1\star^1, \ldots, A_k\star^k) = \frac{1}{|D|}|\{o = (x, x') \in D/\forall y \in [1, k] A_y(x) \star_y A_y(x')\}| \tag{2.6}$$

By relying on the Definition 2.5, the authors in [DLT08], present a heuristic based method to extract gradual association rules. To build these rules the first step is to compute the support. Let P be a gradual pattern (itemset), where $P = (A_1\star^1, \ldots, A_k\star^k)$, the support of P is given as follows.

$$Supp(P) = \frac{1}{|D|} \max_{L_i \in \mu} |L_i| \tag{2.7}$$

The support measure $Supp(P)$ is the maximal number of rows $\{r_1, ..., r_m\}$ in D for which there exist a permutation π such that $\forall x \in [1, m-1], \forall y \in [1, k]$, it holds $A_y(r_{\pi_x}) \star_y A_y(r_{\pi_{x+1}})$, denoting μ as the set of all such sets of rows [LLR09].

Taking into account the support measure given in Equation 2.7, the authors in [DLT09] present a new approach based on precedence graph to extract gradual patterns. The algorithm to extract such patterns is named as GRITE which is described as follows.

2.4.6 GRITE Algorithm

The GRITE is acronym of *GRadual ITemset Extraction (GRITE)*, which is an algorithm proposed in [DLT09] for extracting gradual patterns from numerical tabular datasets. The main features of algorithm include:

- Associate the binary matrix to each item and perform the join operation of two items by using Hadamard product method [1]
- Compute the support of resulting itemset

The algorithm uses the vertical bitmap representation presented in [Ayr+02] to store the ordering in the binary matrix. In the proposed method the binary matrix is used to represent the ordering of gradual items. The order is associated with each gradual item (i.e., item with its variation e.g., $Age \uparrow$) and represented in a binary matrix as per following principle.

"If there exists an order relation between two tuples a and b of an item, then the bit corresponding to the line of a and the column position of b is set to 1, and to 0 otherwise" [DLT09].

Let us consider the sample data as shown in Table 2.3 to show the binary matrix of orders. Let n be the number of tuples of considered item, then every item or itemset of size n being processed is associated with an $(n * n)$ binary matrix. For example we take the gradual item $Age \uparrow$, the binary matrix for $Age \uparrow$ will be of size $(5 * 5)$, since there are 5 tuples. The process to build binary matrix of order (for " $>$ " to represent the variation $\uparrow$) for $Age \uparrow$ is shown from Figure 2.11 to Figure 2.15. According the above mentioned principle, the value of $p1$ is checked with all the other tuples to see if there exist an order relation between them. For instance, $p1 > p2$ results as 1 and $p1 > p4$ results as 0. The resulting comparison of $p1$ w.r.t all other tuples is then placed in first column of the matrix as shown in Figure 2.11. For $p2$ the result is stored in second column and so on for all the respective tuples. The final binary matrix of order for Age $\uparrow$ is shown in Figure 2.16.

Similarly, this process of binary matrix of order is performed for all the size-1 items for both variations (i.e., " $>$ " for $\uparrow$ and " $<$ " for $\downarrow$). Therefore, for these 3 items we will have 6 binary order of matrices as Age $\uparrow$, Age $\downarrow$, Experience $\uparrow$, Experience $\downarrow$, Publications $\uparrow$, and Publications $\downarrow$. Once we have all these resultant binary order matrix, then the join operation of items takes place by performing Hadamard product (bit-wise AND operation) [1] as described below.

Id	Age	Experience	Publications
$p1$	45	13	35
$p2$	35	7	20
$p3$	30	5	25
$p4$	55	10	28
$p5$	40	14	38

Table 2.3 – *Person Data*

$$M_{Gs} = M_s \textbf{ AND } M'_s$$

Where M_s and M'_s are the matrices representing order for the itemsets s and s'. The matrix M_{Gs} is the bit-wise AND for these items. Therefore, a gradual pattern P for gradual items e.g., ($i_1 \uparrow$, $i_2 \downarrow$) can be generated, if the following relation holds [DLT09]:

$$P = (i_1 \uparrow \quad \textbf{AND} \quad i_2 \downarrow) \tag{2.8}$$

Age $\uparrow$
$p1 > p1 = 0$
$p1 > p2 = 1$
$p1 > p3 = 1$
$p1 > p4 = 0$
$p1 > p5 = 1$

$=$

	$p1$	$p2$	$p3$	$p4$	$p5$
$p1$	0	0	0	0	0
$p2$	1	0	0	0	0
$p3$	1	0	0	0	0
$p4$	0	0	0	0	0
$p5$	1	0	0	0	0

Figure 2.11 – *p1 w.r.t all other tuples*

Age $\uparrow$
$p2 > p1 = 0$
$p2 > p2 = 0$
$p2 > p3 = 1$
$p2 > p4 = 0$
$p2 > p5 = 0$

$=$

	$p1$	$p2$	$p3$	$p4$	$p5$
$p1$	0	0	0	0	0
$p2$	1	0	0	0	0
$p3$	1	1	0	0	0
$p4$	0	0	0	0	0
$p5$	1	0	0	0	0

Figure 2.12 – *p2 w.r.t all other tuples*

Age $\uparrow$
$p3 > p1 = 0$
$p3 > p2 = 0$
$p3 > p3 = 0$
$p3 > p4 = 0$
$p3 > p5 = 0$

$=$

	$p1$	$p2$	$p3$	$p4$	$p5$
$p1$	0	0	0	0	0
$p2$	1	0	0	0	0
$p3$	1	1	0	0	0
$p4$	0	0	0	0	0
$p5$	1	0	0	0	0

Figure 2.13 – *p3 w.r.t all other tuples*

Age $\uparrow$
$p4 > p1 = 0$
$p4 > p2 = 1$
$p4 > p3 = 1$
$p4 > p4 = 0$
$p4 > p5 = 1$

$=$

	$p1$	$p2$	$p3$	$p4$	$p5$
$p1$	0	0	0	1	0
$p2$	1	0	0	1	0
$p3$	1	1	0	1	0
$p4$	0	0	0	0	0
$p5$	1	0	0	1	0

Figure 2.14 – *p4 w.r.t all other tuples*

Age $\uparrow$
$p5 > p1 = 0$
$p5 > p2 = 1$
$p5 > p3 = 1$
$p5 > p4 = 0$
$p5 > p5 = 0$

$=$

	$p1$	$p2$	$p3$	$p4$	$p5$
$p1$	0	0	0	1	0
$p2$	1	0	0	1	1
$p3$	1	1	0	1	1
$p4$	0	0	0	0	0
$p5$	1	0	0	1	0

Figure 2.15 – *p5 w.r.t all other tuples*

1. Hadamarad product is a bit-wise/entry-wise product named due to the early work of the French mathematician Jaques Hadamard (1865–1963)
2. Hasse diagram, proposed by the German mathematician Helmut Hasse (1898–1979), that makes it possible to

Id	Age
$p1$	45
$p2$	35
$p3$	30
$p4$	55
$p5$	40

$=$

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Figure 2.16 – *Final Binary Matrix of Order for Gradual Item (Age ↑)*

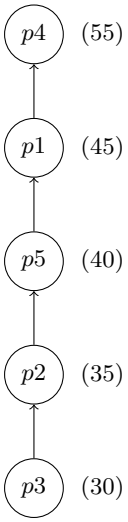

Figure 2.17 – *(Age ↑) - Hasse Diagram Representation*

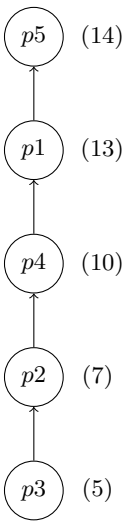

Figure 2.18 – *(Experience ↑) - Hasse Diagram Representation*

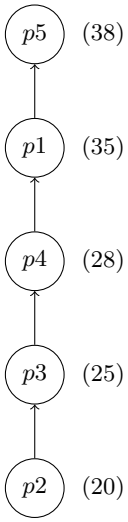

Figure 2.19 – *(Publications ↑) - Hasse Diagram Representation*

The GRITE algorithm uses precedence graph based approach to compute the support of gradual patterns so as to check whether the pattern is frequent or not. It states the for a given gradual pattern P and its associated matrix M_{Gs}, the support is the longest list from M_{Gs} [DLT09]. To illustrate the method authors use Hasse diagram [2]. The representation of items $Age \uparrow$, $Experience \uparrow$, and $Publications \uparrow$ as Hasse diagram is shown in Figure 2.17, 2.18, and 2.19 respectively.

The Hasse diagram shown in Figure 2.17, for instance when considering the pattern $Age \uparrow$, we observe that that $p3(Age30)$ precedes $p2(Age35)$ which precedes $p5(Age40)$, and so on. This results in the fact that the value 1 at the intersection between $p3$ and $p2$ in the final matrix as shown in Figure 2.16. Similarly, we can observe the Hasse diagram for Experience ↑ and Publication ↑ and the resulting values in matrices in Figure 2.20 and Figure 2.21 respectively.

When considering gradual items for more than one attribute, as for instance $(Age \uparrow, Experience \uparrow)$, the binary matrices of $(Age \uparrow)$ and $(Experience \uparrow)$ are mixed to compute the

represent a finite ordered set graphically

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 2.20 – *(Experience ↑) Binary Matrices for Gradual Item of Size-1*

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 2.21 – *(Publications ↑) Binary Matrices for Gradual Item of Size-1*

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 2.22 – *Binary AND of concordant object pairs for gradual patterns (Age↑ , Experience↑)*

resulting matrix as the Hadamard product. The resulting output is shown in the matrix representation in Figure 2.22.

The Figure 2.23 shows the precedence graph in which $p3$ (Age 30, Experience 5) precedes $p2$ (Age 35, Experience 7) because we both have $30 < 35$ and $5 < 7$ following the pattern $(Age \uparrow , Experience \uparrow)$ but $p1$ and $p4$ are incomparable as $45 < 55$ but $13 > 10$. Indeed, in the matrix the cells [p1][p4] and [p4][p1] are set to 0 which means that there is no order relation between

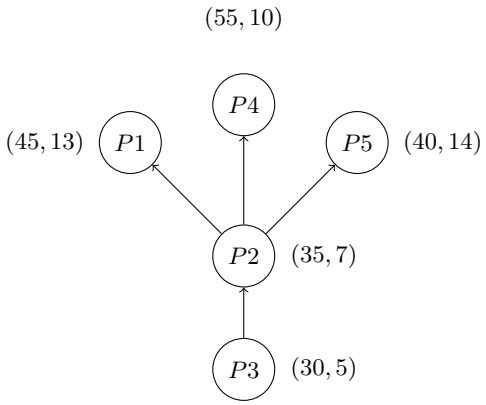

Figure 2.23 – *Precedence Graph (Age ↑ , Experience ↑) - Hasse Diagram representation*

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

(Age ↓)

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

(Publications ↓)

Figure 2.24 – *Binary matrix representing orders for the dataset shown in Table 2.3*

$p1$ and $p4$. This mapping of precedence is achieved using bitwise AND operation of itemsets ($Age \uparrow$ and $Experience \uparrow$). *The support of the considered itemset can then be obtained as the length of the maximal path in the graph* [LLR09]. The algorithm uses a memory conserving knowledge for information about the traversal of node from Hasse diagram for keep the knowledge about its maximality. In the final result the solution is considered with the longest path.

In order to enhance the performance of algorithm [DLT09], a scalable implementation is presented in [Do+15] in which parallelism is exploited by employing multi-core processors for mining gradual patterns.

2.4.7 GRAANK Algorithm

The GRAdual rANKing (GRAANK) algorithm presented in [LLR09] combines the principals of existing approaches to exploits the rank correlation for gradual pattern extraction. More precisely, it considers the framework of Definition 2.5 that evaluates gradual tendency in terms of ranking correlation and propose the algorithm for support computation [LLR09]. It uses Kendall's tau rank correlation method of concordant and discordant pairs in the process of mining gradual pattern.

Kendall's tau ranking correlation coefficient is one of the most widely used non-parametric measure of association between two variables for rank correlations [PL84]. It evaluates the strength of dependence between two variables. *The Kendall's rank correlation coefficient evaluates the degree of similarity between two sets of ranks given to a same set of objects. This coefficient depends upon the number of inversions of pairs of objects which would be needed to transform one rank order into the other* [Abd07]. The pairs that can be ordered in the same way are called *concordant* and those ordered differently are called *discordant*. The Kendall tau rank correlation is calculated as follow:

$$\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)} \tag{2.9}$$

$$\begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{array} \begin{array}{c} \begin{array}{ccccc} p1 & p2 & p3 & p4 & p5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 2.25 – *Binary AND of concordant object pairs for gradual patterns (Age↓ , Publications↓)*

Where for given n objects to be ranked, n_c is list of concordant pairs and n_d is list of discordant pairs fro the given ranking order.

As stated in [LLR09]. *In order to take into account information regarding the sense of variation of the attributes, i.e. to distinguish between $A \leq$ and $A \geq$, we consider object couples instead of pairs, dissociating the two cases of concordance defined in the Kendall's tau: we keep the information whether the couple (i, j) or the couple (j, i) is concordant. The support then equals the length of the concordant couple list, divided by the total number of object pairs.* For example, considering the sample data Table 2.3 shows the list of concordant pairs for gradual items (Experience ↑, Publications ↑) and (Age ↓, Experience ↓).

$$(Experience \uparrow,\ Publications \uparrow) = (5, 25), (10, 28), (13, 35), (14, 38)$$

$$(Age \downarrow,\ Experience \downarrow) = (55, 10), (35, 7), (30, 5)$$

This algorithm uses binary matrices for representing orders and performs the computation of binary matrices representing the set of concordant object pairs for instance for (Age↑ **AND** Experience↑) the result is shown in Figure 2.22 and for (Age↓ **AND** Publications↓) the result is shown in Figure 2.25. Then for all candidate itemsets, the support is computed as the sum of their concordance matrices divided by $n(n-1)/2$ where n is the number of objects.

2.5 Fuzzy Gradual Patterns

Gradual patterns allow to express patterns in the form *the higher the age, the higher the experience* [Ayo+10]. However, they may lead to crisp behaviours because as seen in section 2.4, gradual patterns allow to deal with numerical data. We thus investigate fuzzy pattern mining and its extension to gradual patterns aka *fuzzy gradual pattern mining*. To do so, we first describe the basics of fuzzy sets and membership functions.

2.5.1 Fuzzy Logic and Fuzzy Sets

"Everything is vague to a degree you do not realize till you have tried to make it precise."

– Bertrand Russell

"Fuzzy logic is not fuzzy. Basically, fuzzy logic is a precise logic of imprecision and approximate reasoning" - Lotfi A.Zadeh [Zad08].

Humans have the capability to converse, reason, and take decision in an environment of imprecise information. Fuzzy logic can be seen as formalization of these human capabilities [Zad08].

Fuzzy set theory was first introduced by Lotfi A. Zadeh in 1965 [Zad65] to allow to deal with natural language and imprecise concepts. The theory of fuzzy sets is central to fuzzy logic [Dub80, PG98]. Fuzzy sets help to represent the uncertain or imprecise information by using linguistic variables such as very low, very high, moderate, very young, almost young, very old, almost old etc., With fuzzy sets we can have non-sharp boundaries (e.g., low or high) for an attribute which in turn help to provide a reasonable representation of different cognitive concepts and linguistic expressions [Hül02]. A fuzzy set is described by its membership function.

Membership Functions

A fuzzy subset is identified by a so-called *memberships function* which is a generalization of characteristic function that specifies the degree of membership of x in the fuzzy set [Hül02]. *Usually, membership degrees are taken from the unit interval [0, 1], i.e. a membership function is a mapping of [0, 1]* [Hül02]. The membership function formally defined in [Dub80] as follows.

Definition 2.6 (Membership Function)**.** Let X be a universal set (aka universe of discourse) containing all possible objects whose elements are denoted as x, the membership in a fuzzy subset A of X is often viewed as a characteristic function μ_A from X to [0,1] such that $\mu_A(x) = 1$ if x fully belongs to A and $\mu_A(x) = 1$ if x does not belong at all to A.

It states that each element of X is mapped to a value between 0 and 1 and that value is called as *memberships value* or *degree of membership*. The degree of membership quantifies the grade of membership of that element in X to the fuzzy subset A.

Common Types Membership Functions

The usefulness of fuzzy sets very much depends on the construction of membership function. Various methods for construction of memberships functions are addressed in literature [Str15]. The membership function allows us to represent a fuzzy set graphically. Figure 2.26 shows the most common membership function.

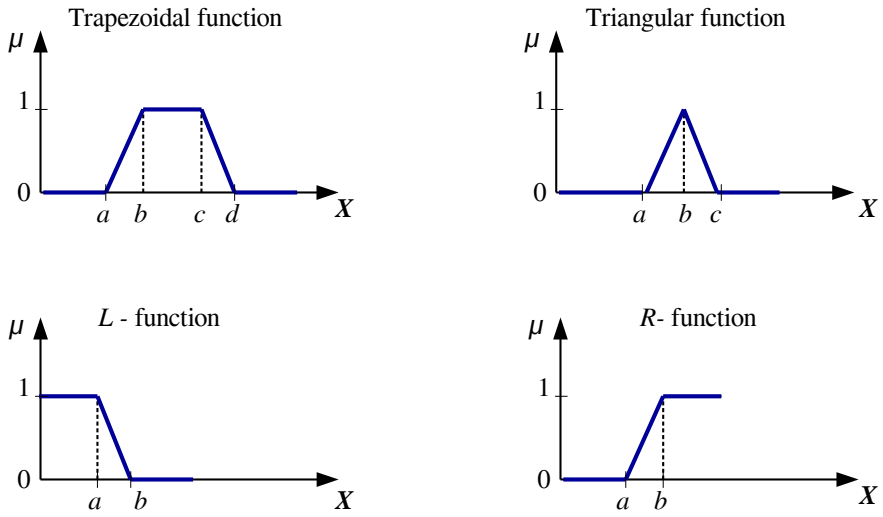

Figure 2.26 – *Membership Functions [Str15]*

2.5.2 Defining Fuzzy Gradual Patterns

Since the inception of fuzzy sets [Zad65], considerable literature has be contributed in variety of disciplines with its applications from medicine to consumer products [Bez+99]. A comprehensive work on fuzzy models for pattern recognition is presented in [Bez+99]. *Pattern recognition* is defined as a field about feature analysis, clustering and classifier design.

The applications of fuzzy systems have been seen in the field of control theory, computational intelligence and artificial intelligence [Bez+99, JB09]. Research is conducted to investigate applications of fuzzy sets in the fields of statistical data analysis [CGK06], machine learning and data mining [Hül11] and more recently in the big data [Fer+16] and data science [Bou18].

The authors in [QLP11] present an approach for frequent gradual patterns in terms of rank correlation measure on the basis of fuzzy orderings. [Ayo+10] presents a method to automatically build fuzzy modalities (such as, *almost x*) from numerical dataset to mine fuzzy gradual pattern. The approach allows to extract the knowledge (gradual patterns) from numerical datasets that may not be discovered by crisp approach. As of now, there is no work presented

in the literature to extract such kind of fuzzy gradual patterns from semi-structured graph databases that involves two tasks i.e., to first deal with missing data and then extract patterns.

Fuzzy gradual patterns are of the form *"the more/less A_1 is F_1 , the more/less A_k is F_k"* [Ayo+10, KH10, Bou+10]. For instance, *the more the age is "almost 40", the more the experience is "almost high"*. These patterns express the information about the attributes and their co-variations which can leads to valuable knowledge for experts for decision making.

In contrast with the crisp data for example "Age" and "Experience " attributes as shown in Table 2.3, the fuzzy data allows linguistic variables associated to fuzzy modalities such as (e.g. "almost", "all the more", "low", "normal", "high" etc.,). For instance, "Age" can be associated to 3 modalities such as "low", "normal", and "high". *The data are then described by membership degrees that indicate the extent to which their characteristics belong to the considered modalities* [Bou+10]. *A "fuzzy gradual item" is then a "triplet" made of an attribute, one of its modalities and a variation, such as (low, normal, high)* [LLR09]. As a result, fuzzy gradual pattern may allow to discover some interesting patterns that would not have been retrieved by existing methods because of the crisp approaches such as presented in [DLT09].

Definition 2.7 (Fuzzy Gradual Item)**.** Given an attribute X defined on a universe U, a fuzzy subset A defined on U describing one of X modalities and $\star \in \{\uparrow, \downarrow\}$, a *fuzzy gradual item* is defined as the triplet $(X, A, \star)$.

Definition 2.8 (Fuzzy Gradual Itemset)**.** A *fuzzy gradual itemset* is defined as a combination of several gradual items, semantically interpreted as their conjunction.

Example 2. Fuzzy gradual items FP can be written as, $FP = \{(X_1, A_1, >),\ (X_2, A_2, <)\}$ and interpreted as *" The more the X_1 is A_1, the less X_2 is A_2"*, as for instance *the more the age is "almost 40", the more the experience is "almost high"*.

The issue arises in defining that "what would be the best fuzzy modality". [Ayo+10] presents a method to build fuzzy modalities automatically.

[QLP11, Sic+13] present methods to mine fuzzy gradual patterns based on multi-core architectures and fuzzy ordering. Fuzzy orderings refer to the fact that two values can *more or less* different. For instance, 0.1 may be considered as very different from 0.8 while 0.000001 may be considered as not that different from 0.0000011. In real-world world database applications, it is sometimes hardly possible to rank the data according to graduality. Therefore, fuzzy ordering and fuzzy ranking techniques presented in the literature help us to address vagueness, ambiguity and imprecision occuring in some situations [Sic+13].

To the best of our knowledge as of today, no work has been presented to deal with the problem of mining fuzzy gradual patterns when the data is in graph. In our work, we thus define fuzzy property graph gradual items and patterns and present the mechanism to retrieve these

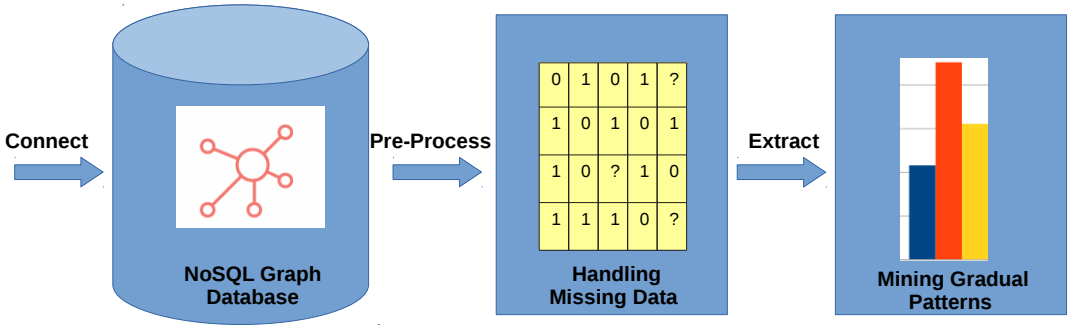

Figure 2.27 – *Property Graphs Missing Data Handling and Pattern Mining Process*

patterns from property graphs. However, we rely on former work for dealing with fuzziness within property graphs. [CL14] has introduced fuzzy queries on Neo4j by presenting an extension of the *Cypher* declarative query *aka. cypherf*. For instance, it can be applied to retrieve *cheep* and *popular* hotels from a graph database.

In our work, it may be the case that source properties are missing. We thus review in the next section the main existing methods for dealing with missing data.

2.6 Handling Missing Values

In order to extract gradual patterns from property graphs, often we face the situation when all the properties of a node may not be present because they do not have a predefined schema. In property graph, a node may not contain all the properties, as the data may be missing since it may not be available, or it does not exist as it may not be applicable (for example no salary, or experience in the case of kids). Therefore, it requires to treat missing data and then apply mining algorithms to retrieve patterns of interest from graphs.

In our proposed approach, we investigate a mechanism to handle semi-structured data for mining gradual pattern. The main steps of our approach are; (i) Retrieve the graph data, (ii) Treat missing data and (iii) Extract gradual patterns. An overall process flow depicting these three steps for gradual pattern mining from a property graph is shown in Figure 2.27.

When dealing with missing data, it is important to consider the type or nature of missing data. For example the missing data can be random, structured, forgotten or sometimes not available etc. Types of missing data were first described in [RUB76] as; (i) Missing Completely At Random (MCAR), (ii) Missing At Random (MAR), and (iii) Missing Not At Random (MNAR).

In MCAR, the missing data is not related to a specific value or observed response. When

missing data depends on set of observed responses, but not related to specific values it is called MAR. The missing values are distributed randomly across all the data in MCAR whereas, in MAR missing values are distributed within one or more sub-samples. If the missing data is neither MCAR nor MAR, it is then categorized as MNAR. The missing values in MNAR are related to actual values and they do depend on missing values. The example for MNAR can be the people with high income are less likely to report their income [RUB76].

2.6.1 Handling Missing Data Techniques

Primarily there are two main approaches to deal with missing values i.e., (i) ignoring / deletion and (ii) imputation. It is important to decide the best strategy that results the least biased results. This can be performed by evaluation of the source data and identifying type of missing data. A classification of handling missing data give in [Swa18] is shown in Fig.2.28.

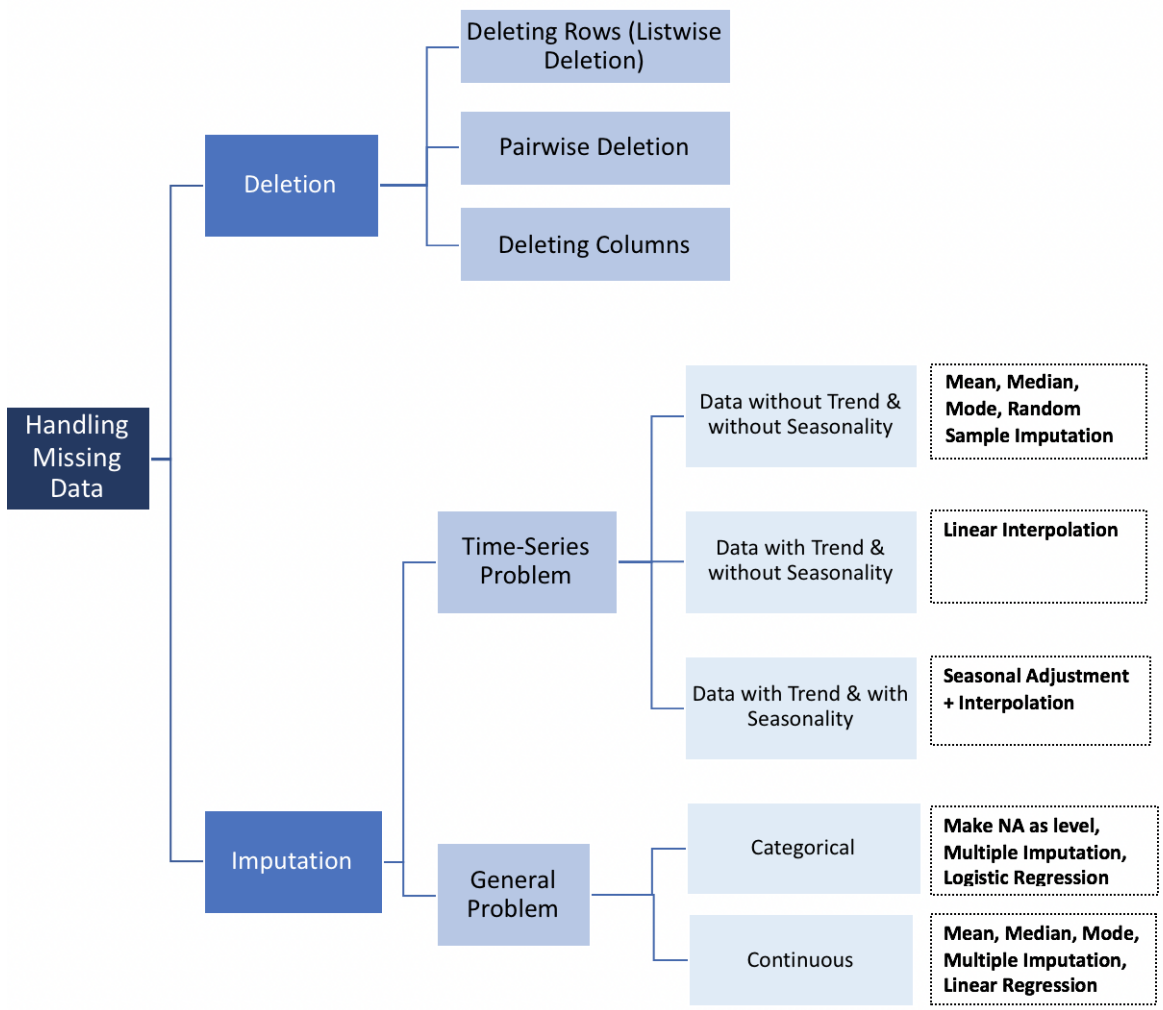

Figure 2.28 – *Handling Missing Data [Swa18]*

Ignoring or deleting techniques include listwise deletion (i.e., ignoring a row/ object which contains missing data) and attribute deletion. Imputation techniques include replacing missing data with fixed constant, replacing with mean of the objects of attribute and multiple imputation method. Expectation Maximization presented in [DLR77] is a missing data imputation technique in which missing values are estimated (parameter estimation) based on the maximum likelihood method.

2.6.2 Handling Missing Data With Replacement

Imputation methods involve replacing missing data. However, we claim that we can not replace missing data in case of property graphs. Indeed, if the property is missing in a node, there is often no meaning to add it artificially, which creates a bias in results, particularly in the case of real datasets.

The treatment of missing data using imputation is studied by employing supervised learning algorithms [BM03, HHE03, SRA17]. In [WWC04, Ben+09], an association rules based model is used to complete missing data.

The use of graphical models for processing of missing data is presented in [MP18]. "Graphical models enable an efficient and transparent classification of the missingness" [MP18] i.e., whether it is MCAR or MAR. Maximum Likelihood and Multiple Imputation are well known missing data estimation methods but they assume MAR. Consistent estimation for missing data and its implementation procedure is discussed theoretically for MAR and MNAR categories.

2.6.3 Handling Missing Data Without Replacement

[PL84] has considered the question of dealing with missing data for the computation of rank correlation. However, the paper only considers the case where one or several individuals are missing, meaning, in the tabular representation, a full line whereas we consider here the case where single values are missing.

As mentioned in [OW72], we claim that *"obviously the best way to treat missing data is not to have them"*. In real world applications, where data is managed in relational databases, we often find missing values. [RC98] has proposed to mine association rules in relational databases in the presence of missing data by segmenting the database into several so-called *valid databases* (*"vdb"*), in such a way that a vdb does not contain any missing value. The authors have hence redefined the concepts of support and confidence for vdb that are fully compatible with existing association rule mining algorithms.

The authors in [RC98] redefine the classical *support* and *confidence* for *vdb*. Figure 2.29 shows two datasets with and without missing values.

Id	X1	X2	X3	X4
1	a	a	a	c
2	a	a	b	c
3	a	b	c	c
4	a	b	d	c
5	b	b	e	d
6	b	b	f	d
7	b	c	g	d
8	b	c	h	d

Database 1 (DB1)

Id	X1	X2	X3	X4
1	?	a	a	c
2	a	a	b	?
3	a	b	c	c
4	a	b	d	c
5	?	b	e	d
6	b	b	f	?
7	b	c	g	d
8	b	c	h	d

Database 2 (DB2)

Figure 2.29 – *Datasets with and without Missing values*

We recall the classical definition of support given in Equation 2.1 and confidence in Equation 2.4, if the required *minimum support threshold* is 40%, and the confidence is 80% for DB1 shown in Figure 2.29, the itemsets result is given as below.

$$\begin{aligned} &Length\ 1: \quad \{X1 = a\}, \{X1 = b\}, \{X2 = b\}, \{X4 = c\}, \{X4 = c\}; \quad supp = 50 \\ &Length\ 2: \quad \{X1 = a,\ X4 = c\}, \{X1 = b,\ X4 = c\}; \quad supp = 50 \end{aligned}$$

The resulting association rules will be:

$$\begin{aligned} X1 = a &\rightarrow X4 = c, \quad supp = 50, \quad conf = 100 \\ X1 = b &\rightarrow X4 = d, \quad supp = 50, \quad conf = 100 \\ X4 = c &\rightarrow X1 = a, \quad supp = 50, \quad conf = 100 \\ X4 = d &\rightarrow X1 = b, \quad supp = 50, \quad conf = 100b \end{aligned}$$

Now, taking the same required *minimum support threshold* is 40%, and the confidence is 80% for DB2, the results are as under:

$$\begin{aligned} Length\ 1: \quad \{X1 = a\} &= \frac{3}{8} = 0.375 \\ \{X1 = b\} &= \frac{3}{8} = 0.375 \\ \{X2 = b\} &= \frac{4}{8} = 0.5 \\ \{X4 = c\} &= \frac{3}{8} = 0.375 \\ \{X4 = d\} &= \frac{3}{8} = 0.375 \end{aligned}$$

As we can see in the above result that none of the $Length\ 1$ has met the *minimum support threshold requirement* of 40%, except $\{X2 = b\}$ which has support of 50%. As as result there will be no $Length\ 2$ itemsets will be formed and hence no rules. Now, we see the how this missing data results when we take *vdb*. The *vdb* approach stated that we can avoid the above mentioned problem by *partially ignoring missing values*, as a fact only certain data containing missing values will be partially disabled. To explain the *vdb*, we recall the definitions given in [RC98] as below.

Let D be a database, X an itemset, and t a data.

Definition 2.9 (Disabled data)**.** t is disabled for X in D, if t contains missing values for at least one item i of X.

We note that $Dis(X)$ is the subset of D disabled for X.

We note that D_X is the subset of D containing X, i.e.

$$D_X = \{t \in D \mid X \subseteq\ t\}$$

Definition 2.10 (Valid database)**.** The valid database vdb for an itemset X in D is:

$$vdb(X)\ =\ D\ -\ Dis(X)$$

Consequently, the *support* is redefined as follows:

$$Support(X) = \frac{|D_X|}{|vdb(X)|} = \frac{|D_x|}{|D| - |Dis(X)|} \tag{2.10}$$

With the new definition of support, now we are taking the same required *minimum support threshold* of 40% for DB2, and the results are as under:

$$\begin{aligned}
Length\ 1: \quad \{X1 = a\} &= \frac{3}{6}\ =\ 0.5; \quad (data\ disabled\ =\ 1, 5)\\
\{X1 = b\} &= \frac{3}{6}\ =\ 0.5; \quad (data\ disabled\ =\ 1, 5)\\
\{X2 = b\} &= \frac{4}{8}\ =\ 0.5\\
\{X4 = c\} &= \frac{3}{6}\ =\ 0.5; \quad (data\ disabled\ =\ 2, 6)\\
\{X4 = d\} &= \frac{3}{6}\ =\ 0.5; \quad (data\ disabled\ =\ 2, 6)\\
Length\ 2: \{X1 = a,\ X4 = c\} &= \frac{2}{4}\ =\ 0.5; \quad (data\ disabled\ =\ 1, 2, 5, 6)\\
\{X1 = b,\ X4 = d\} &= \frac{2}{4}\ =\ 0.5; \quad (data\ disabled\ =\ 1, 2, 5, 6)
\end{aligned}$$

When dealing with missing values, one of the main issue is to select *minimum support threshold*, because if the value is small, too many itemsets will be generated and hence the association rules and if the value is large, very few or no itemsets may be generated. So in data mining task, it is generally considered as a hard problem to fix these minimum values.

2.7 Conclusion

In this chapter, we explained the basic concepts to form the basis for our proposed method. The topic covered in this chapter include graph databases their applications and importance in dealing with large unstructured data. We discussed Neo4j graph database which is used in our proposed method to create and manage property graphs. We also discussed the basic concepts and definitions related to frequent pattern mining and gradual pattern mining. We discussed two main algorithms for gradual pattern mining. Furthermore, we have briefly described fuzzy sets and fuzzy gradual patterns. Finally, we discussed about various missing data handling techniques to form a basis for dealing with missing data issue related to property graphs when it is required to extract gradual patterns.

Chapter 3

Defining Gradual Patterns from Property Graphs

3.1 Introduction

In this chapter, we introduce different approaches in which gradual patterns can be defined from property graphs. The definition of these patterns rely on both nodes and relationships and their properties, which provides several opportunities to exploit the information. The patterns are extended to fuzzy gradual patterns. In our work, we have designed the following types of gradual patterns from property graphs:

- Intra-Label-Node-Properties,
- Inter-Label-Node-Properties,
- Node-Properties-with-Relationships-count,
- Node-Properties-with - Relationships -Properties,
- Inter-Relationships -Properties,
- Fuzzy Gradual Patterns from Property Graphs.

All these scenarios provide various perspectives of the data depending upon the application domain and requirements of the end user. The first two types of patterns rely on node properties while the other three ones also exploit relationships. In the rest of this chapter, we detail and illustrate these different types of gradual pattern mining by providing relevant definitions and examples.

3.2 Definitions

Property graphs have been studied in commercial solutions and the literature for the last ten years. However, existing definitions do not provide satisfactory frameworks including all the specificities of such data: properties put to both nodes and relationships, types put on relationships (only one type, only one direction, several relationships having to be defined for bi-directional relationships and/or for several types), several possible labels for every node, etc.

An overall nodes and relationships visualization of a property graph is shown in Figure 2.7 and the details of the nodes are shown in 2.1 for label "Person", Listing 2.2 for label "Paper" Listing 2.3 for label "Project" respectively. We define property node, property relationship, and property graph as follows.

Definition 3.1 (Property Node)**.** Let Λ_N be a set of node Labels with $NULL \in \Lambda_N$, Π_N be a set of properties where every property $\pi \in \Pi_N$ can take values over a domain $dom(\pi)$. P_N is

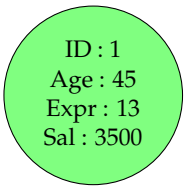

Figure 3.1 – *Property Node*

Figure 3.2 – *Property Relationship between two Property Nodes*

the set of all possible pairs (π, v) with $v \in dom(\pi)$. A property node n is given by the tuple (id_n, Λ_n, P_n) with

- id_n the unique identifier of n,
- $\Lambda_n \subseteq \Lambda_N$ the set of labels defining the node,
- and $P_n \subseteq P_N$ is the set of properties of n.

Example 3. Λ_N ={Person, Professor, NULL}, Π_N = {Age, Expr, Sal}, n_1 = (1, { Person, Professor}, {(Age : 45), (Expr : 13), (Sal : 3500)}) is a property node as shown in Figure 3.1.

Definition 3.2 (Property Relationship)**.** Let N be a set of property nodes, Λ_R be a set of relationship types with $NULL \in \Lambda_R$, Π_R be a set of properties where every property $\pi \in \Pi_R$ can take values over a domain $dom(\pi)$. P_R is the set of all possible pairs (π, v) with $v \in dom(\pi)$. A property oriented relation r is given by the tuple $[id_r, n_1, n_2, \lambda_r, P_r]$ with

- id_r the unique identifier of r,
- $n_1 \in N$,
- $n_2 \in N$,
- $\lambda_r \in \Lambda_R$ the Type of the relation,
- and $P_r \subseteq P_R$ is the set of properties of r.

Example 4. Λ_R = { PUBLISH, RCV_GRANT, COLAB. }, Π_R = {date, since, NULL},and r_1 = [28, {1}, {15}, {PUBLISH}, {(date:2016-08-10)}] is a property relation as shown in Figure 3.2.

```
neo4j> MATCH (n:Person) -[r:PUBLISH]->(p:Paper )
       WHERE ID(n)=1 AND ID(p)=15
       RETURN ID(r),ID(n),ID(p),type(r), r.date;
+-------------------------------------------------------------+
| ID(r) | ID(n) | ID(p) | type(r)   | r.date                  |
+-------------------------------------------------------------+
| 28    | 1     | 15    | "PUBLISH" | 2016-08-10T00:00Z |
+-------------------------------------------------------------+

1 row available after 18 ms, consumed after another 0 ms
neo4j>
```

Listing 3.1 – *Property Relationship*

Listing 3.1 shows the cypher-shell query to retrieve the record for *"date"* property for relationship type "PUBLISH" using node IDs given in Example 4

Definition 3.3 (Property Graph)**.** Let Λ_N be a set of node Labels with $NULL \in \Lambda_N$, Π_N be a set of properties of node, P_N a set of (key:value) pairs over Π_N, Λ_R be a set of relationship Types with $NULL \in \Lambda_R$, Π_R be a set of properties of relationship, P_R a set of (key:value) pairs over Π_R.

A property graph G is given by (N, R) where:

- N stands for a set of property nodes defined over Λ_N and P_N,
- R stands for a set of property relationships defined over N, Λ_R and P_R.

Example 5. The details of property graph PG as shown in "Fig 3.3" are : Λ_N = ({(Person:Professor), (Paper), (Project) }, Λ_R = ({ COLLAB, PUBLISH , RCV_GRANT }, Π_N = {(ID, Name,Age, Desig, Expr, Sal, NULL), (ID, Pr_Title, Type, Year, Citations, Impact_Factor, NULL), (ID, Year, Amount, NULL)}, Π_R = {(date, NULL), (since, NULL)}.

Definition 3.4 (Property Graph Node Gradual Item)**.** Let Λ_N be a set of node Labels with $NULL \in \Lambda_N$, Π_N be a set of properties, P_N a set of (key:value) pairs over Π_N, Λ_R be a set of relationship types with $NULL \in \Lambda_R$, Π_R be a set of properties, P_R a set of (key:value) pairs over Π_R.

Let $G_D = (N, R)$ be a graph defined over Λ_N, P_N, Λ_R and P_R. A graph data gradual item is a gradual item $i\star$ where $i \in \Pi_N \cup \Pi_R$.

Example 6. An example of such property graph data gradual item could be *"The higher the Age"* $(Age \uparrow)$, where "Age" is a property of a node labeled with "Person".

Definition 3.5 (Graph Relationship Gradual Item)**.** Let Λ_N be a set of node Labels with $NULL \in \Lambda_N$, Π_N be a set of properties, P_N a set of (key:value) pairs over Π_N, Λ_R be a set of relationship Types with $NULL \in \Lambda_R$, Π_R be a set of properties, P_R a set of (key:value) pairs over Π_R.

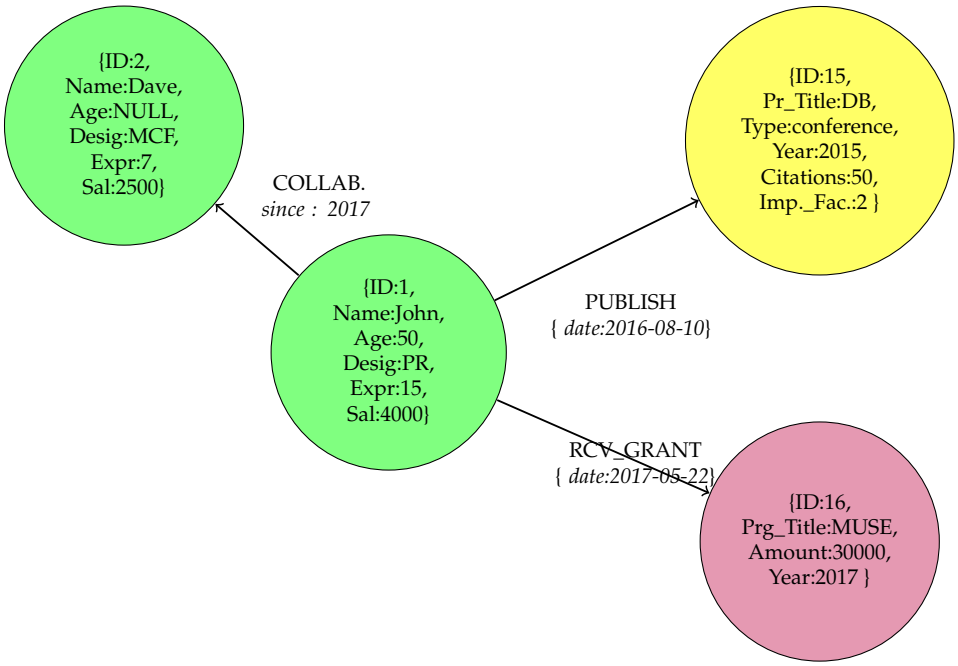

Figure 3.3 – *Property Graph*

Let $G_D = (N, R)$ be a graph defined over Λ_N, P_N, Λ_R and P_R. A graph relationship gradual item is a gradual item $i^j\star$ where $j \in \mathbb{N}$ is the depth of the item, $i \in \Lambda_R$ and $\star \in \{\uparrow, \downarrow\}$.

For the sake of simplicity, when a property graph relationship gradual item is of depth 1, we can omit to mention the depth, as shown in Example 7.

Example 7. (COLLABORATE $\uparrow$) is equivalent to (COLLABORATE $^1\uparrow$) and stands for *the number of collaborations increases*. This considers that the relatioship depth is 1 i.e., a relationship path between two nodes. (COLAB.$^3\uparrow$) stands for *number of collaboration of collaboration of collaboration increase. This involves a path traversal of depth 3.*

The path traversal at the depth of 3 relationships in Neo4j can be extracted using the cypher query shown in Listing 3.2

```
1
2 neo4j> MATCH path=(:Person) -[ :COLLABORATE* ] -> (e:Person)
3         USING SCAN e:Person
4         RETURN EXTRACT (n in NODES(path)| ID(n)) AS path_Nodes,
5         length(path) AS Length
6         ORDER BY Length DESC;
```

Listing 3.2 – *Path Traversal*

The graph topology structure that is retrieved from property graphs in order to deal with the data in a more efficient way. For this purpose, we consider graph structure summaries as

described below.

Definition 3.6 (Property Graph Structure Summary)**.** Let SN_L be a source node label with $NULL \in SN_L$, DN_L be a destination node label with $NULL \in SN_L$, R_T be a unique of relationship type between SN_L and DN_L, and C be the count of relationships between SN_L and DN_L.

A property graph structure summary G_{SS} is given by a matrix of tuple (SN_L, R_T, DN_L, C) representing the strucure of graph.

Example 8. (SN_L = (Person) , R_T =[COLLABORATE] , DN_L = ((Person), C=15) or (SN_L = (Person) , R_T =[PUBLISH] , DN_L = ((Paper), C=15).

Now, as described in Section 1.2, our two main objectives are; (i) Defining gradual patterns in the context of property graphs and (ii) Mining gradual patterns automatically from property graphs. We have defined basic definitions as described above and we start different types of patterns that can be extracted from property graphs.

3.3 Types of Gradual Patterns in Property Graphs

In this section we describe the five different types of gradual patterns in the context of property graphs as mentioned in Section 3.1. These scenarios describe the possibilities in which gradual patterns can be extracted from property graphs depending upon the requirement of end user.

3.3.1 Intra-Label-Node-Properties

We investigate the scenario of *Intra-Label-Node-Properties* for gradual pattern extraction. In this scenario, the pattern extraction process is to be performed for nodes sharing the *"same label"*. It is important to note that in this scenario, there are no relationships involved.

First, the program retrieves the data from property graph using for each label. This steps involves the connection of the program with graph database and retrieval of data for all nodes for each labels. Listing 3.3 shows the cypher-shell query to show a representation of query and retrieved data. For example, considering label "Person" to retrieve its data from property graph. Listing 3.4 shows the cypher query and retrieved data in cypher-shell for label person.

At this stage, the program transforms the output of cypher query response received from graph database is into tabular representation so as to as to be able to apply pattern mining

algorithms. The property graph for label person with (key:vales) is shown in Figure 3.4 and its transformation in tabular representation is shown in Table 3.1.

```
neo4j> MATCH (n)
RETURN labels(n) as LABEL, count(*) as nodeCount;
+--------------------------+
| LABEL      | nodeCount   |
+--------------------------+
| "Person"   | 7           |
| "Paper"    | 5           |
| "Project"  | 2           |
+--------------------------+

3 rows available after 2 ms, consumed after another 0 ms
```

Listing 3.3 – *Retrieve Node Labels with Node Count*

```
neo4j> MATCH (n:Person)
       RETURN n AS Person_Label_Nodes ;
+-------------------------------------------------------------------------------------------+
| Person_Label_Nodes                                                                        |
+-------------------------------------------------------------------------------------------+
| (:Person {desig: "PR", name: "John", expr: 13, age: 45, sal: 3500})                       |
| (:Person {desig: "MCF", name: "Dave", expr: 7, sal: 2500})                                |
| (:Person {desig: "ETU", name: "Emily", expr: 5, age: 30})                                 |
| (:Person {desig: "MCF", name: "Tim", expr: 10, sal: 2800})                                |
| (:Person {desig: "PR", name: "Carl", expr: 14, age: 44, sal: 3800})                       |
| (:Person {desig: "MCF", name: "Chris", expr: 9, age: 38, sal: 2600})                      |
| (:Person {desig: "ETU", name: "David", expr: 5, age: 32, sal: 1400})                      |
+-------------------------------------------------------------------------------------------+

7 rows available after 8 ms, consumed after another 0 ms
```

Listing 3.4 – *Label Person Data*

To check gradual patterns for label "Person", the algorithm finds the correlation among the properties of nodes. The output pattern according to definition 2.4 may be of the form *"the higher the age, the higher the experience"*. To do so, the program creates binary order matrix for given properties as discussed in Section 2.4.

We compute the support as described in algorithm presented in [LLR09] for gradual pattern extraction. But, it requires to handle missing values first, because the exiting algorithm does not allow to treat missing data. In section 4.1, we present the details for mining this type of pattern.

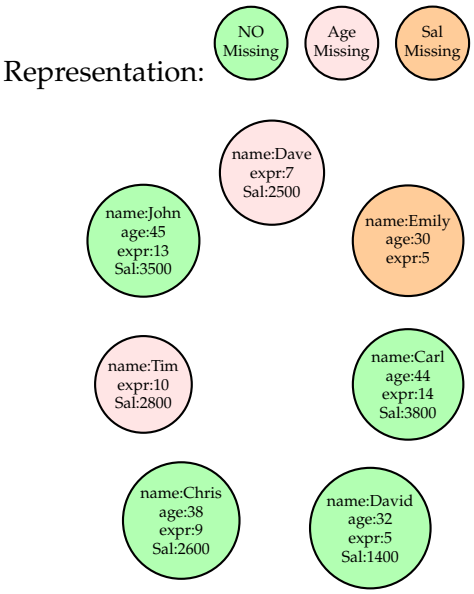

Figure 3.4 – *Label "Person" Nodes*

Id	Name	Age	Expr	Sal
1	John	45	13	3500
2	Dave	?	7	2500
3	Emily	30	5	?
4	Tim	?	10	2800
5	Carl	44	14	3800
6	Cris	38	9	2600
7	David	32	5	1400

Table 3.1 – *Label "Person" Tabular form*

3.3.2 Inter-Node-Label-Properties

In the *Inter-Node-Label-Properties* scenario, gradual patterns are extracted from the nodes having "*different labels*" that may or may not involve a relationship between them. From our running example graph schema Figure 2.7, we take two node labels i.e., "Person" and "Paper" to explain the scenario. In this situation, they involve a relationship "PUBLISH" between these two types of nodes but this is not mandatory. A query to display the node properties of label "Paper", the cypher query is shown in Listing 3.5.

```
neo4j> MATCH (n:Paper)
       RETURN Id(n), n.pr_title,n.year, n.Citations, n.Impact_Factor;
+-------------------------------------------------------------------+
| Id(n) | n.pr_title | n.year | n.Citations | n.Impact_Factor |
+-------------------------------------------------------------------+
| 11    | "GRAPH"    | "2015" | 10          | NULL            |
| 12    | "LOGIC"    | "2016" | 30          | 4.5             |
| 13    | "FUZZY"    | "2015" | 5           | NULL            |
| 14    | "RDF"      | "2017" | NULL        | NULL            |
| 15    | "DB"       | "2016" | 50          | 2               |
+-------------------------------------------------------------------+

5 rows available after 7 ms, consumed after another 0 ms
neo4j>
```

Listing 3.5 – *Query Node Properties for Label "Paper"*

```
1 MATCH (n:Person)-[r:PUBLISH]-> (m:Paper)
2 RETURN n,r,m
```

Listing 3.6 – *Label "Paper" and "Person" Nodes with Relationships*

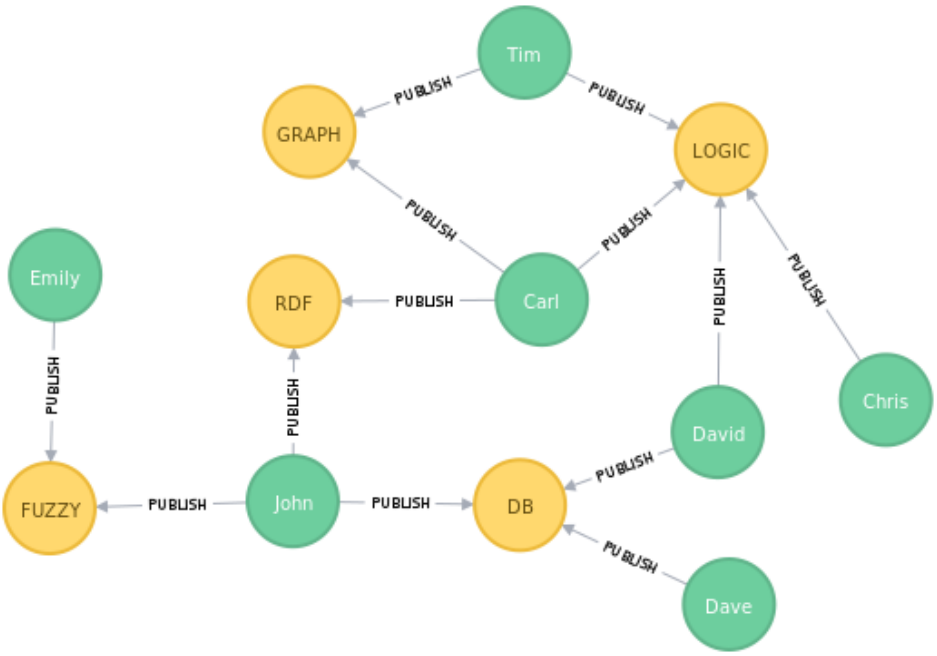

Figure 3.5 – *Label "Person" Nodes with Relationships Graph Visualisation*

As we can observe in the output result of Listing 3.5, we have missing data in Citations and Impact_Factor properties represented as "NULL". Therefore, it requires to deal with missing data before mining the gradual patterns. Also, we know that some nodes in label "Person" have missing values as shown in 3.1. After dealing with the missing data and *vdb* computation as explained in Section 3.3.1, the pattern mining algorithm is applied by the program. To find the gradual patterns in this scenario, we need to address both nodes with labels and the relationships types between them. To show graph visualization for nodes and corresponding relationship of Person and Paper, a *cypher* query is given in Listing 3.6. The resulting output is shown in Figure 3.5.

Once the data is retrieved from the graph and missing values between the labels are addressed then then mining process tries to find correlation of the type *"the higher the experience, the higher the paper citation"* and other that might exits in the data. This done by checking the patterns which have support greater than the user defined minimum support threshold. The algorithm for this mining task are discussed in Section 4.3.

3.3.3 Node-Properties with Relationships Count

In *"Node-Properties-with-Relationships- count"* scenario, we try to extract the gradual patterns that involve node properties for a label as well as the corresponding relationships count. The pattern extraction process will be processed to see if there exists any correlation between them. In this case, the correlation can be *"the higher the age, the higher the number of collaborations"* or *"the higher the experience, the higher the number of punlications"*.

To perform this type of pattern mining we first need to extract graph summaries describing relationships. *"Graph summarization facilitates the identification of structure and meaning in data"* [Liu+18]. There are various summarization techniques depending upon the nature and requirement of the task [KBB17, Liu+18]. These graph summarization techniques include aggregation-based (topology), attribute-based (topology and attributes) [Wu+14], application-oriented (e.g., graph pattern matching), and domain specific (e.g., bioinformatics) [KBB17]. A mechanism to retrieve structure summaries from NoSQL graph engine is also presented in [CL17]. Listing 3.7, shows the relationships details.

In our scenario, our program first extracts the graph topology summary including the labels and relationships count and further uses that for mining gradual patterns. The output of graph schema summary for running example is shown in Listing 2.6.

```
neo4j> MATCH ()-[r]->()
       RETURN type(r) as Relationship_Type, count(*) as Relationships_Count;
+-------------------------------------------------+
| Relationship_Type | Relationships_Count |
+-------------------------------------------------+
| "RCV_GR"          | 3                   |
| "PUBLISH"         | 13                  |
| "COLLABORATE"     | 14                  |
| "RECEIVE"         | 3                   |
| "EVALUATE"        | 3                   |
+-------------------------------------------------+

5 rows available after 7 ms, consumed after another 0 ms
neo4j>
```

Listing 3.7 – *Retrieve Relationships with Relationships Count*

After retrieving the graph schema summary, the program extracts summary per label with all its nodes and relationships count. The output of this summary in tabular form for label "Person" is shown in Figure 3.6. From this point the process of handling missing values (see Section 4.1) takes place as shown in Figure 2.27. After handling missing values, the support computation takes place to find out the gradual pattern that have support greater than the min_threshold support.

	Label "PERSON" Data				Relationships Count		
Id	**Name**	**Age**	**Expr**	**Sal**	**No. Of Collaboration**	**No. of Publications**	**No. of Grants**
1	John	45	13	3500	3	3	2
2	Dave	?	7	2500	2	1	0
3	Emily	30	5	?	1	1	0
4	Tim	?	10	2800	1	2	0
5	Carl	44	14	3800	3	3	1
6	Chris	38	9	2600	3	1	0
7	David	32	5	1400	1	2	0

Figure 3.6 – *Property Graphs Gradual Pattern Mining Process*

3.3.4 Node-Properties-with-Relationships-Properties

In this type of scenario, the gradual patterns are extracted from the node properties and relationships properties. Since the relationships are treated as much important as node in property graphs so there may arise a situation where pattern need to be extracted from relationship properties. These patterns will be extracted for each label with the corresponding relationship types. The Listing 3.7 shows the relationship types with the respective count and an overall schema topology was shown in Listing 2.6 showing which relationships are connected between which source and destination label nodes.

Now first objective is to retrieve the relationships properties. The Listing 3.8 shows the properties for the running example. As we can see that for our running example, only keys are "since" and " data", which are not enough to fully explain the scenario. Therefore, to present this scenario, we use the synthetic graph generator to generate new property graph of nodes and *relationships with properties*. The detailed description about the graph generator is given in Section 5.2.

```
neo4j> MATCH (n:Person)-[r]->(m)
       RETURN labels(n),type(r), keys(r), labels(m),count(r);
+-----------------------------------------------------------------------+
| labels(n)   | type(r)        | keys(r)    | labels(m)    | count(r) |
+-----------------------------------------------------------------------+
| ["Person"] | "COLLABORATE"  | ["since"] | ["Person"]   | 6        |
| ["Person"] | "PUBLISH"      | []        | ["Paper"]    | 12       |
| ["Person"] | "PUBLISH"      | ["date"]  | ["Paper"]    | 1        |
| ["Person"] | "COLLABORATE"  | []        | ["Person"]   | 8        |
| ["Person"] | "RCV_GR"       | ["date"]  | ["Project"]  | 2        |
```

```
| ["Person"] | "RCV_GR"      | []         | ["Project"] | 1         |
+-------------------------------------------------------------------+

6 rows available after 12 ms, consumed after another 0 ms
neo4j>
```

Listing 3.8 – *Relationship Properties*

With the synthetic graph generator, just to explain the current scenario and for the sake of simplicity, we create a synthetic graph of 10 nodes and 40 relationships. We set 5 properties for each node namely PN1, PN2, PN3, PN4 and PN5. We set 3 properties for each relationship namely, PR1, PR2, and PR3. To keep the synchronization the running example, we keep same naming convention for node labels and relationship types. The details of node and relationships for this newly created property graph is shown in Listing 3.9 and the details of the relationships is shown in Listing 3.10.

```
neo4j> MATCH (n)
       RETURN labels(n) as LABEL, count(*) as nodeCount;
+---------------------------+
| LABEL        | nodeCount  |
+---------------------------+
| ["Person"]   | 3          |
| ["Paper"]    | 4          |
| ["Project"]  | 3          |
+---------------------------+

3 rows available after 28 ms, consumed after another 0 ms
```

Listing 3.9 – *Node and Relationship Details*

```
neo4j> MATCH (:Person)-[r]->()
       RETURN type(r), keys(r),count(*);
+-----------------------------------------------------------+
| type(r)       | keys(r)                  | count(*)  |
+-----------------------------------------------------------+
| "EVALUATE"    | ["PR1", "PR2", "PR3"]    | 2         |
| "RCV_GR"      | ["PR1", "PR2", "PR3"]    | 5         |
| "COLLABORATE" | ["PR1", "PR2", "PR3"]    | 5         |
| "PUBLISH"     | ["PR1", "PR2", "PR3"]    | 3         |
+-----------------------------------------------------------+

4 rows available after 8 ms, consumed after another 0 ms
neo4j>
```

Listing 3.10 – *Relationship Types with their Properties from Synthetic Property Graph*

As we can see in both Listings 3.9, and 3.10 for the created graph using synthetic graph generator, the distribution of 10 nodes in their respective labels is evenly distributed. Similarly, the distribution of 40 relationships their respective types is also evenly distributes. This is due to the fact that, we select the feature of " Uniform distribution" in GUI interface of the tool for creation of node and relationships.

To extract the patterns, the program needs to check the node properties along with its relationships properties for each label from the property graph. The Listing 3.11 shows the output for label Person and its related relationship properties with values.

```
neo4j> MATCH (n:Person)-[r]->()
       RETURN distinct ID(n), n.PN1, n.PN2, n.PN3, n.PN4, n.PN5,
       r.PR1, r.PR2, r.PR3;
+-------------------------------------------------------------------------------------------+
| ID(n) | n.PN1 | n.PN2 | n.PN3 | n.PN4 | n.PN5 | r.PR1 | r.PR2 | r.PR3 |
+-------------------------------------------------------------------------------------------+
| 2     | 7     | 1     | 81    | 72    | NULL  | 92    | 68    | 52    |
| 2     | 7     | 1     | 81    | 72    | NULL  | 97    | 0     | 2     |
| 2     | 7     | 1     | 81    | 72    | NULL  | 8     | 23    | 36    |
| 2     | 7     | 1     | 81    | 72    | NULL  | 15    | 74    | 8     |
| 2     | 7     | 1     | 81    | 72    | NULL  | 35    | 21    | 30    |
| 4     | 52    | 98    | 45    | 63    | NULL  | 99    | 96    | 33    |
| 4     | 52    | 98    | 45    | 63    | NULL  | 62    | 48    | 82    |
| 4     | 52    | 98    | 45    | 63    | NULL  | 1     | 74    | 24    |
| 4     | 52    | 98    | 45    | 63    | NULL  | 55    | 88    | 10    |
| 4     | 52    | 98    | 45    | 63    | NULL  | 75    | 99    | 58    |
| 9     | 67    | 1     | 47    | 7     | 54    | 30    | 52    | 22    |
| 9     | 67    | 1     | 47    | 7     | 54    | 30    | 25    | 35    |
| 9     | 67    | 1     | 47    | 7     | 54    | 93    | 51    | 38    |
| 9     | 67    | 1     | 47    | 7     | 54    | 68    | 10    | 93    |
| 9     | 67    | 1     | 47    | 7     | 54    | 53    | 4     | 26    |
+-------------------------------------------------------------------------------------------+

15 rows available after 5 ms, consumed after another 2 ms
```

Listing 3.11 – *Node Properties with Relationship Properties from Synthetic Property Graph*

Once the node properties and relationships are retrieved by the program, it proceed to the process of extracting pattern according to the Algorithms as presented in 1, 2. It must be noted that we have shown the explicit queries to explain the scenario, but for the program this process has to automatic retrieval of data, then pre-processing and finally pattern extraction.

3.3.5 Inter-Relationships-Properties

In *Inter-Relationships-Properties*, for extracting patterns we are only concerned with the relationship properties regardless of the node labels present in the graph schema. The resulting patterns will be showing the co-variance only between the relationship properties. To explain this scenario, we use the same synthetic graph that is generated for the example data as described in Section 3.3.4. The Listing 3.12 show the output of synthetic graph relationships type PUBLISH with its properties and Listing 3.13 the output of properties for all relationship types.

```
1
2
3  neo4j> MATCH ()-[r:PUBLISH]->()
4         RETURN type(r), keys(r),count(*);
5  +-------------------------------------------------------+
6  | type(r)    | keys(r)                  | count(*) |
7  +-------------------------------------------------------+
8  | "PUBLISH" | ["PR1", "PR2", "PR3"] | 13       |
9  +-------------------------------------------------------+
10
11 1 row available after 22 ms, consumed after another 1 ms
12
13
14 neo4j> MATCH ()-[r:PUBLISH]->()
15        RETURN distinct ID(r), r.PR1, r.PR2, r.PR3;
16 +-----------------------------------+
17 | ID(r) | r.PR1 | r.PR2 | r.PR3 |
18 +-----------------------------------+
19 | 41    | 29    | 81    | 11    |
20 | 21    | 6     | 53    | 51    |
21 | 1     | 90    | 46    | 5     |
22 | 33    | 52    | 89    | 51    |
23 | 13    | 11    | 92    | 23    |
24 | 45    | 82    | 37    | 100   |
25 | 25    | 47    | 52    | 15    |
26 | 5     | 83    | 45    | 38    |
27 | 37    | 47    | 77    | 5     |
28 | 17    | 94    | 21    | 36    |
29 | 49    | 30    | 52    | 22    |
30 | 29    | 93    | 51    | 38    |
31 | 9     | 53    | 4     | 26    |
32 +-----------------------------------+
33
34 13 rows available after 53 ms, consumed after another 5 ms
35 neo4j>
```

Listing 3.12 – *PUBLISH Relationship type with Properties Synthetic Property Graph*

```
neo4j> MATCH ()-[r]->()
       RETURN distinct ID(r), r.PR1, r.PR2, r.PR3;
+-----------------------------------+
| ID(r) | r.PR1 | r.PR2 | r.PR3 |
+-----------------------------------+
| 49    | 30    | 52    | 22    |
| 43    | 35    | 39    | 57    |
| 37    | 47    | 77    | 5     |
| 34    | 62    | 48    | 82    |
| 9     | 53    | 4     | 26    |
| 4     | 75    | 99    | 58    |
| 39    | 30    | 25    | 35    |
| 27    | 48    | 35    | 23    |
| 25    | 47    | 52    | 15    |
| 23    | 43    | 94    | 25    |
| 6     | 52    | 36    | 82    |
| 0     | 59    | 51    | 78    |
| 5     | 83    | 45    | 38    |
| 44    | 99    | 96    | 33    |
| 41    | 29    | 81    | 11    |
| 36    | 85    | 48    | 44    |
| 31    | 43    | 33    | 16    |
| 21    | 6     | 53    | 51    |
| 19    | 68    | 10    | 93    |
| 16    | 71    | 69    | 21    |
| 1     | 90    | 46    | 5     |
| 48    | 41    | 59    | 40    |
| 45    | 82    | 37    | 100   |
| 24    | 1     | 74    | 24    |
| 47    | 1     | 66    | 65    |
| 18    | 76    | 15    | 38    |
| 12    | 15    | 74    | 8     |
| 11    | 81    | 23    | 48    |
| 7     | 69    | 79    | 99    |
| 42    | 92    | 68    | 52    |
| 38    | 71    | 81    | 28    |
| 33    | 52    | 89    | 51    |
| 15    | 29    | 19    | 93    |
| 13    | 11    | 92    | 23    |
| 10    | 54    | 34    | 92    |
| 2     | 35    | 21    | 30    |
| 40    | 16    | 21    | 6     |
| 35    | 8     | 93    | 10    |
```

```
| 30     | 92     | 15     | 81     |
| 26     | 28     | 18     | 38     |
| 14     | 55     | 88     | 10     |
| 46     | 93     | 1      | 91     |
| 28     | 76     | 84     | 32     |
| 22     | 8      | 23     | 36     |
| 17     | 94     | 21     | 36     |
| 32     | 97     | 0      | 2      |
| 29     | 93     | 51     | 38     |
| 20     | 43     | 25     | 69     |
| 8      | 51     | 62     | 38     |
| 3      | 98     | 78     | 65     |
+-----------------------------------+

50 rows available after 3 ms, consumed after another 3 ms
neo4j>
```

Listing 3.13 – *Properties for all Relationship types from Synthetic Property Graph*

In Listing 3.13, we retrieve the proprieties of all relationships whatever their types. We do not differentiate a property named PR1 for a relationship of type PUBLISH than a property of any other relationship type for example COLLABORATE. An other scenario could have been to group properties by relationship types. For instance, we could have a column with PUBLISH.PR1 and another with COLLABORATE.PR1 and so on for all the relationship types and their respective properties. We have not chosen to present this alternative scenario in this section but it is also a legitimate scenario that can be exploited in further works.

3.4 Fuzzy Property Graph Gradual Pattern

This section relies on the definitions of fuzzy gradual items and patterns as defined in the last chapter.

The concepts have to be extended when dealing with property graph data in nodes and relationships. As discussed in Section 2.3, a property graph is a combination of nodes and relationships as shown in Figure 2.5. The *property graph node gradual item* is defined in Definition 3.4 and *property graph relationship gradual item* is defined in Definition 3.5. Based on these, we define the the concept of *fuzzy property graph gradual item* for node properties.

Let Λ_N be a set of node labels with $NULL \in \Lambda_N$, Π_N be a set of properties, P_N a set of (key:value) pairs over Π_N, Λ_R be a set of relationship Types with $NULL \in \Lambda_R$, Π_R be a set of properties, P_R a set of (key:value) pairs over Π_R. Let $G_D = (N, R)$ be a graph defined over Λ_N, P_N, Λ_R and P_R. Let $\mathcal{P}_i$ be a fuzzy partition of the domain of property $i \in \Pi_N \cup \Pi_R$ within the fuzzy sets $\{f_j\}_{j \in [1,n_i]}$. A graph data gradual item is a fuzzy gradual item i, f_j, δ where $i \in \Pi_N \cup \Pi_R$, $f_j \in \mathcal{P}_i$ and $\delta \in \uparrow, \downarrow$.

Id	Age	Age_Low	Age_High
$p1$	26	1	0
$p2$	57	0.6188	0.3811
$p3$	78	0	1

Table 3.2 – *Age Fuzzy Sets*

Example 9. (Expr, Expr_Low, ↑) is a fuzzy property graph node gradual item expressing "the more the Experience is Low" , where "Expr" is the node property under consideration, "Expr_Low" is the fuzzy set and "↑" is the associated variation with the property.

In order to mine fuzzy gradual patterns from property graphs, data have to be transformed in order to get fuzzy descriptions of the information. For this purpose, we consider the use of fuzzy partitions over the universes of the retrieved the properties (e.g., Age, Salary, Experience for Label Person for the running example) and the relationships (e.g., number of Publications). The data are transformed from quantities to the degree of membership for every subset of the partitions being considered, as for instance Low and $High$ for the "Age" property.

For instance, we can consider the combination of $L-function$ and $R-function$ and user-provided partitions as in Figure 3.7 where the thresholds have been given or computed by a given protocol.

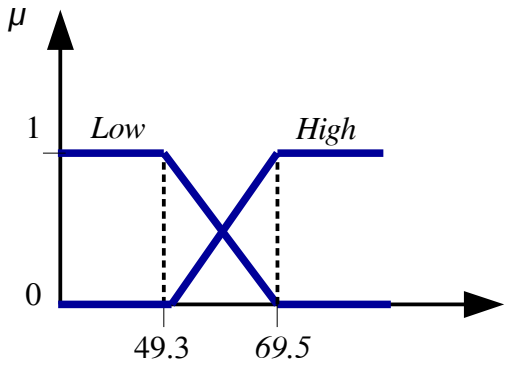

Figure 3.7 – *Fuzzy Partition*

Being provided with such a partition (with 2 elements here) allows us to transform the data by computing every membership degrees as shown in Table 3.2.

Therefore, for the sake of experiments on the property graphs to extract fuzzy gradual patterns, we used synthetic graph generator (Section 5.2) to generate the node properties with symmetric range of values for all properties. For example if the user input in the graph generator to have three properties namely $A1$, $A2$, and $A3$ for the given number of node (e.g. 5 nodes), the output for the fuzzy sets is shown in Table 3.3.

Once the data is retrieved from property graphs and fuzzified according to the membership

Id	A1	A1_Low	A1_High	A2	A2_Low	A2_High	A3	A3_Low	A3_High
$n1$	50	0.965	0.034	71	0	1	33	1	0
$n2$	55	0.717	0.282	15	1	0	62	0.371	0.628
$n3$	NULL	NULL	NULL	54	0.767	0.232	65	0.222	0.777
$n4$	60	0.470	0.529	2	1	0	NULL	NULL	NULL
$n5$	45	1	0	51	0.915	0.084	76	0	1

Table 3.3 – *Data Fuzzification*

function then the next step it to extract the extract the gradual pattern. In the following section we discuss the extraction of these patterns as well handling of missing data if they appear in the properties.

Chapter 4

Extracting Gradual Patterns from Property Graphs

Id	Name	Age
1	John	45
3	Emily	30
5	Carl	44
6	Cris	38
7	David	32

Table 4.1 – $|vdb\ \ (Age)|$

Id	Name	Sal
1	John	3500
2	Dave	2500
4	Tim	2800
5	Carl	3800
6	Cris	2600
7	David	1400

Table 4.2 – $|vdb\ \ (Sal)|$

4.1 Dealing with Missing Values for Mining Gradual Patterns

Mining gradual patterns from property graphs is a novel ideal as it involves automatic retrieval of graph data including nodes and relationships. It becomes more challenging because the retrieved information often contains missing value due to inherent nature of property graphs. Therefore, we need to devise a mechanism for this kind of mining task when there exist missing values in the data.

Gradual Pattern extraction from tabular data has been studied [DLT09, LLR09, Do+15], but there does not exist any approach for such a task from property graphs. Due to inherent nature of property graph, a node may not contain all the properties. This data may be missing because it does not exist or is not applicable (for example no salary and experience in the case of kids or a response to question that is only to be given by woman and not by man etc.,). Therefore, it is not possible to apply existing algorithms of gradual pattern extraction due to the semi-structure nature of property graphs.

We can see in Table 3.1, there are missing values in "Age" and "Sal" attributes. To represent this missing data of nodes in tabular form we use "?" sign. For the sake of representation, this missingness is shown in different colors in Figure 3.4. We present how to deal with such "missing" values in gradual patterns extraction.

For the treatment of missing values in the context of property graphs, We consider the *"vdb"* approach presented in [RC98]. The *"vdb"* is the cardinality of an attribute without missing values. Let n be the number of tuples of an attribute and a_m be the number of missing values in that attribute then the *vdb* of that attribute of the dataset is:

$$|vdb| = |n| - |a_m| \tag{4.1}$$

Therefore, the *"vdb"* calculation for (Age) is 5 and for (Sal) is 6 as shown in Table 4.1 and Table 4.2 respectively.

Let us see with an example about the issue of missing data and then we will propose our approach to handle the missing data. There are seven nodes for Label "Person" in which some

Id	Name	Age	Expr
1	john	45	13
2	dave	?	7
3	emily	30	5
4	tim	?	10
5	carl	44	14
6	cris	38	9
7	david	32	5

Table 4.3 – *Retrieved Data from Graph for Age & Expr*

nodes are missing properties like age and salary as shown in Figure 3.4. In property graph, in response to a projection cypher query, a missing property (i.e., *key*) and a missing value of the property are treated as same. The result of the cypher query will be "NULL", if the node contains a missing value or a missing property [Bow19]. Now, the first step is to transform this data into tabular form by querying the graph as shown in Figure 2.27. The second step is to represent these properties in a binary order of matrix as explained in Section 2.4. The binary matrices of $(Age \uparrow)$ and $(Expr \uparrow)$ representing the order are shown in Figure 4.1.

As we can observe in Table 4.3, we have missing data of Age for Id 2 and 4. This missingness is represented with the "?" sign in the table and in the binary matrix representing order. Since, we do not know the missing value of dave's age, so when representing the order between John's age and Dave's age, we place "?" sign in the 2nd row of 1st column of the matrix $(Age \uparrow)$ in Figure 4.1. The same procedure applies for John and Tim's missing age, we place "?" sign in the 4th row of 1st column in the matrix. For the binary AND operation of $(Age \uparrow)$ and $(Expr \uparrow)$, we consider the fact that any bit (1 or 0) multiplied with "?" sign will result in "?" sign. Hence, the resulting Hadamard product of these two gradual items in matrix representation is shown in Figure 4.2.

The *"vdb"* is the cardinality of an attribute without missing values. When combining several attributes, the value of vdb reduces as it results in the elimination of the entire tuple for the considered attributes in order to have no missing data. [RC98] states that, for a given itemset, we cut the dataset into a valid database *"vdb"*, such that the *"vdb"* must not have any missing values. In our approach, it requires us to preserve the location of missing value, so as to maintain the cardinality for computing the support when combining multiple attributes.

The *vdb* representation in tabular form when combining $|vdb\ (Age\ \&\ Expr)|$ attributes is shown in Table 4.4. Similarly, for $|vdb\ (Age\ \&\ Sal)|$ shown in Tables 4.5 and $|vdb\ (Age\ \&\ Sal\ \&\ Expr)|$ in Table 4.6 respectively. As we can observe that the size of *vdb* is the same for (Age & Sal) and (Age & Sal & Experience) because the $Expr$ attribute does not have any missing value.The calculation of *"vdb"* for label "Person" is given below:

$$|vdb(Age)| = 5,\ \ |vdb(Sal)| = 6,\ \ |vdb(Expr)| = 7$$

$$|vdb(Age\ \&\ Sal)| = 4, \quad |vdb(Age\ \&\ Expr)| = 5$$
$$|vdb(Sal\ \&\ Expr)| = 6$$
$$|vdb(Age\ \&\ Sal\ \&\ Expr)| = 4$$

For the gradual pattern extraction process, *vdb* calculation is an essential part because the value of vdb is then used for support computation.

Id	Name	Age	Expr
1	john	45	13
3	emily	30	5
5	carl	44	14
6	cris	38	9
7	david	32	5

Table 4.4 – $|vdb\ (Age\ \&\ Expr)|$

Id	Name	Age	Sal
1	john	45	3500
5	carl	44	3800
6	cris	38	2600
7	david	32	1400

Table 4.5 – $|vdb\ (Age\ \&\ Sal)|$

Id	Name	Age	Sal	Expr
1	john	45	3500	13
5	carl	44	3800	14
6	cris	38	2600	9
7	david	32	1400	5

Table 4.6 – $|vdb\ (Age\ \&\ Sal\ \&\ Expr)|$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & ? & 0 & ? & 0 & 0 & 0 \\ ? & ? & ? & ? & ? & ? & ? \\ 1 & ? & 0 & ? & 1 & 1 & 1 \\ ? & ? & ? & ? & ? & ? & ? \\ 1 & ? & 0 & ? & 0 & 0 & 0 \\ 1 & ? & 0 & ? & 1 & 0 & 0 \\ 1 & ? & 0 & ? & 1 & 1 & 0 \end{bmatrix} \end{array}$$

(Age ↑)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

(Expr ↑)

Figure 4.1 – *Binary matrix representing orders for Table 4.3*

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & ? & 0 & ? & 0 & 0 & 0 \\ ? & ? & ? & ? & ? & ? & ? \\ 1 & ? & 0 & ? & 1 & 1 & 1 \\ ? & ? & ? & ? & ? & ? & ? \\ 0 & ? & 0 & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & 1 & 0 & 0 \\ 1 & ? & 0 & ? & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Figure 4.2 – *Hadamard product for binary AND operation of Age AND Expr*

4.2 Support Computation

The existing support and confidence measures are misleading when there exist missing data in such a way that they are lacking in the crucial monotonicity property of support [CGM07]. We are interested to compute the "support" with valid databases. We are not computing the confidence as the objective over here is not rules formation. The anti-monotonicity of the support can not be considered anymore in a simple manner because support cannot be computed object by object but requires to consider the ranking of all objects.

In order to compute the support, we compute the logical AND of gradual items as explained in Section 3.3.3. From the resultant matrix, we take the sum of binary '1' bit from the matrices. Consequently, we calculate the support as given below:

$$Support(X_i) = \frac{Sum\ of\ concordant\ pairs}{|vdb(X_i)| * (|vdb(X_i)| - 1)/2} \tag{4.2}$$

$$where,\ X_i = itemset$$

In [RC98], authors suggest, *"To obtain good results a vdb must be a good sample of the database. This is normally true if values are missing at random"*, therefore a new parameter *"representativity"* is introduced. The representativity is the proportion of the $vdb(X_i)$ over the entire dataset tuples. The itemset should be representative so as to be considered for support computation. Representativity is a user defined parameter. It helps to ensure that support computation should not take place for the itemset which is not representative i.e., having value less than user defined value. For the running example, the representativity is calculated as follows:

$$Representativity(X_i) = \frac{|vdb(X_i)|}{|n|} \tag{4.3}$$

$$where,\ X_i = itemset,\ n = number\ of\ tuples$$

$$|Rep(Age)| = 5/7,\ \ |Rep(Sal)| = 6/7,\ \ |Rep(Expr)| = 7/7$$
$$|Rep(Age\ \&\ Sal)| = 4/7,\quad |Rep(Age\ \&\ Expr)| = 5/7$$
$$|Rep(Sal\ \&\ Expr)| = 6/7,\quad |Rep(Sal\ \&\ Expr)| = 6/7$$
$$|Rep(Age\ \&\ Sal\ \&\ Expr)| = 4/7$$

4.3 Algorithms

The overall process of mining gradual patterns includes following steps.

1. Connection to the graph database.
2. User specified properties (attributes) extraction from property graph.
3. Pre-process the data including checking for missing values in attributes.
4. Handling the missing data (if exist)
5. Extraction of patterns.

The process is shown in Figure 2.27 and to perform the step 4 and step 5 we show the two core algorithms as follow.

4.3.1 Algorithm 1: Mining Property-based Gradual Items

Algorithm 1 shows the procedure for computing the support based on the *vdb* method as explained in Section 4.2. After the retrieval of data from the graph data, following are the main steps.

- Store the items (properties/attribute(s)) in an array list
- Initialize the binary matrix for each item of the list
- Compute the binary order matrix
- Compute the sum of high "1" bits and missing data represented as "?"
- compute the vdb (cardinality of valid data) to be used for support computation

Algorithm 1: Mining Property-based Gradual Items in the Presence of Missing Values

Input: Properties, minSupport

Output: List of gradual items having support greater than minSupport

```
1: Initialize the matrices for each property and store into list L
2: for all items in list L do
3:    for all rows of matrix M for listItem L_i do
4:       for all columns of row of M do
5:          if L_i.M[row][col] == 1 then
6:             sum ← sum + 1
7:          else if L_i.M[row][col]==? then
8:             card ← card + 1
9:          end if
10:         if card == L_i.M.length then
11:            flag ← flag + 1
12:         end if
13:      end for
14:   end for
15:   vdb = (L_i.M.length - flag)
16:   support = sum / (vdb * (vdb-1)/2)
17:   if support < minSupport then
18:      remove L_i
19:   end if
20: end for
21: Update the list L with successful gradual items of size-1
```

- compute the support and update the list

An explanation of these steps is presented as follows. After the retrieval of data from the graph data, binary matrix initialization for each property is performed. Then we calculate binary-ordered representation for each item. These matrices contain 0 or 1 and missing values are represented by "?" sign. Then, we then calculate the sum of all high-bits i.e., 1 which represent the concordance for the respective item. A complete row with "?" sign in matrix represents the missing value in that attribute as shown in Figure 4.1, (Age↑) for row 4 i.e., Tim's age is missing. To keep track of this, a *card* variable is used. For instance, in Figure 4.1, (Age↑), the number of rows containing "?" are 2. The *flag* variable is used to track the the length of matrix that in turns helps to calculate *vdb* i.e., the cardinality of that item without missing values. Finally, the support is computed as expressed in Equation 4.1. If the item's support is less than the minimum support threshold, then it is removed from the list.

4.3.2 Algorithm 2: Mining Property-based Gradual Patterns

Once we have the list of successful gradual items, i.e., those having $(support > minSupport)$ the program check for gradual patterns as defined in Definition 2.4. Therefore, the Hadamard product of binary AND operation of gradual items is performed and then the support is calculated. The successful gradual patterns are updated in the list and those which do not meet the minimum support requirement are removed. This is performed in Algorithm 2. Following are the main steps:

- Take the list of input gradual items that are result of Algorithm 1 and multiply L_i item with L_j item.
- Perform the Hadamard product for L_i AND L_j.
- Compute the binary order matrix
- Compute the sum of high "1" bits and missing data represented as "?"
- Compute the vdb (cardinality of valid data) to be used for support computation and compute support
- Update the list of gradual patterns and go for next item in L_i
- Output the list of successful gradual patterns

Regarding the complexity of the algorithms, research shows that pattern mining algorithms are known to be NP-hard. It is concluded that the complexity of enumeration problem for mining maximal frequent itemsets is NP-hard [Yan04, Han+07].

4.4 Embedding Gradual Patterns Mining within a Graph Database

In this section, we are introducing our work to integrate gradual patterns mining features inside a graph database. First, we will explore several ways to integrate functionalities inside a graph database engine such as Neo4j. Then, we will challenge them and explain our choice. A proof-of-concept has been made to demonstrate our approach. We will conclude this section by giving more details on our implementation choices and the limits of the current implementation.

4.4.1 Integrating Features in a Graph Database Engine

Before going further in this chapter, we introduce some concepts about the internal architecture of a graph database.

Algorithm 2: Mining Gradual Patterns in the Presence of Missing Values from Gradual Items

Input: List L_i of gradual items, minSupport,

Output: Frequent Property-based Gradual Patterns

while items in list $|L_i| ! = 0$ **do**
  for all listItem L_i **do**
    for all listItem $L_j = L_i + 1$ **do**
      for all rows of matrix M for listItem L_i **do**
        for all columns of row of M **do**
          $resultM[row][col] = L_i.M[row][col]$* $L_j.M[row][col]$
          if resultM[row][col] == 1 **then**
            $sum \leftarrow sum + 1$
          else if resultM[row][col] == ? **then**
            $card \leftarrow card + 1$
          end if
          if card== resultM.length **then**
            $flag \leftarrow flag + 1$
          end if
        end for
      end for
      vdb = ($L_i$resultM.length - flag)
      support = sum / (vdb * (vdb-1)/2)
      if support $<$ minSupport **then**
        remove L_i
      end if
    end for
  end for
  Update the list L
end while
Output the frequent patterns

4.4.1.1 Preliminary Concepts: Internal Architecture of Graph Databases

A graph database engine is composed of several components such as:

- A declarative query language like SPARQL or Cypher that allows an end-users to explain **what** they want to do. The system will then determine the most optimized way, or at least try to do so, to accomplish the user request.
- Some programming interfaces that allow a developer to explain to the system **how** to realize the desired operation.
- Low-level system (kernel) operations that execute core operations on the graph storage engine.

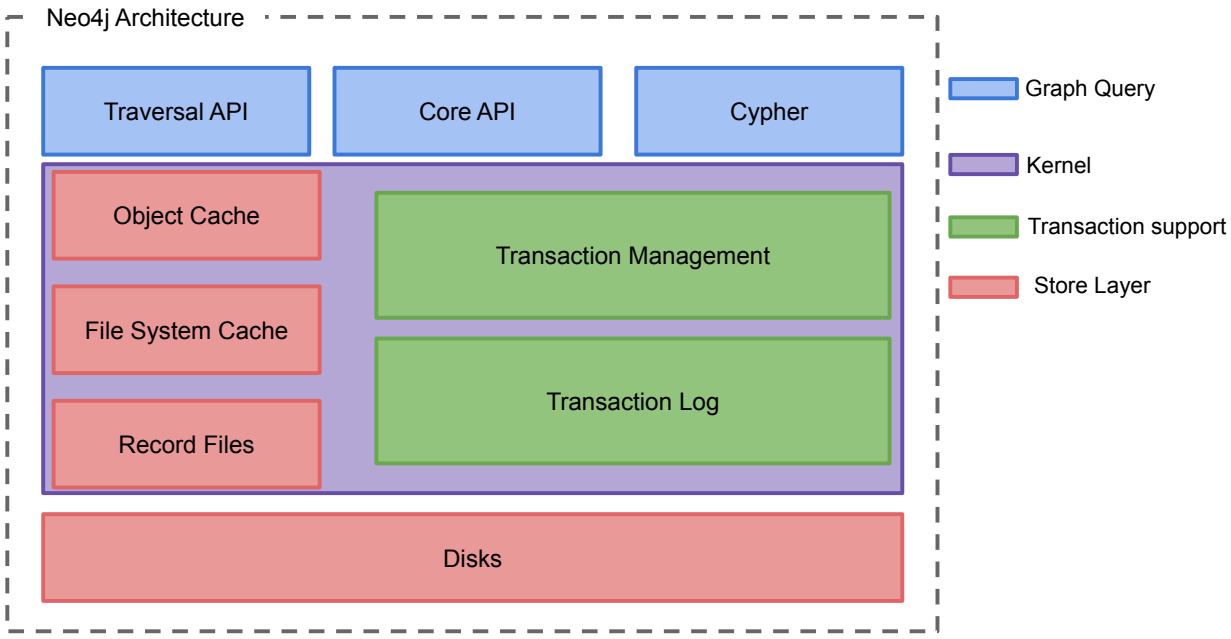

Figure 4.3 – *Neo4j Internal Architecture*

This list is non-exhaustive and there are often more components in a graph database system. The Figure 4.3 shows the internal architecture of the Neo4j database. As the reader can see, this graph database management system also have a transaction management system.

Also, every graph database do not have a declarative query language even if during the last years there is a trend to provide one in every graph database engine. Some initiatives have emerged to standardise declarative languages, such as the OpenCypher [1] initiative or GQL [2] (Graph Query Language).

4.4.1.2 Adding Features to a Graph Database

In the previous section we described the internal architecture of graph databases. It highlights some components that can be extended to provide new functionalities to the database.

The first possibility is to choose a low-level implementation, extending core features inside the graph database kernel. Doing so can be a wise choice when, for example, we have to manage records or add new low-level capacities. This is the case for example when we want to add a new functionality such as spatial and geographic objects or to add some special indexes such as R-tree (proposed by Antonin Guttman in 1984 [Gut84]).

1. https://www.opencypher.org/
2. https://gql.today/

The second possibility is to choose to enhance the API[3] by providing new code and interfaces that rely on the kernel implementation to build new functionalities that can be exploited by developers. It is a common use-case to do so for adding new traversal algorithms such as neighbourhood-search (e.g. returns distinct nodes of the given relationships in the pattern up to a certain distance) or to add utilities functions (temporal functions, text functions, etc.).

The third possibility is to extend the graph database at the declarative language layer. There are several ways to do so:

- Creating an overlay layer upon the Domain Specific Language (DSL). The concept is to create a dedicated expressive language that will be processed to generate Cypher well-formed queries. This approach has been used in [Piv+16] where the authors provide a system for querying graph databases in a flexible way by offering a transcriptor (SUGGAR) that generates Cypher queries from a new query language (FUDGE) and execute those queries on the graph database engine. This scenario can be seen as an additional layer at the top of Figure 4.4.
- Extending the declarative language and binding it to the current low-level API. In this scenario, the declarative language is extended but the user is still using the same underlying API. This scenario can work for some utility functions or some business functionalities that do not require to add new complex code. Adding fuzzy logic in a system without adding new functions may be hard to achieve (or even not possible).
- Extending the declarative language and adding new APIs to offer the best of the two worlds: on the one hand adding new API offers to the developers new possibilities and on the other hand extending the declarative language offers to the end-user the ability to use it.

The Figure 4.4 summarizes the approaches presented above.

4.4.2 Integration Challenges - Discussion

As described in Section 4.4.1.2, there are several ways to extend a graph database. We will not get in too much details about what are the pros and cons of each solution as this topic as already been cover in [CM18]. In short, each of them has pros and cons but what really matter here is to determine what is the one that fit the best our needs.

It seems pretty obvious that using an overlay upon a declarative language is not a good choice. On the one hand, declarative languages describe *what* the user want to do, and not *how* the user want it to be done. On the other hand, our goal is to explain to the engine how to process gradual patterns. Thus, it seems pretty difficult or, at least, really inefficient to try to

3. A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.

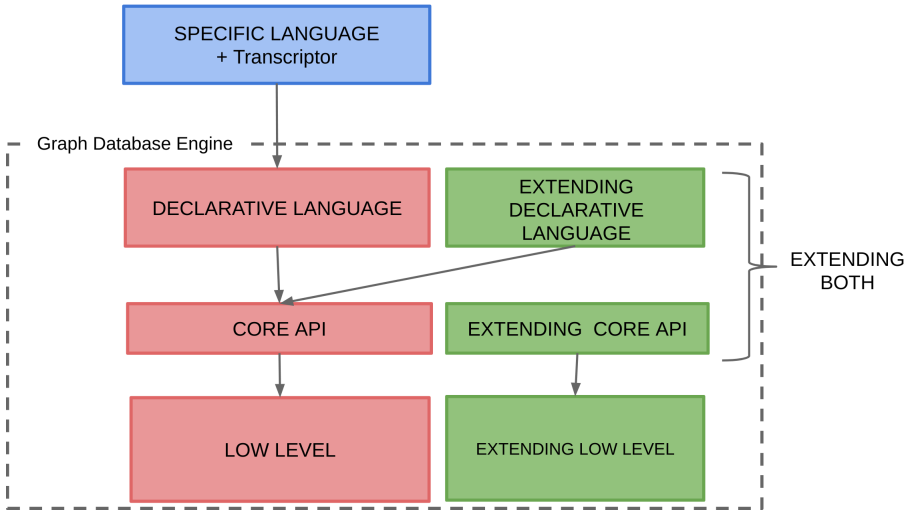

Figure 4.4 – *Extension possibilities*

drive a declarative language, that is by definition focusing on the **what**, to ask the engine to do implement gradual mining over property-graph.

Using Low-level API could be a solution if we want to persist our binary matrices (as a new level storage object type) or if we would like to add new indexes that keep track of gradual patterns. This does not solve our primary issue but as we think that this could be an interesting further step, therefore, we have added it as a perspective and discussed it in a more detail in Section 6.2.1.4.

Our main concern is to provide some gradual mining feature inside graph database engines. There is two kind of users that we can target:

- For the developers, we have to offer a library and some APIs
- For the end-users, we have to provide a high-level extension; meaning extending the declarative language and to bind it on an API (specific or not).

We think that we need to address both kind of users. We then have (i) to create a library and add an API to enhance core APIs with new features and (ii) to provide an extension of the query language.

The next section will focus on the specification of the API and Section 4.4.4 will focus on how to extend the declarative language on a real use-case scenario.

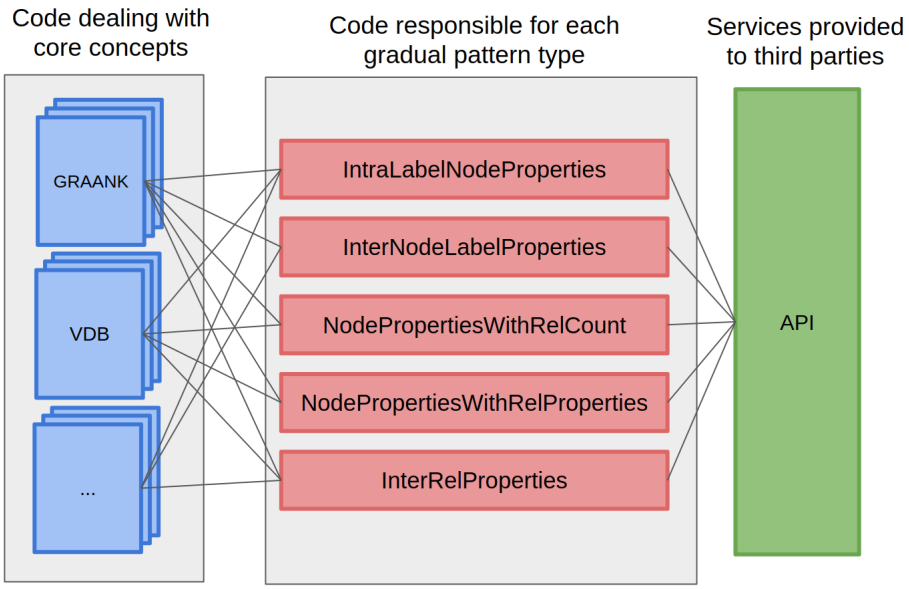

Figure 4.5 – *API Service Illustration*

4.4.3 API Specification

In this section, we are describing the specifications of the library that we have to define to allow developers to retrieve gradual patterns for following scenarios:

- Intra-Label-Node-Properties as defined in Section 3.3.1
- Inter-Node-Label-Properties as defined in Section 3.3.3
- Node-Properties with Relationships Count as defined in Section 3.3.3
- Node-Properties-with-Relationships-Properties as defined in Section 3.3.4
- Inter-Relationships-Properties as defined in Section 3.3.5

For each of these scenarios, we don't want that developers have to deal with this underlying complexity or to require them to understand the code structure and the dependencies between our classes. Thus, we suggest to use the API Facade design pattern as illustrated in Figure 4.5.

The Facade design pattern is one of the twenty-three well-known GoF design patterns[Gam95]. Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code.

As a proof-of-concepts that we are able to integrate our work inside a graph database engine we have chosen to integrate the Intra-Label-Node-Properties scenario inside the Neo4j database. We need now to define the features that will be available.

4.4.3.1 Defining a Service

Adding the Intra-Label-Node-Properties into Neo4j implies to define what services will be provided by the API to developers.

In our point of view, developers should be able to find gradual patterns by passing the label and the minimum support threshold that will be used. Then the code should retrieve the data, execute the operation accordingly with what has been defined in Section 3.3.1 and return the result.

Also, we think that we should also provide an overloaded version of this function that gives a default value (0.5) for the minimum support to allow developers to only give the label as parameter is suitable.

We have defined a service named SearchGradualPatternsService that implements and respects this specifications. This service is acting as an API and is responsible to offer searching/mining features. It will be completed with the other scenarios in further works, we invite the reader to see Section 6.2 for more information.

4.4.3.2 Treating Unsortable Columns

As explained in Section 2.3.1, the schema-less nature of property graphs implies that there is no predefined schema. We do not have meta-data information on each property of a node neither do we for relationships. It is then difficult to know if a column is sortable or unsortable. In chapter 2, we explained that this is an important matter for discovering gradual patterns.

We also are highlighting the reader that a property with the same name could be used in two different nodes for storing two different types of data. For instance node A could have a property named prop1 with value 12 (integer) and node B could also have a property named prop1 with a value of false (boolean). We are dealing with this issue in two ways:

- A naive approach, that is mainly a type checking approach. We iterating on the properties of all the node that we have to treat (e.g: all node with the same label in the Intra-Node-Properties scenario) and every time that a node has a property value that is non-numerical then we remove the property from the list of properties to use for gradual patterns. This is not the best approach as we can have some properties that are not numerical but have an order (such as academic rank) and we can also have numerical properties that do not have a functional/business order, that is to said that the numerical order is does not make sense on it.
- Using a user-knowledge based approach. In that scenario, we are providing a third function specification in the API that allows the user through a new parameter named ‘ ‘skipProperties” to give an array of all the properties that should be skiped whether it is be-

cause they are not sortable or for any other user-based consideration. This can be seen as a pre-filter phase before the execution of the algorithms described in Section 4.3.

To efficiently process a property that is an ordinal variable we need to go beyond and offer a way to the user to give a comparison function that allow our program to order the categories of the property. This has not been addressed during the thesis but is noted as a perspective in Section 6.2.1.

4.4.4 Extending Neo4j

In this section, our purpose it to demonstrate the feasibility of integrating our algorithms into Neo4j. In this section, we look into the new techniques offered by Neo4j to integrate user-defined features and the integration of the Intra-Node-Label-Properties scenario into it. The section is concluded by a running example.

4.4.4.1 User-defined procedures and User-defined functions

In the latest versions of Neo4j, it is now possible to use a new system to extend the standard functionalities:

- Adding new procedures and functions that extend the capabilities of the Cypher query language
- Authentication and authorization plugins that extend the Neo4j security framework
- Server extensions that enable new surfaces to be created in the HTTP API

In our case, the first item is the one that can help us to extend Neo4j by providing gradual patterns functionalities. Of course, writing extensions requires to be familiar with Java (or Scala or any JVM compliant language).

User-defined procedures and User-defined functions are mechanism that allow the user to write code that can be invoked directly from Cypher. Thus, it is a good way to provide access to third-party code. The main difference between user-defined procedures and user-defined functions lies in the fact that functions are read-only and only return a single value.

This mechanism is a component-based scan approach that browses the class-loaders to find class with some annotations and add it to a registry of available functionalities. It offers an easy way to extend both the declarative language and the core API to offer a new feature as presented in Figure 4.4.

Previously, before the release of user-defined procedures mechanism in neo4j, we had to extend the Parser Expression Grammar (PEG) [For04] as it has been done in [CM18]. A Parsing Expression Grammar (PEG) is a recognition-based grammar formalism.

4.4.4.2 Integrating Intra-Label-Node-Properties Gradual Patterns

The specification presented in Section 4.4.3 lead to three function signatures:

- fr.lirmm.gradualpatterns.GPIntraNode(labelname, minsup) - find GP for a label with minimal support of 'minsup'
- fr.lirmm.gradualpatterns.GPIntraNode(labelname) - find GP for a label with minimal support value of O.5
- fr.lirmm.gradualpatterns.GPIntraNode(labelname, minsup, skipProperties) - find GP for a label with minimal support and skip properties that are defined in the skipProperties array

As we can see, these signatures can be seen as overloading signature for a unique function.

To implement this functionality we have created a user-defined function that takes three parameters: the label, the minimal support as defined in section 4.4.3 and an array named skipProperties.

```
@UserFunction(value = "lirmm.gpintranode")
@Description("fr.lirmm.gradualpatterns.GPIntraNode(labelname,
 ↪  minsup) - find GP for a label with minimal support of 'minsup'
 ↪  (default 0.5) and exclusion of properties found in
 ↪  skipProperties array")
public String GPIntraNode(@Name("label") String label,
 ↪  @Name(value="minsup", defaultValue="0.5") Double minsup,
 ↪  @Name(value="skipProperties", defaultValue="") List<String>
 ↪  skipProperties) {
        ...
}
```

The code of the above listing can be invoked in cypher with statement defined in listing 4.1 ,4.2 and 4.3. You can also see screenshot of the execution of that function in Figure 4.6, 4.7 and 4.8.

```
RETURN lirmm.gpintranode('Person') AS result
```

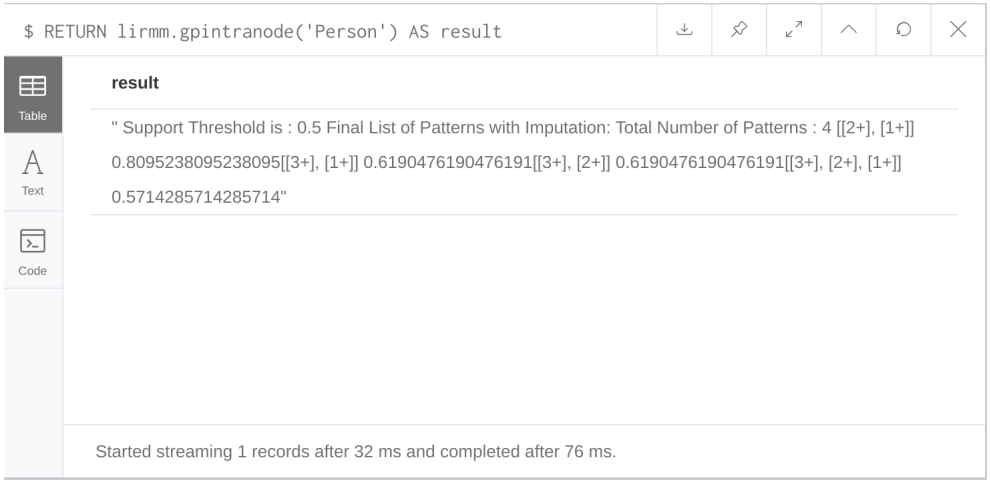

Figure 4.6 – *Neo4j: run of Intra-Label-Node-Properties GP with on a label*

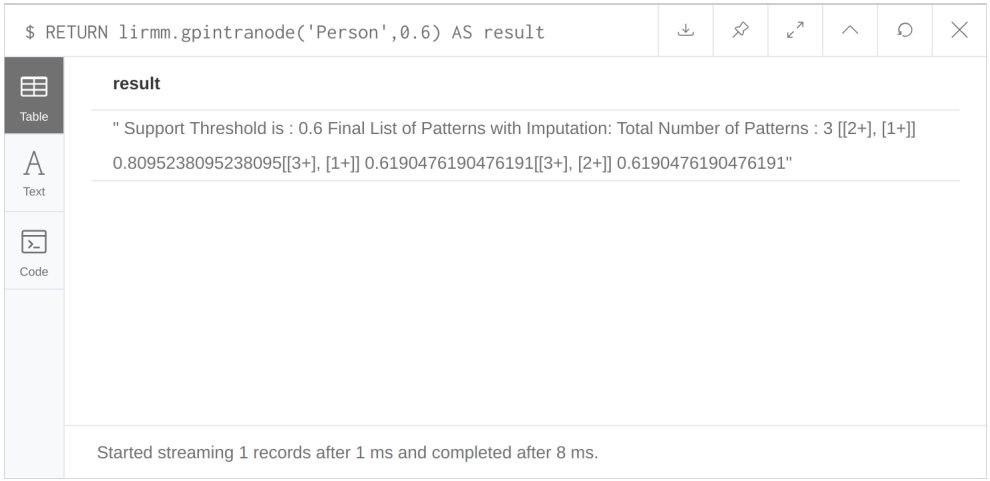

Figure 4.7 – *Neo4j: run of Intra-Label-Node-Properties GP with on a label and minsup*

Listing 4.1 – *Invoking Intra-Label-Node-Properties GP with on a label*

```
RETURN lirmm.gpintranode('Person', 0.7) AS result
```

Listing 4.2 – *Invoking Intra-Label-Node-Properties GP with label and minsup*

```
RETURN lirmm.gpintranode('Person', 0.7, ['desig']) AS result
```

Listing 4.3 – *Invoking Intra-Label-Node-Properties GP with label minsup and skipProperties*

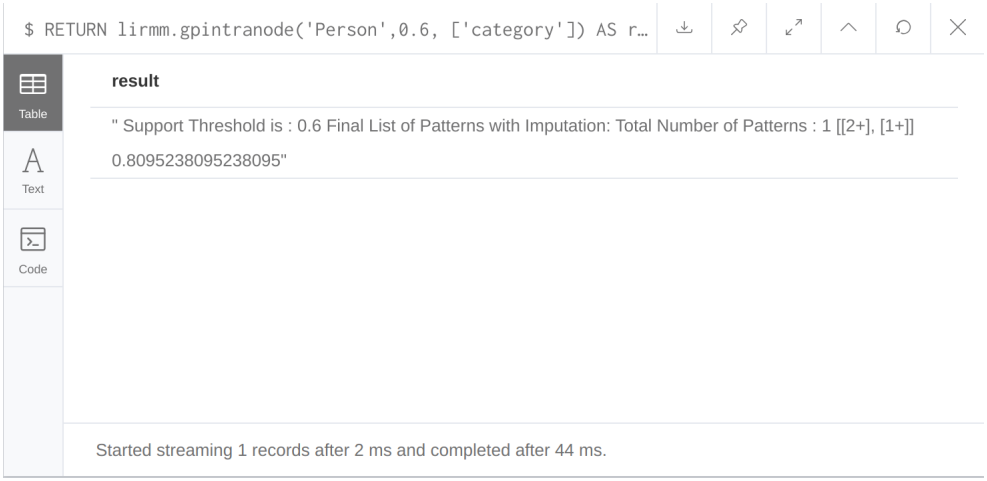

Figure 4.8 – *Neo4j: run of Intra-Label-Node-Properties GP with on a label and minsup + skipProperties*

4.4.5 Limits of the Current Integration

At the beginning of this section 4.4, we explain that this implementation is a proof-of-concept. Indeed, our goal here was not to make an optimized implementation of our entire work but only to lead the way for further works as it is explained in the perspectives Section 6.2. That is why, the current implementation is lacking from some limitations such as:

- We do not have done any optimization on memory consumption, cache system, or storage. We are streaming data from the graph database without any further optimization.
- we have only integrated the first of five scenarios defined in Section 4.4.3 but this is enough to demonstrate the feasibility of integrated our work into a graph database system.

Chapter 5

Experiments & Results

5.1 Introduction

In this chapter we present our experiments that we have performed on real datasets as well as on synthetic graphs using a graph generator. We show the results in terms of time utilization, memory consumption and the number of patterns being generated. Furthermore, we present discussion on results for each data and identify the issues for future investigations and results optimizations.

Before presenting the results, we first describe the synthetic graph generator used to conduct the experiments and then introduce the execution environment, experimental protocol, and datasets respectively. Finally, we show and experimental results and discussions.

5.2 Synthetic Graph Data Generation

The use graphs databases has widespread in industries and academia in recent past to represent the connected data with its applications in many different environments like social network analysis, fraud detection, industrial management, knowledge analysis, etc. [Fra+18, LMD14]. *More and more companies provide services that can not be anymore achieved efficiently using relational databases* [JV13]. The representation of real world data as graph brings a lot of opportunities for data analytics, but unfortunately most of big data graph databases are propriety systems as discussed in Section 2.2. Also, these graph based systems are in phase of evolution process, hence very limited number graph databases are publicly available in repositories for research purpose. This arises the need for development of tools that can facilitate to generate graph data to create graph in open source database engines like Neo4j.

In our urge to perform experiments for mining gradual pattern from property graphs, we extended an existing graph generator which was developed as an internship project. This web based tool is available at [Win18] and its GUI interface shown in Figure 5.1. The aim of project was to set up some parameters like the number of nodes and relationships he wants in the database and the type of distribution of relationships between nodes. Then, the generator produces the query corresponding to the parameters set by the user which are then used for graph creation. An example with description is shown in Section 5.2.3.

The existing tool [Win18] is tool is very useful to generate Cypher for creating nodes and relationships but it does not allow to add node properties and relationship properties. Therefore, an extension of the existing version is developed to generate synthetic graph *data* nodes and relationships. The extended version of the tool's web interface is available at [Sha19b]. The GUI interface of the extended version is shown in Figure 5.2. The extended section in encircled with red box. Angular platform is used to build this web application. *AngularJS is a JavaScript-based open-source front-end web framework maintained by Google* [Wik19]. The details about the source code and program execution are available in Appendix A.3.

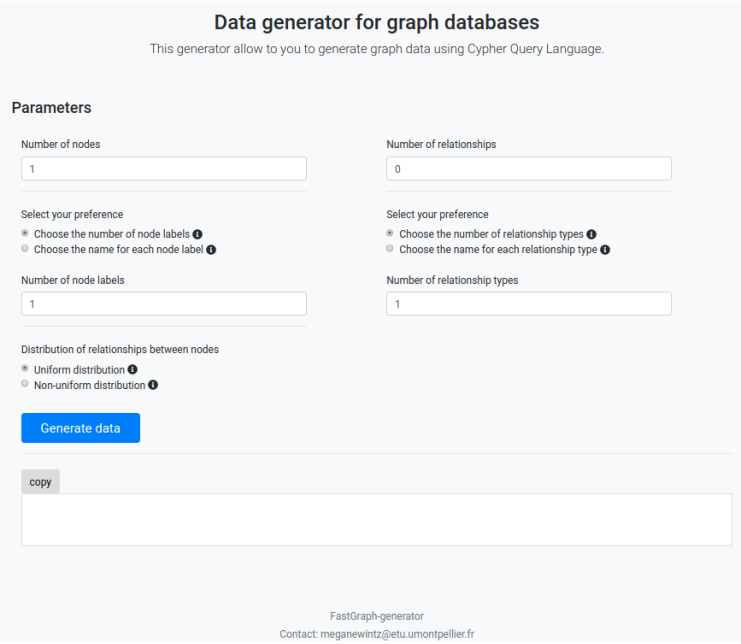

Figure 5.1 – *Graph Data Generation tool*

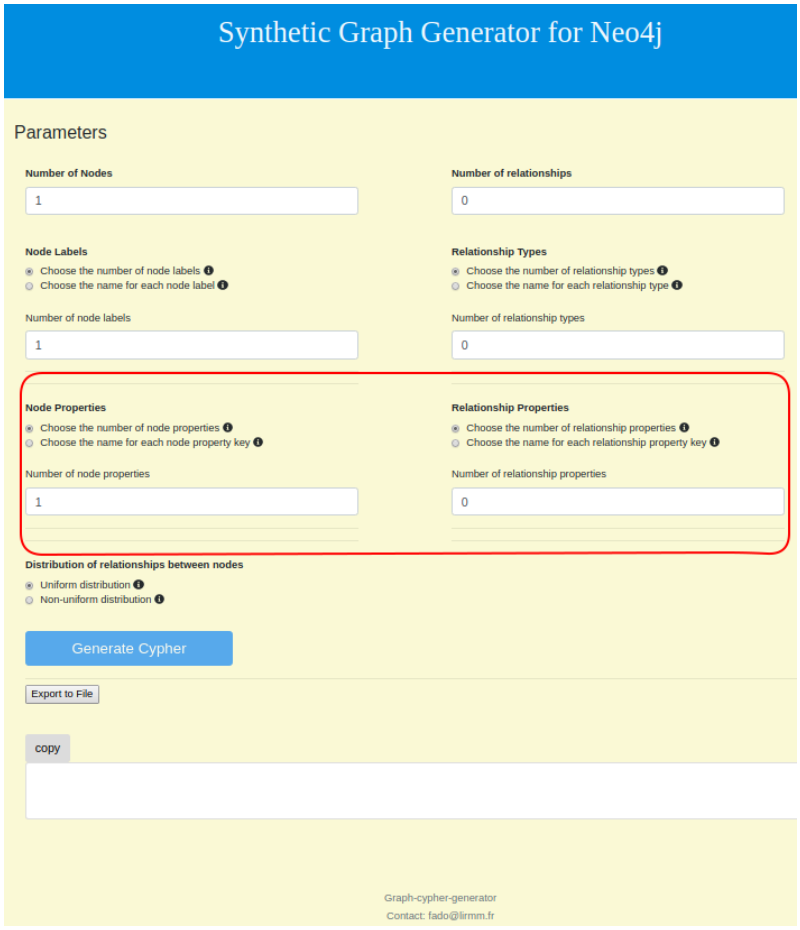

Figure 5.2 – *Synthetic Graph Generator for Neo4j*

5.2.1 Description of Data Generation Parameters

In our synthetic graph generation tool there are six fields showing various parameters for generating Cypher and two options for uniform and non-uniform distribution of relationships generation between the created nodes. The description of each of the field shown in Figure 5.3 is given as below.

1. **Number of Nodes**

   To define the number of nodes to be created in graph database. The minimum limit is set to 1 which is also defined as default "placeholder" as can be seen in Figure 5.2. There is no limit for the maximum number of nodes that can generated with this tool but user's machine specification are important[1].

2. **Number of Relationships**

   To define the number of relationships to be created in between nodes. The minimum limit is set to 0, because sometimes it not necessary to create relationships and user may only want to create nodes. There is no limit for the maximum number of relationships that can generated with this web based tool but user's machine specification must be under consideration[1].

3. **Node Labels**

   Node labels define the role of nodes in graph. There are two options for user to create names for the node labels. First is to assign randomly generated words and the second is to assign explicit names. As it can be seen in Figure 5.3, we assigned explicit label name as "Person" for the nodes. The output Cypher generated is shown in Figure 5.3. An example for assigning random names for labels is also created as shown in Figure 5.4 where two (02) labels namely ('fibeg' and 'gyhoby') and 10 nodes from 'n1' to 'n10' are shown with the generated Cypher.

   To make label field as mandatory, the field is set with alert message with "The number of labels should be at least 1". For the explicit names of label, label deletion is also feature is also configured using "chipsContainer" of javascript to customize the inputs.

4. **Relationship Types**

   A relationship is a directed edge that connects two nodes. Relationships in property graphs are considered as much important as nodes. Relationships names are called *"type"*. Similar to the node labels, relationship types can also be assigned explicitly

1. It is pertinent to mentions that AngularJS application execution is basically client-side (user's browser) therefore, the number of nodes that can be generated in web browser are mainly dependent on the user's machine specifications which is being used to generate nodes.

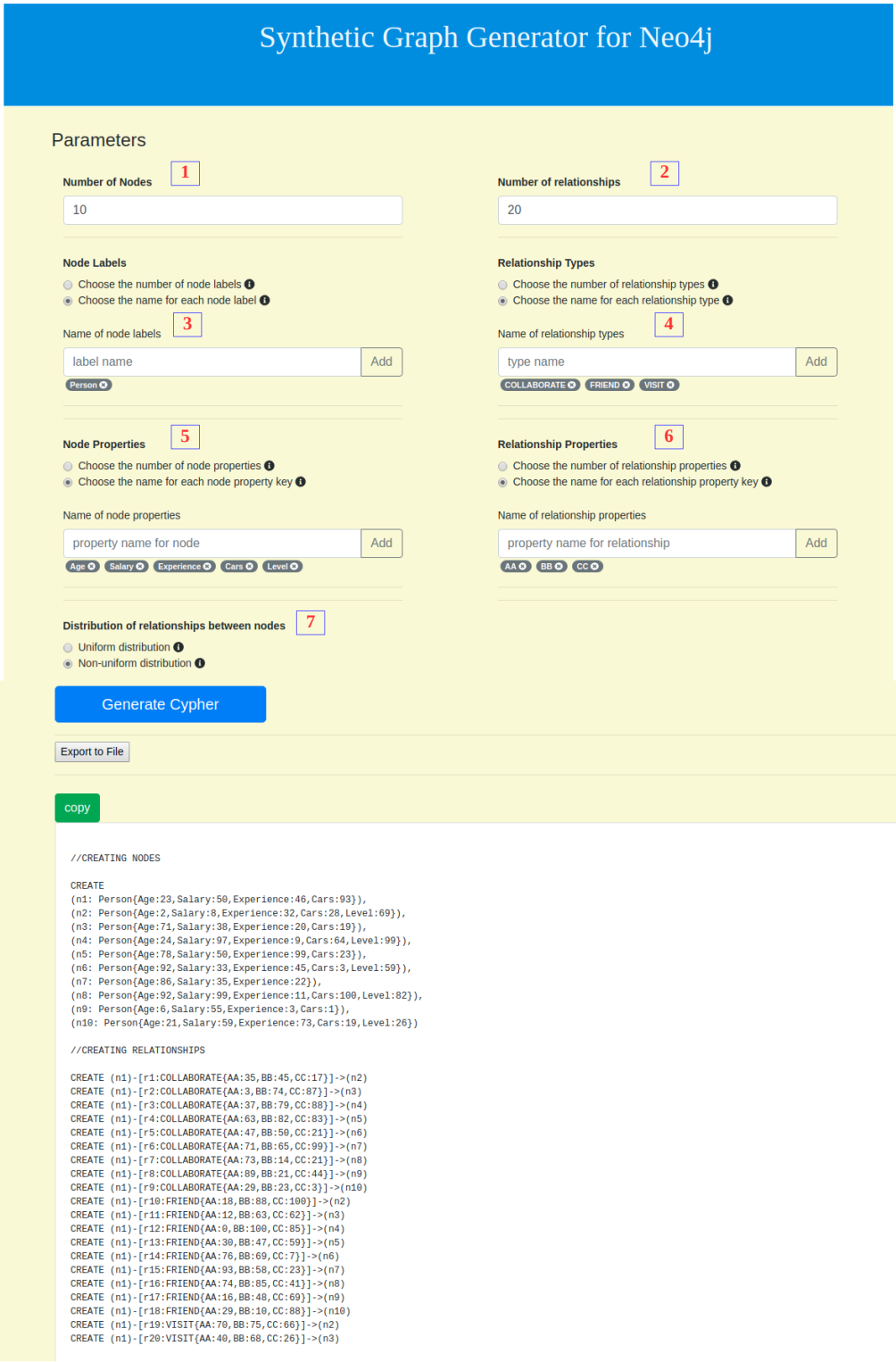

Figure 5.3 – *Graph Tool Parameters Descriptions*

or can be given a number to generate respective number of relationships. Figure 5.3 shows three (03) relationships types namely ('COLLABORATE', 'FRIEND', 'VISIT'). Figure 5.4 shows the randomly generated words for three (03) relationship types namely ('bicub', 'cihi', 'hubygy') with 20 relationships links from 'r1' to 'r20'.

5. **Node Properties**

   Node properties is the new feature added in the extended version of synthetic graph generator [Sha19b]. It generates the properties for the nodes in *(key:value)* format. Essentially, there is no maximum limit for the number of properties that can be assigned to nodes but it is recommended to give realistic number of properties to be generated. Two graph generation examples with explicit node labels and properties are shown in Figure 5.3 (for explicit property names) and Figure 5.4 (for random property name words) respectively. In the first example, there are five (05) explicit node property keys defined for which random values will be generated for each node in order to create property graph. These properties are namely (Age, Salary, Experience, Cars, Level). As it is visible in Figure that some properties are missing for some nodes. This is discussed more in Section 5.2.2.

   Tooltip information is also configured to make the parameters more informative and interactive for users. It is implemented using Popover with the trigger event of hover effect. The two information messages for numbers and names respectively are "You can configure only the number of node properties you want" and "You can choose the names that you want for node properties to be treated as keys". For the sake visibility of this description, second message is also shown in Figure 5.3.

   Two more important functionalities added in this field are:
   (i) To generate the a random value for each property key
   (ii) To introduce missing values randomly for key values

   The details about both of these functionalities are discussed in Section 5.2.2.

6. **Relationship Properties Parameters**

   Relationships in property graphs are treated as first class citizens and often contain properties in *(key:value)* format. This field is also a new addition in this extended version of graph generator. To make a realistic number of relationship properties we set the maximum limit of properties to 10 for each relationship type. This is not a hard limit and can be changed based on the experiment requirements if and when needed. Figure 5.3 and Figure 5.4 respectively shows the explicit and random names for relationships types. We would also like to mention that missing values feature in the properties of relationship types is not set. This can also be set whenever required based on the requirement of experiments.

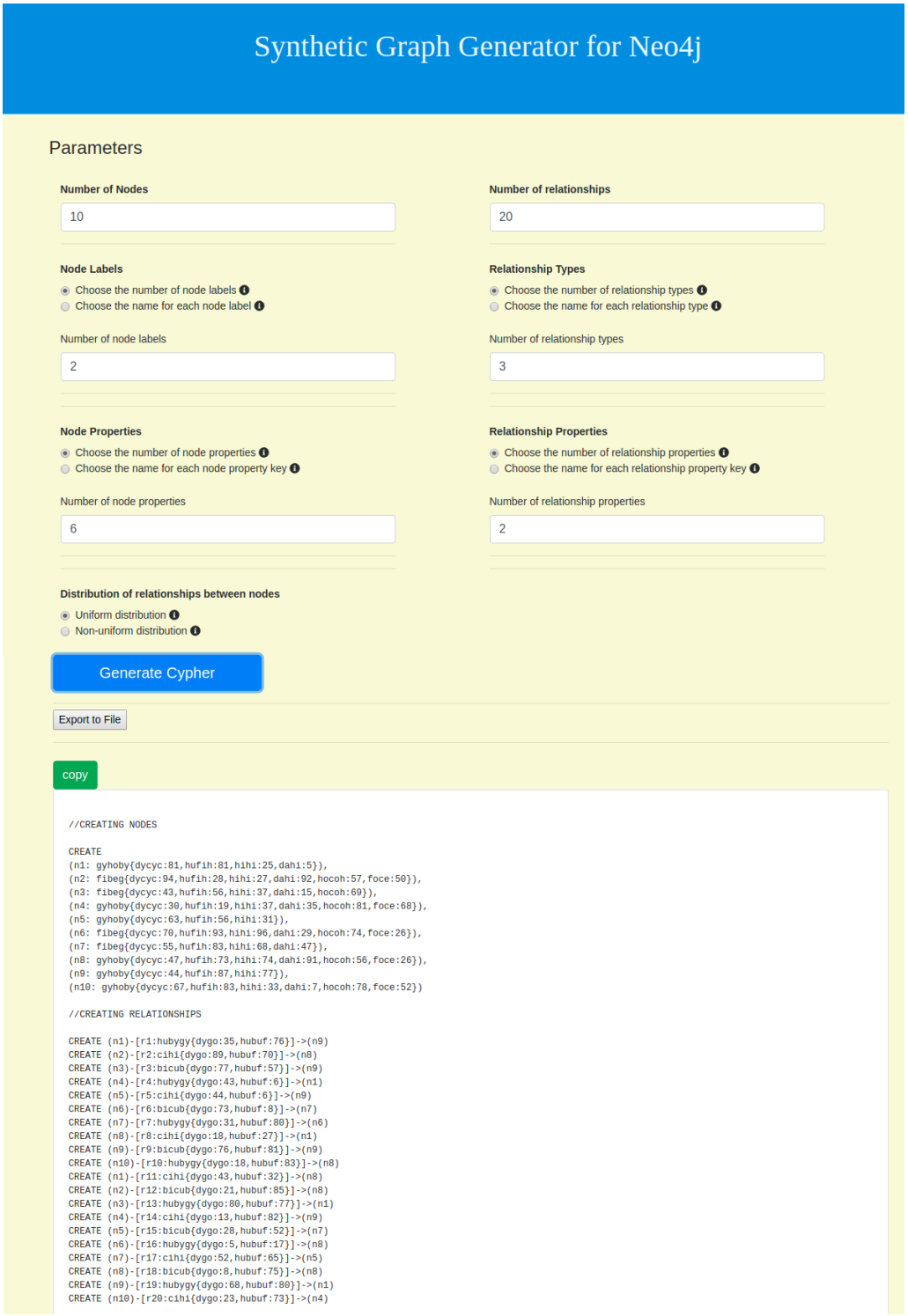

Figure 5.4 – *Cypher with Random Names of Labels, Relationships and Properties*

7. **Distribution of Relationships between Nodes**

To generation relationships between nodes for the defined number of nodes two

features are provided as shown in 5.1. In uniform distribution, the relationships are divided evenly between the total defined number of nodes whereas, in non-uniform distribution, some nodes have much more relationships than other. This feature is added to introduce a realist representation of the data that can resemble with real life data, it may be the case that some nodes have more relationships than other. The graph visualization for both of these features is shown in Section 5.2.3

5.2.2 Code Customization for Properties

In this section, we briefly describe the code listing with some explanation for the code customization performed to add the two fields namely "Node Properties" and "Relationship Properties". We specifically discuss random values generation for each property key and missing properties. To perform further enhancements and improvement in the existing code, the complete source is made available at git [Sha18].

- **FormControl validator:** Field alerts for maximum and minium values for number of node properties and relationship properties are defined using the function as shown in Listing 5.1. This is place where we can manage the min. and max. value of properties and can be increased or decrease as per given requirements.

```
1  //————————————FOR NODE PROPERTIES————————————————
2  node_property_radio: new FormControl('option_number'),
3  node_property_number: new FormControl(1,
4  [ Validators.required,
5  Validators.min(1),  // Minimum required properties
6  Validators.max(100)]
7  ),
8  //————————————FOR RELATIONSHIP PROPERTIES————————————————
9  relationship_property_radio: new FormControl('option_number'),
10 relationship_property_number: new FormControl(0,
11 [
12 Validators.required,
13 Validators.min(0),  // Minimum required properties
14 Validators.max(10)]
15 )
```

Listing 5.1 – *FormControl Validators for Alerts Configuration*

- **Generate Cypher Button:** When clicked on the button, the onSubmit() method send the form to the service "dataGenerator". The functionality behind the "dataGenerator" related to node and relationships properties is given in shown in Listing 5.2.

Here the the check between the options of number of node properties and explicit name is performed. If user has opted for explicit names for the node (or relationship) properties, then the entered keys will be stored in array named "node.property_keys", which is defined in export class "DataParameters" in a separate typescript file named "data_parameters_model".

If the user has opted for number of node properties in parameters then the "getRandomWords()" function is called which take the input number of properties entered by user and generates the random word for property keys. The maximum length for the randomly generated word is set to 5 which can be increased or decreased as per requirements.

```
//————————FOR NODE PROPERTIES————————————

if (this.node_property_radio.value === 'option_name') {
this.dataParameters.node_property_keys = this.node_property_keys;
}
else {
this.dataParameters.node_property_keys =
this.getRandomWords(this.node_property_number.value, 5);
}

//————————————FOR RELATIONSHIP PROPERTIES————————————

if (this.relationship_property_radio.value === 'option_name') {
this.dataParameters.relationship_property_keys =
this.relationship_property_keys;
} else {
this.dataParameters.relationship_property_keys =
this.getRandomWords(this.relationship_property_number.value, 5);
}
```

Listing 5.2 – *onSubmit() method*

- **Function for Random Values Generation for Keys:** The functions shown in Listing 5.3 are used to random values for the keys(properties) of nodes and relationships.

```
//————————FOR NODE PROPERTIES————————————
getNodeProperties(nodeProperties): string[] {
const words: string[] = [];
for (var j = 0; j < nodeProperties.length; j++) {
words.push(nodeProperties[j]+ ':'+ this.getRandomNumber(0,100));
// THE RANGE OF GENERATED VALUES IS BETWEEN 0 TO 100
//AND CAN BE CHANGED TO ANY RANGE
```

```
}
return words;
}
//————————————FOR RELATIONSHIP PROPERTIES————————————
getRelationshipProperties(relationshipProperties): string[] {
const words: string[] = [];
for (var j = 0; j < relationshipProperties.length; j++) {
words.push(relationshipProperties[j] + ':'
+ this.getRandomNumber(0,100));
// THE RANGE OF GENERATED VALUES IS BETWEEN 0 TO 100
//AND CAN BE CHANGED TO ANY RANGE
}
return words;
}
```

Listing 5.3 – *FormControl Validators for Alerts Configuration*

- **Nodes with Missing Properties:** The functions shown in Listing 5.4 is used to generate missing properties randomly out of total number of properties mentioned in the parameters by the user. This can be seen in both Figure 5.3 and Figure 5.4 that some nodes has missing properties whereas, other have complete. This feature is added to introduce a realist representation of the data that can resemble with real life data for the nodes as much as possible. This function assumes that users has desired at least the 3 properties for each node, which is set the minimum value of "getRandomNumber()" function. This minimum value can be changed as per user requirement.

```
//————————FOR NODE PROPERTIES————————————
getNodePropertiesWithMissing(nodeProperties): string[] {
const words: string[] = [];
for (var j = 0;
j < (this.getRandomNumber(3, nodeProperties.length)); j++) {
words.push(nodeProperties[j] +':'+ this.getRandomNumber(1,100));
}
return words;
}
```

Listing 5.4 – *Function for Nodes with Missing Properties*

5.2.3 Property Graph Creation Using Graph Generator Cypher

To show the graph creation process we show node and relationships generation for the Cypher generated in Figure 5.3 and Figure 5.4. First we show the graph generated without

properties parameters for the first version of the tool [Win18]. The Figure 5.5 shows the given parameters and generated Cypher with the tool [Win18]. The property graphs created with the generated Cypher are shown in Figure 5.6 (for uniform relationships distribution) and Figure 5.7(for non-uniform relationships). It is clearly visible that generated nodes have no properties except the node ID as highlighted with blue circle. Similarly relationships also do not have any properties.

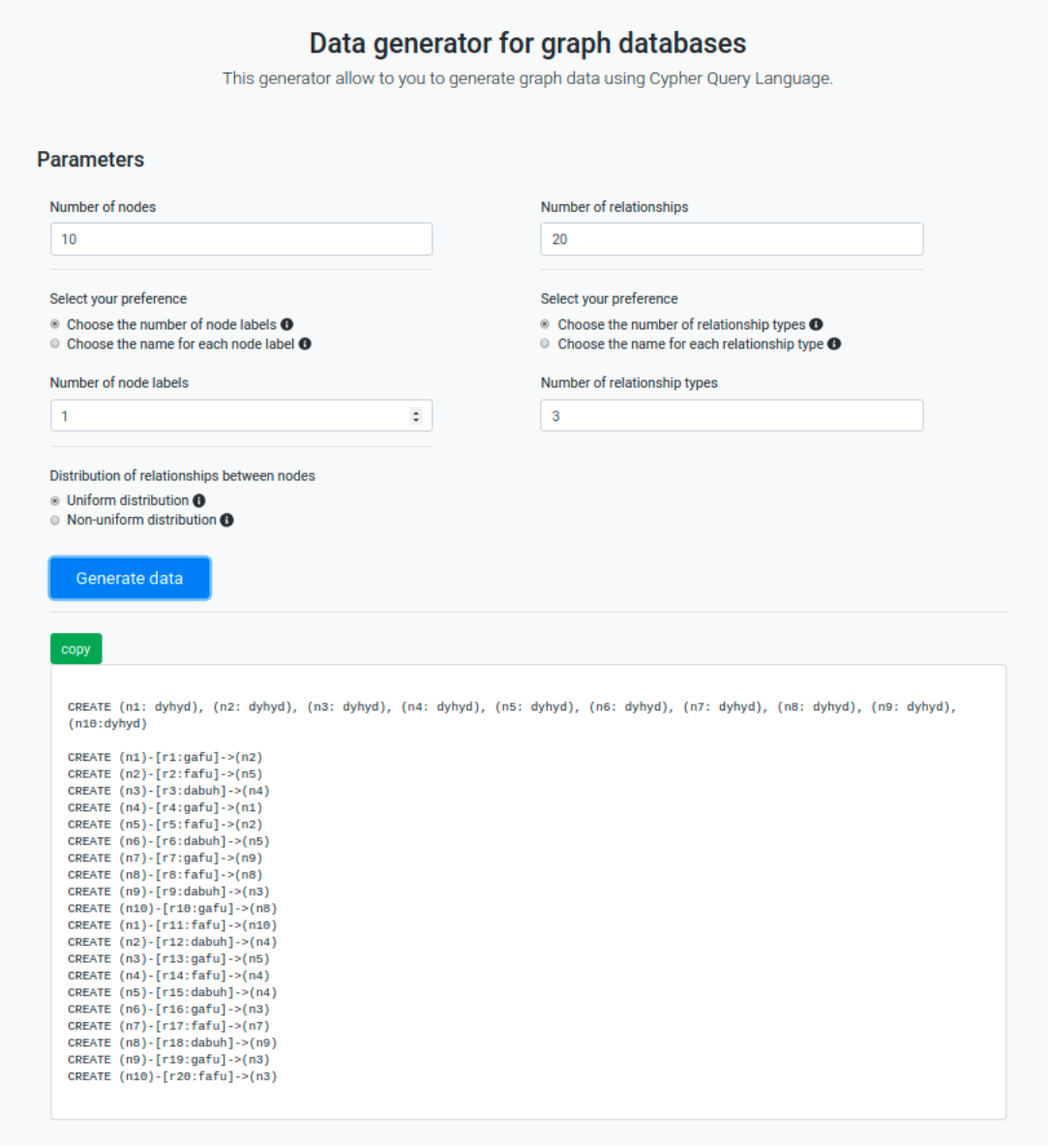

Figure 5.5 – *Original Graph Generator Tool with Cypher*

To create the property graphs with extended version of graph generator tool, we use the Cypher text generated in our examples as shown in Figure 5.3 and Figure 5.4. The parameters

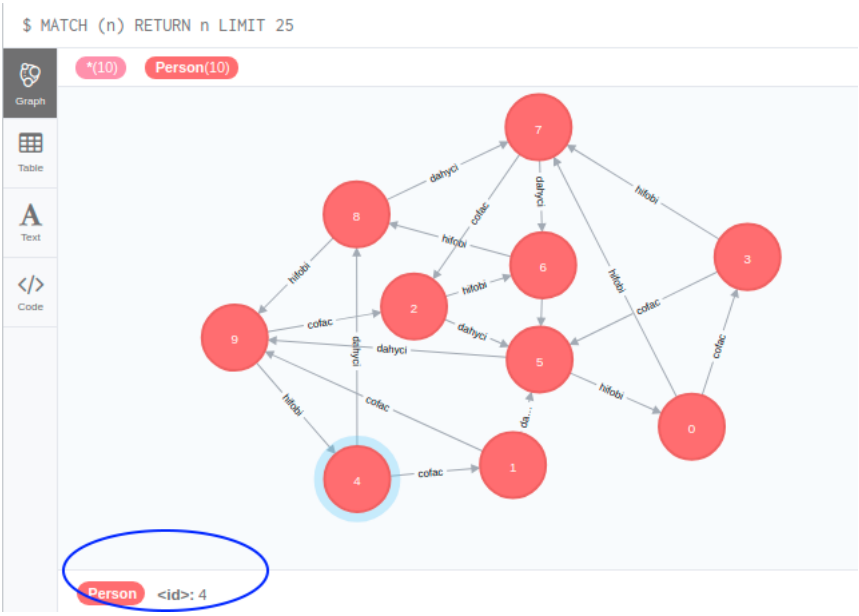

Figure 5.6 – *Uniform Nodes and Relationships "Without" Properties in Original Graph Generator*

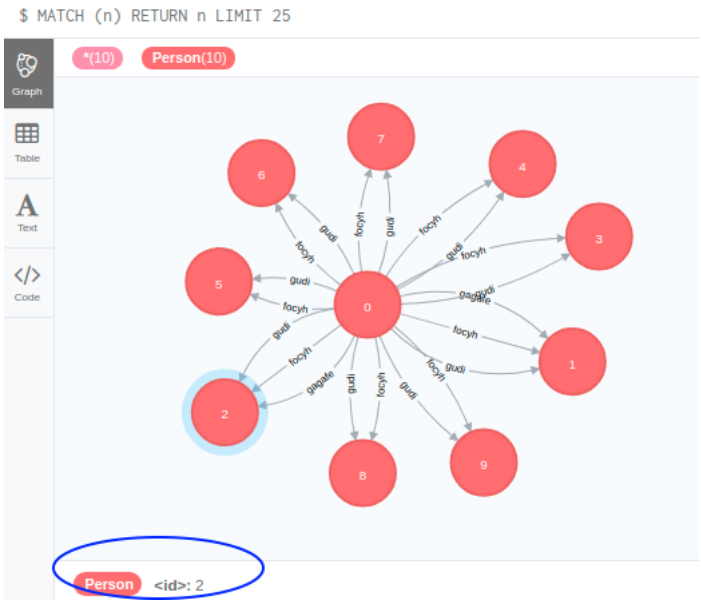

Figure 5.7 – *Non-uniform Nodes and Relationships "Without" Properties in Original Graph Generator*

description is discussed in Section 5.2.1 in detail and also visible in the figures as well. Non-uniform distribution is used for explicit label names and relationship types. For random label names and relationships types, uniform distribution is used. The output graphs are shown in Figures 5.8, Figures 5.9, Figures 5.10, Figures 5.11 respectively.

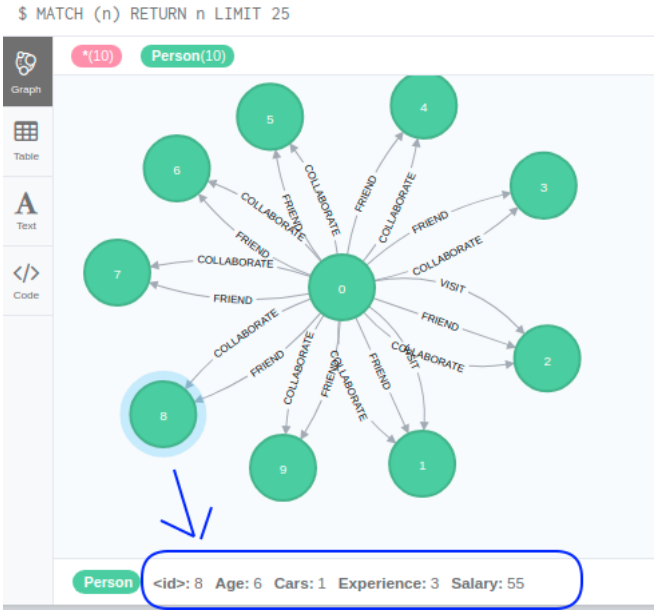

Figure 5.8 – *Explicit Label Names Nodes "With" Properties in Extended Graph Generator*

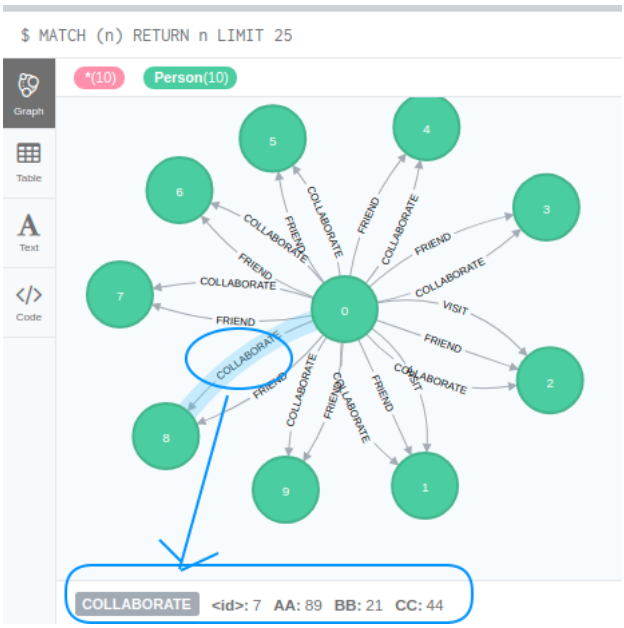

Figure 5.9 – *Explicit Relationship Names Nodes "With" Properties in Extended Graph Generator*

As we can see in Figure 5.8, the node ID 8 is selected in Neo4j GUI interface and the properties for this node are highlighted. Similarly in Figure 5.9, the "COLLABORATE" relationship between node ID 0 and node ID 8 is selected to hight the properties contained for this relationship. Similar depiction for node and relationships properties is shown in Figure 5.10 and Figure 5.11 respectively. For each of these property graph, each time a fresh graph database is

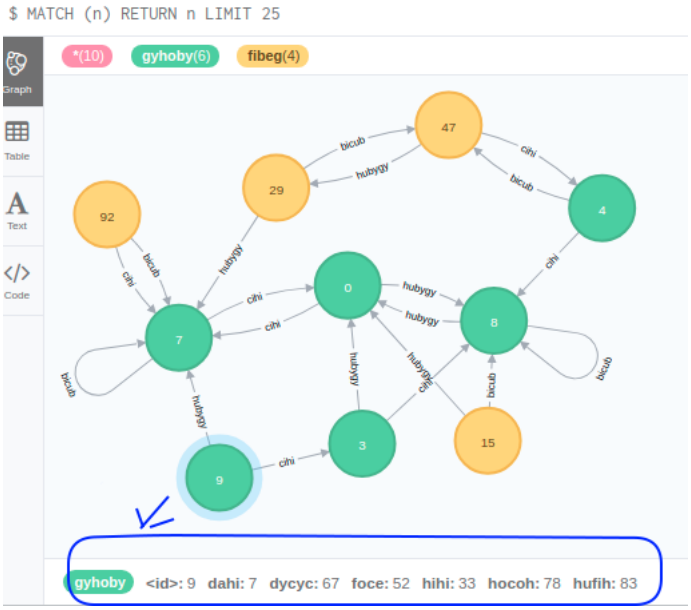

Figure 5.10 – *Random Label Names Nodes "With" Properties in Extended Graph Generator*

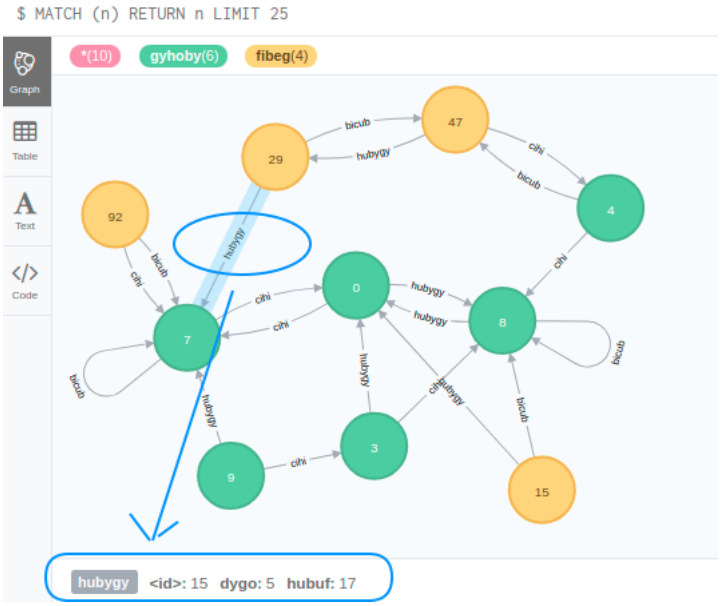

Figure 5.11 – *Random Relationship Names Nodes "With" Properties in Extended Graph Generator*

created to show these examples.

5.3 Program Setup Environment and Protocol

In this section we briefly describe the program execution environment, experimental protocol, outlier removal method and memory consumption metrics used to calculate maximum

heap utilization respectively.

5.3.1 Execution environment (hardware & software)

The execution environment describes the software and hardware on which these experiments are conducted. The details of tools and packages given below.

- Hardware: Intel Core i7-4610M, 3.00 GHz, quad core processor, 16 GB RAM
- Operating System: Linux generic kernel 4.4.0-134, Ubuntu 16.04 LTS
- Software:
  - JDK version "1.8.0_181", jre build 1.8.0_181-b13, HotSpot java 64-bit server VM (build 25.181-b13, mixed mode)
  - Neo4j graph database community edition 3.4.7, bolt protocol enabled
  - Neo4j-java-driver version 1.4.4

It may be noted that the program can safely run on the latest version of Java and Neo4j irrespective of the operating system. These configurations are just to describe our testing environment.

5.3.2 Experiments protocol

The experiment protocol describes the necessary steps that program takes from the database connection till the final list of gradual patterns. From these outputs, we plot the results shown in Section 5.5. The source code and output files are available at git [Sha19a].

1. The Java program establishes the connection with graph database using Neo4j's bolt protocol (port 7867) with the given credentials.
2. After the successful connection, the program queries graph database to retrieve the label's data for the required properties (attributes). In general, these are the user desired properties on which the gradual patterns extraction process is applied. This functionality is achieved by implementing HashMap. For our program, these properties must be numerical.
3. Once the properties with data are retrieved, they are stored in array list. Then for each property the binary matrix initialization is performed as discussed in Section 2.4
4. After the matrix initialization, and concordant object pairs AND operation, the *vdb* and support computation takes place as discussed in Section 4.1

5. The result is stored for size-1 gradual patterns (i.e., two items) and program iterate for next level and so on as explained in Algorithm 1 and Algorithm 2.

The program takes min_support threshold as input parameter. An executable jar file is created for the program and it is run 5 times. The average value is then used to plot the charts. In this process we also perform outliers removal as discussed below.

5.3.3 Outlier Removal Approach

The outlier removal method is adopted to eliminate the result which is not considered as the expected output. This non-corresponding result may have appeared due to any software, memory or hardware issues. The mechanism used to remove the outlier is as under. It is important to mention that this approach is adopted for results of time and memory because there is no change in the total number of gradual patterns that are generated in the result.

Each data point which is plotted in results (Section 5.5) is considered by taking the average of 5 times execution of the program with same set of input parameters. In case of any outlier result point in time or peak heap memory utilization, we use the Three Sigma rule also known as 68-95-97.7 or "empirical rule". It states that for a normal distribution almost all of the data will be within the range of 3 standard deviations (std.dev). Empirical rule has three parts i.e., 68% of data will fall with 1 standard deviation of the mean value, 95% of data within 2 standard deviations of the mean value and 97.7% of data within 3 standard deviations of the mean value. For removing the outliers, we adopt the rule of 2 standard deviations i.e., (Mean $+ 2 \times std.dev$) for upper bound point and (Mean $- 2 \times std.dev$) for lower bound point [Nar19].

5.3.4 Memory Consumption Metric

Choosing a metric for memory consumption is an essential task. A Java program consumes both heap and non-heap memory. The JVM considers memory pools of type heap and non-heap [ora18]. For the memory consumption plots we take peak used memory pools which are of type HEAP and NON_HEAP. For this we use java $MemoryPoolMXBean$ management interface for a memory pool and call the method $getPeakUsage()$. The plots for datasets show the sum of heap and non-heap peak memory usage.

For each program execution, we have used the same JVM settings. First of all, we have decided to use the G1 garbage Collector. First introduced with Java 7 this G1 garbage collector has the unique ability to efficiently and concurrently deal with very large heaps. It can also be configured to not exceed a maximum pause time. We also set the *BiasedLockingStartupDelay* to 0 for improving performance of unaccounted synchronization. We have set the Xmx parameter to fix the maximum heap size at 10G for memory allocation pool. To summarize, the jvm flags as arguments for program execution are:

S#	Dataset	Nodes	Properties	Missing %
1	Russian_tweet_Users	393	6	3
2	Hepatitis	155	6	13
3	Synthetic	10,000	5	25

Table 5.1 – *Intra-Node Gradual Pattern Mining*

-XX:+UseG1GC -XX:BiasedLockingStartupDelay = 0 -Xmx10G

5.4 Datasets

We have performed the experiments on two types of gradual pattern mining approaches explained in Section 3.3. These are:

- Intra-node gradual pattern mining (Section 3.3.1)
- Node properties with relationships count (Section 3.3.3)

The Table 5.1 shows the datasets for which intra-node gradual pattern mining experiments were performed and Table 5.2 shows the datasets for node properties with relationship count. It is pertinent to mention that program is applicable for numerical attributes.

For the intra-node datasets, since the Russian_tweet is a real dataset therefore, in order to apply gradual pattern mining, first the dataset need to imported into graph database. Similarly, for Hepatitis dataset which is taken from UCI Machine Learning repository needed to for imported first into graph. The data import method for these datasets is discussed later in this section. Also, in case of these both datasets, a pre-processing step is performed to opt the attributes as properties which are numerical due to our program requirement of processing only such kind of data. The synthetic numerical data generated using legacy data generator [Sha19a] was also imported into graph database in order to apply gradual pattern mining. For all these, we have created separate graph databases and run the program as per experimental protocol described in Section 5.3.2. The more details for each of these datasets is presented later in this section.

For the nodes properties with relationships count datasets Table 5.2, we do not require an import because synthetic graph generator generates the *Cypher* which can be directly executed in cypher-shell to create nodes and relationships whereas Hetionet dataset is a real graph database which is publicly available at [Him19].

S#	Dataset	Nodes	Relationships	Properties	Relationship Types	Missing %
1	Synthetic Graph	15,000	999,268	6	3	10
2	Hetionet(Gene)	20,945	1,289,190	1	4	1

Table 5.2 – *Nodes with Relationships Gradual Pattern Mining*

5.4.1 The Russian Twitter Troll

The Russian_tweet_troll is a real dataset taken from [New19]. This dataset contains more than $232,061$ tweets and 453 that Twitter has tied to "malicious activity" from Russia-linked accounts during the 2016 U.S. presidential election. The dataset is publicly available as Neo4j sandbox use case cite and the graph database can be accessed through direct URL as shown in "Details" section of Figure 5.12. The graph schema visualization is shown in Figure 5.12.

The overall summary of graph schema showing the relationships count between source and destination labels is shown in Listing 5.5.

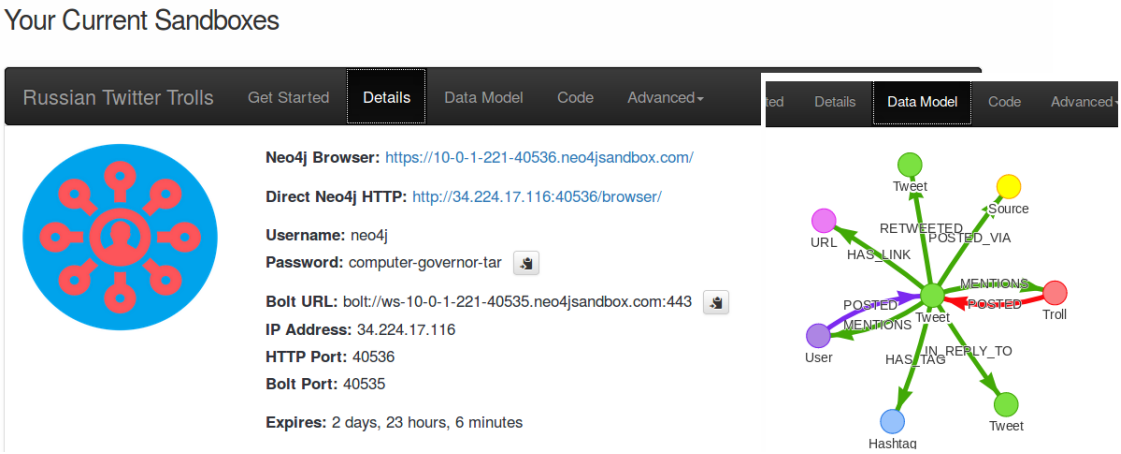

Figure 5.12 – *Russian-Twitter-Troll Sandbox Details*

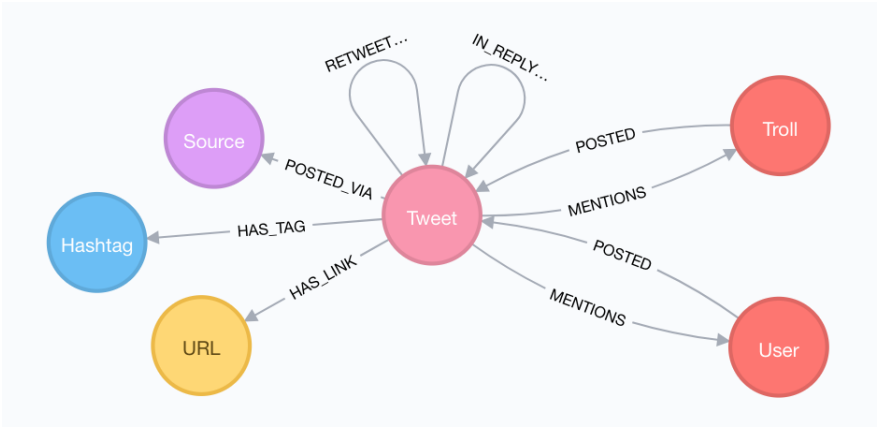

Figure 5.13 – *Russian-Twitter-Troll-Graph Schema Visualization*

```
$ ./bin/cypher-shell -a bolt://34.224.17.116:40535 -u neo4j -p computer-governor-tar
Connected to Neo4j 3.4.11 at bolt://34.224.17.116:40535 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j> MATCH (n) -[r]-> (m)
RETURN labels(n) AS SrcLabel , type(r) AS Relationship,
labels(m) AS DstLabel, Count (*) AS Relationships_Count;
+-----------------------------------------------------------------------------------+
| SrcLabel          | Relationship   | DstLabel          | Relationships_Count |
+-----------------------------------------------------------------------------------+
| ["Tweet"]         | "IN_REPLY_TO"  | ["Tweet"]         | 559                 |
| ["Tweet"]         | "MENTIONS"     | ["User", "Troll"] | 956                 |
| ["Tweet"]         | "POSTED_VIA"   | ["Source"]        | 58051               |
| ["Tweet"]         | "HAS_TAG"      | ["Hashtag"]       | 106792              |
| ["Tweet"]         | "MENTIONS"     | ["User"]          | 55154               |
| ["User", "Troll"] | "POSTED"       | ["Tweet"]         | 200833              |
| ["Tweet"]         | "HAS_LINK"     | ["URL"]           | 31166               |
| ["Tweet"]         | "RETWEETED"    | ["Tweet"]         | 39649               |
+-----------------------------------------------------------------------------------+

8 rows available after 853 ms, consumed after another 0 ms
neo4j>
```

Listing 5.5 – *Russian Tweet Graph Schema Summary with Relationships*

From the available data of user accounts file [New19] we selected 7 relevant (numerical) attributes out of 14 for our experiments. These are (userId, followersCount, statusesCount, timeZone, favouritesCount, friendsCount, and listedCount). This data is used to create graph database using cypher import utility. The data is imported using the Cypher commands shown in Listing 5.6

```
LOAD CSV WITH HEADERS FROM
"file:///users_russian_tweet_troll.csv" AS row
MERGE
(:User {userId:row.userId,
followersCount:row.followersCount,
statusesCount:row.statusesCount,
timeZone:row.timeZone,
favouritesCount:row.favouritesCount,
friendsCount:row.friendsCount,
listedCount:row.listedCount})
```

Listing 5.6 – *Cypher import command for Russian-tweet-troll*

5.4.2 Hepatitis from UCI Machine Learning Repository

Hepatitis dataset is taken from UCI machine learning repository [Rep19]. For this dataset, there are 20 attributes, but we consider only 6 relevant attributes (BILIRUBIN, ALKphosphate, PROTIME, ALBUMIN, Age, and SGOT]) that have been imported to graph database. Attributes selection is made considering the numerical attributes requirement of program. Listing 5.7 shows the command to import this dataset in Neo4j.

```
1 LOAD CSV WITH HEADERS FROM
2 "file:///hepatitis.csv" AS row
3 MERGE
4 (:Hepatitis {Age:row.Age,
5 BILIRUBIN:row.BILIRUBIN,
6 ALKphosphate:row.ALKphosphate,
7 SGOT:row.SGOT,
8 ALBUMIN:row.ALBUMIN,
9 PROTIME:row.PROTIME})
```

Listing 5.7 – *Cypher import command for Hypatitis dataset*

5.4.3 Synthetic Dataset

For this experiment, we creates a synthetic dataset by using the Java class "Random" to generate a stream of pseudorandom numbers. The source code for generating dataset is available at gitlab [Sha19a]. We generated the dataset for $10,000$ nodes and 5 attributes. Now, in order to introduce missing values of 25% in the generated file, we use R's package $missForest$ and its method $prodNA$. This package uses $Random\ Forest$ supervised machine learning algorithm to introduce missing values completely at random (MCAR). The database creation command is show in Listing 5.8.

```
1 LOAD CSV WITH HEADERS FROM
2 "file:///genData10K.csv" AS row
3 MERGE
4 (:Synthetic {Id:row.Id,
5 A99RO:row.A99RO,
6 B99RO:row.B99RO,
7 C99RO:row.C99RO,
8 D99RO:row.D99RO,
9 E99RO:row.E99RO})
```

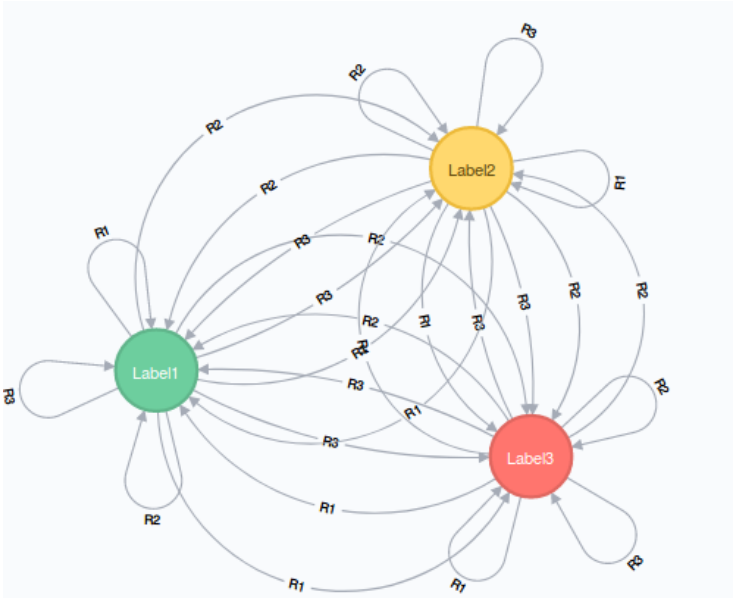

Figure 5.14 – *Synthetic Graph Schema visualizations*

Listing 5.8 – *Cypher import command for Synthetic dataset*

5.4.4 Synthetic Graph Dataset using Graph Generator

We have developed a synthetic graph generator to make property graphs in Neo4j database. This graph generator's web interface is available at [Sha19b]. The synthetic graph dataset contains 3 labels namely Label1, Label2, Label3 and 3 relationship types R1, R2, and R3 respectively. The nodes and relationships details are shown in Table 5.2. We created the graph database with these specifications to run the experiments for both vdb and imputation methods to compare the time and memory utilization as well as to observe the difference in generated patterns. The graph schema visualization for this property graph is shown in Figure . The created database can be downloaded from [Sha19a] to use in Neo4j and the complete source code is available at [Sha19a].

5.4.5 Hetnets in Biomedicine

Hetnet is a research outcome published in [Him+17], which shows a network of biology, disease, and pharmacology. It is an integrated network of nodes and relationships in which "the knowledge from millions of biomedical studies over the last half century have been encoded into this single system" [Him+17]. The graphs schema of Hetionet is shown in Figure 5.15. *Panel A shows the metagraph (graph of types) for Hetionet. Panel B shows the actual hetnet with*

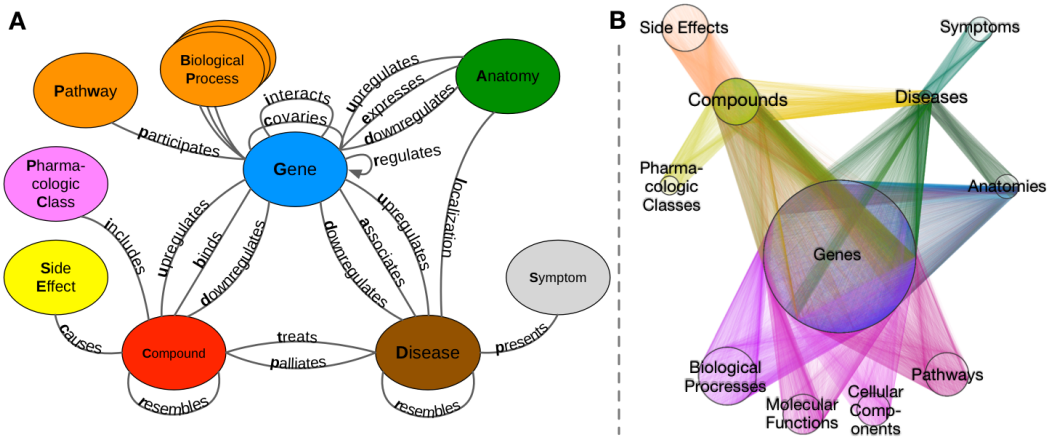

Figure 5.15 – *Hetnet Graph Schema visualizations [Dan19]*

Src	Relationship	Dst	RelCnt
Gene	PARTICIPATES_GpBP	Biological Process	559504
Gene	REGULATES_GrG	Gene	265672
Gene	INTERACTS_GiG	Gene	147164
Gene	PARTICIPATES_GpMF	Molecular Function	97222
Gene	PARTICIPATES_GpPW	Pathway	84372
Gene	PARTICIPATES_GpCC	Cellular Component	73566
Gene	COVARIES_GcG	Gene	61690

Table 5.3 – *Hetionet "Gene"Relationships Summary*

nodes laid out in circles by type. Color denotes edge type[Dan19]. The complete details of project and its relevant data is available on GitHib [Dan19] and its GUI interface is available at [Him19].

In this dataset, we choose the "Gene" label and its relationships. The purpose to choose "Gene" because it has a relevant attribute i.,e number of chromosomes for every gene. We took all the gene nodes and their relationships with other labels like chemical compound, biological process, etc. There are 7 relationship types of "Gene" Label in Hetionet. The summary of its relationships with other labels is shown in Table 5.3. The dataset has a total of 47,031 nodes and 2,250,197 relationships. In the Gene label, there are 20,945 nodes and 1,289,190 relationships with 7 relationship types. For our experiments, we tool all the nodes with 4 relationship type that connect with labels other than Gene itself like, PARTICIPATES_GpBP, PARTICIPATES_GpMF, PARTICIPATES_GpPW and PARTICIPATES_GpCC. The output plots for the time utilization, memory consumptions and number of patterns are shown in in Figure 5.28, Figure 5.29, and Figure 5.30 respectively.

S.No.	Nodes	Labels	Properties	Relationship Types	Missing%
1	1K	3	9	3	17
2	10K	3	9	3	17
3	20K	3	9	3	17

Table 5.4 – *Synthetic Dataset: Nodes Details*

Node Properties									Relationships Count		
A1	A2	A3	B1	B2	B3	C1	C2	C3	R1	R2	R3
25	..	..	43	..	..	..	..	74	1	0	2
45	..	..	88	..	..	..	..	NULL	2	2	1
69	..	..	NULL	..	..	..	..	56	1	0	1

Table 5.5 – *Graph structure (Node Properties with Relationship Count)*

5.4.6 Datasets for Fuzzy and Crisp Gradual Patterns

To generate graph data to mine fuzzy gradual patterns, we used synthetic graph generator [Sha19b]. The graphs were generated for three datasets of 1000, 10,000 and 20,000 nodes and same number of relationships for the respective datasets. The details of the datasets is shown in Table 5.4. The range of values of the node properties are randomly generated between 1 to 100. Also, due to the schema less nature of property graphs, we introduce missing data by randomly selecting the attributes of nodes. The resulting missingness of data is 17% for all the 3 datasets as shown in Table 5.4.

For a particular label e.g., "Label 1", the graph structure of node attributes with relationships count is shown in Table 5.5. This graph structure contains 12 attributes i.e., 9 properties for each node as (A1, A2, A3, B1,..,C3) and 3 attributes for relationship count as (R1, R2, R3). If there does not exist any relationship for a node, the graph database returns 0 in response to the graph structure Cypher query. If there exist a relationship for a node, it returns the count of relationship. For example, the first node in Table 5.5 having relation type R2 as 0 means that it does not have any relationship type R2 where there are 2 relationships of type R3 for the same node.

5.5 Results

In this section we show the plots for various types of datasets for intra-node and node properties with relationships count as discussed in Section 5.4. For each dataset, two types of approaches (i.e., *vdb* and imputation) are compared in terms of time and peak memory consumption with respect to minimum support threshold. Furthermore, we also show the number of patters generated for each of the dataset.

For the plots of missing data imputation method, we made a separate Java program to perform comparisons of time consumption, peak memory and number of generated patterns. The missing data is imputed using R package *Amelia* 2 that uses the expectation minimization algorithm [HKB11] for multiple imputation of missing data. The complete details of source code is available at GitLab [Sha19a] and the program flow for the imputation and non-imputation i.e. *vdb* method is shown in Appendix A.1. The results of experiments are shown in two subsection as below.

5.5.1 Intra-Node Datasets Plots

In case of Russian tweet troll dataset (Section 5.4.1), we took the user accounts data to apply the gradual patterns extraction and plot the results for intra-node scenario. As we can see for the time utilization in Figure 5.16 and Memory consumption in Figure 5.17 that vdb approach performs better that imputation approach. Since, the number of nodes are small as well as the percentage of missing data, the number to generated pattern for both types of approaches (i.e., *vdb* and imputation) are almost same as shown in Figure 5.18.

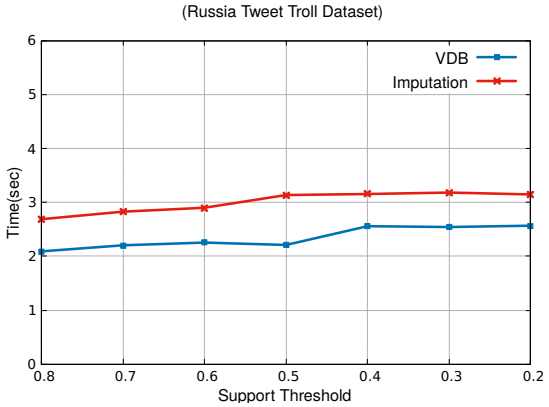

Figure 5.16 – *Time Utilization for Russian Troll Tweets*

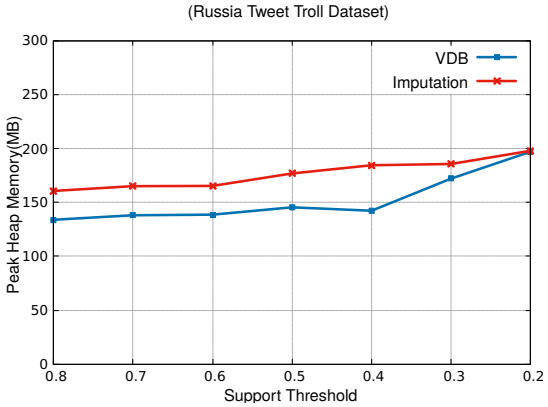

Figure 5.17 – *Memory Consumption for Russian Troll Tweets*

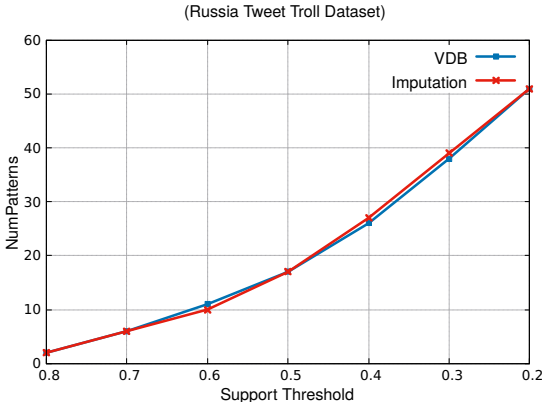

Figure 5.18 – *Number of Patterns for Russian Troll Tweets*

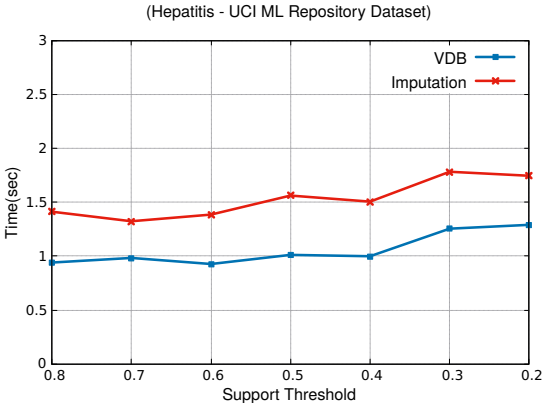

Figure 5.19 – *Time Utilization for Hepatitis - UCI ML Repository Dataset*

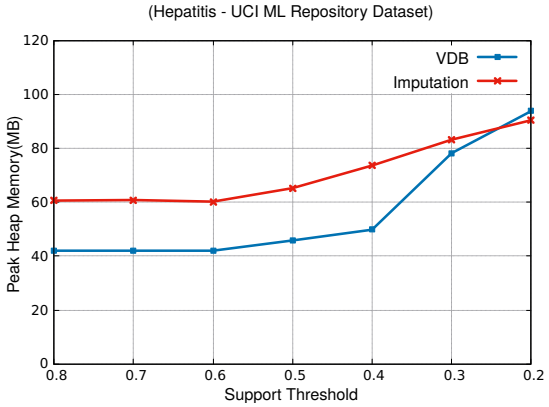

Figure 5.20 – *Memory Consumption for Hepatitis - UCI ML Repository Dataset*

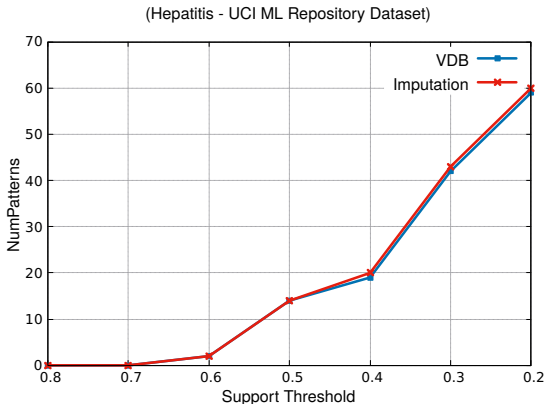

Figure 5.21 – *Number of Patterns for Hepatitis - UCI ML Repository Dataset*

For Hepatitis dataset (Section 5.4.2) we observed some excessive patterns in imputation method. For example, in the output result at minimum support threshold of 0.2, some patterns were not present in VDB method output file for the same minimum support threshold such as (BILIRUBIN ↑, Age↑, SGOT↓), (ALKphosphate ↑, PROTIME↑, ALBUMIN↑) and (ALKphosphate↑ , PROTIME ↑, Age↓). We think that these excessive patterns may not resemble to the reality and can be better decided by the domain expert by looking at the actual available data. In this dataset we have 13% of missing values in the available data.

In terms of time utilization and memory consumption we can see in Figures 5.19 and 5.20, the vdb approach perform better than imputation, whereas the total numbers of patterns is almost same due to smaller volume of data. The complete output results of all these experiments are available at [Sha19a].

Similarly, in the case of synthetic dataset (Section 5.4.3) at support threshold of 0.2, the imputation approach generates 24 patterns whereas vdb approach generates 13 patterns. Some of the excessive pattern generated by imputation approach are (A99RO↑ , B99RO↑), (A99RO↑, C99RO↑), (B99RO↑, C99RO ↑), (A99RO ↑, B99RO↑, C99RO↑), and (B99RO ↑, D99RO↑, E99RO↑).

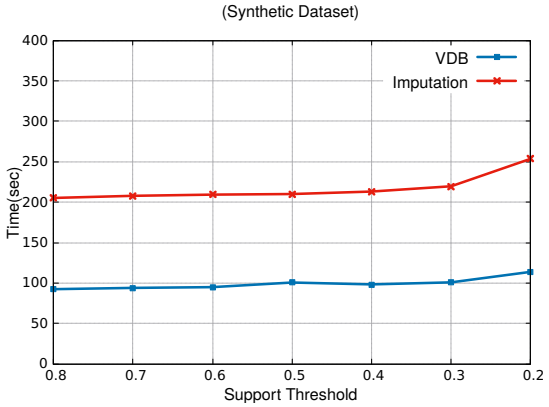

Figure 5.22 – *Time Utilization for Synthetic Dataset*

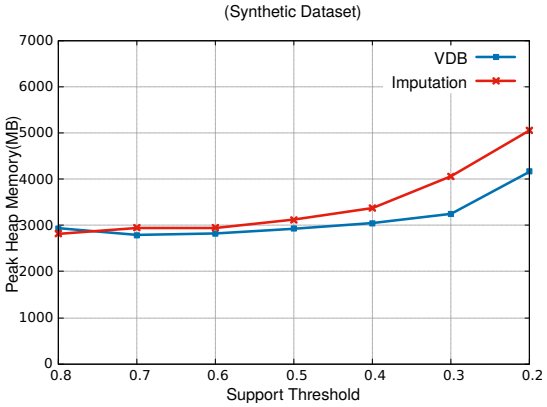

Figure 5.23 – *Memory Consumption for Synthetic Dataset*

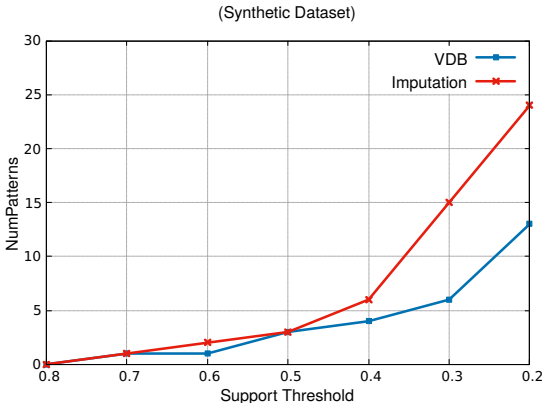

Figure 5.24 – *Number of Patterns for Synthetic Dataset*

As described earlier in dataset description Section 5.4.3, there is 25% missing data and we notice a reasonable difference from minimum support threshold of 0.4 to 0.5 in memory consumption and time utilization as shown in Figure 5.22 and 5.23.

5.5.2 Nodes with Relationships Count Datasets Plots

In case of synthetic graph data generator dataset (5.4.4), we can observe that in Figure 5.26 from minimum support threshold of 0.8 to 0.5 the memory consumption is almost same but at 0.4 the memory consumption becomes more than double for the imputation method from 2GB to almost 5GB and a very clear difference is observed at much lower minimum support of 0.2 with a difference of almost 9GB higher memory consumption. Interesting observation is notice particularly in number of patterns like at minimum support of 0.4 there is difference almost 250 excessive patterns and at 0.2 the difference becomes 600 patterns. This clearly indicates that these excessive patterns may or may not be useful but vdb approach generates only the patterns for actual data which may provide more clarity in decision making on the given data.

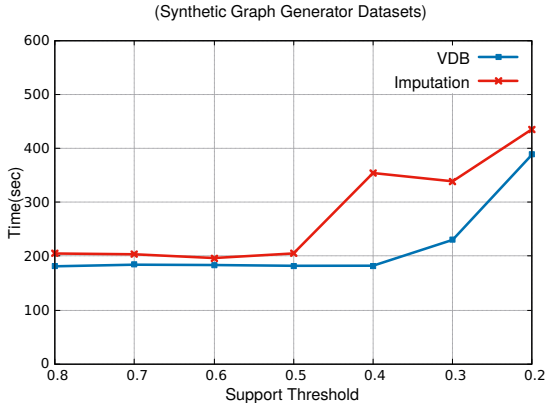

Figure 5.25 – *Time Utilization for Synthetic Graph Generator Dataset*

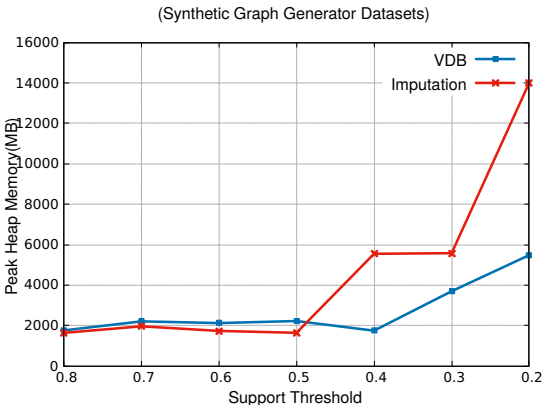

Figure 5.26 – *Memory Consumption for Synthetic Graph Generator Dataset*

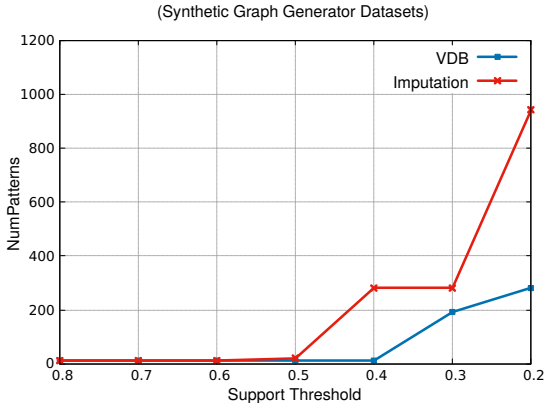

Figure 5.27 – *Number of Patterns for Synthetic Graph Generator Dataset*

For the plots of Hetnets in Biomedicine (Section 5.4.5) we download the complete graph database from its Github source [Dan19] in order to plot the result with same environment setup. Hetnets is a big database with more than 2 million relationships and more than 47 thousand nodes. For the sake of data relevance we choose "Gene" label for the experiments which contains more than 20 thousand nodes. The available data has only 1% missing values therefore, there is not distinguishable difference in number of patterns being generated between vdb and imputation methods. Time utilization for both approaches is almost same for all values of minimum support whereas, at smaller value such as 0.3 we observe that vdn method consumes less memory that imputation method.

5.5.3 Fuzzy and Crisp Datasets Plots

The plots for fuzzy sets and crisp data are shown and for three different size of datasets as shown in Table 5.4. The experimental results show that with fuzzy sets, the time consumption

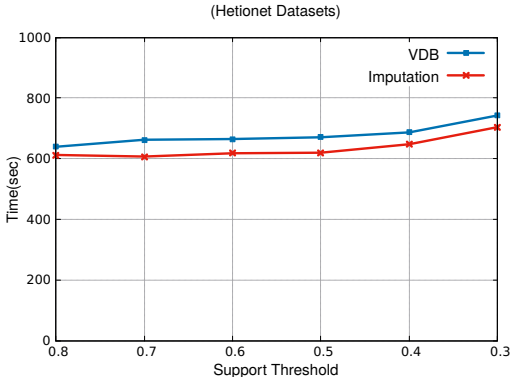

Figure 5.28 – *Time Utilization for Hetionet(Gene) Dataset Dataset*

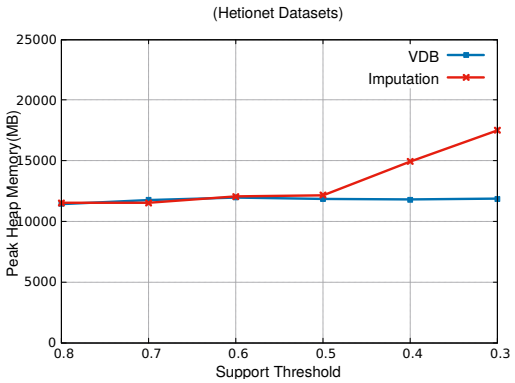

Figure 5.29 – *Memory Consumption for Hetionet(Gene) Dataset Dataset*

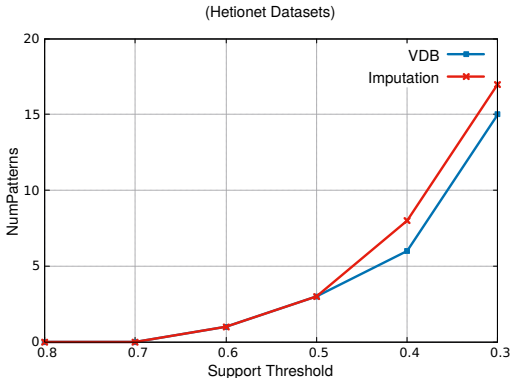

Figure 5.30 – *Number of Patterns for Hetionet(Gene) Dataset*

is higher as shown in Fig 5.31 as compared to crisp dataset as shown in Fig. 5.34. Similarly, peak heap memory consumption is also higher with fuzzy subset Fig. 5.32 and Fig 5.35. This is due to the fact that the number of attributes to be considered for gradual patterns is increased with the inclusion of fuzzy partitions (Low and High)for each property of node.

We observe an increase in the number of patterns being generated particularly at small support threshold values as shown in Fig. 5.33 and Fig 5.36. For example, number of patterns with 10K dataset at 0.1 support threshold, we get almost double the number of patterns with fuzzy sets. As we can see in Fig. 5.33 and Fig 5.36, in both type of experiments, the number of patterns for all the three datasets i.e., 1K, 10K and 20K are almost same. This is probably due to the fact that the data is generated synthetically using the graph generator. It would be interesting to observe the patterns generation with real graph datasets of different sizes. This is one of the points that may be investigated for optimization of results in term of patterns that are being generated.

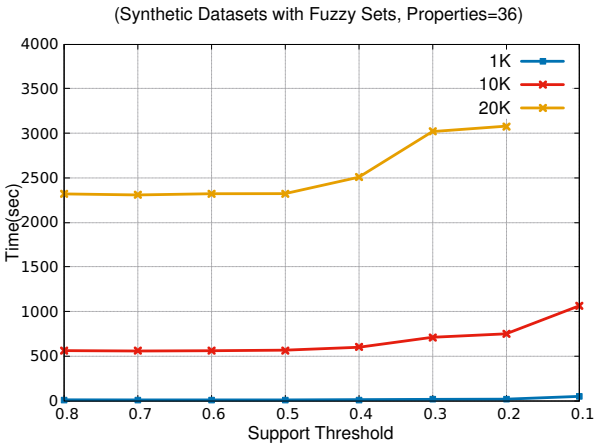

Figure 5.31 – *Time Utilization with Fuzzy Sets*

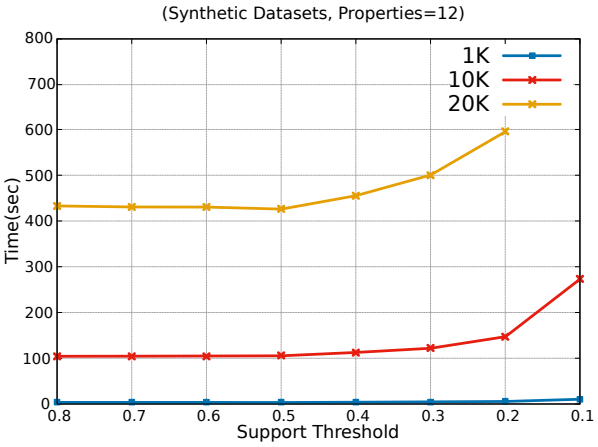

Figure 5.34 – *Time Utilization for Crisp Data*

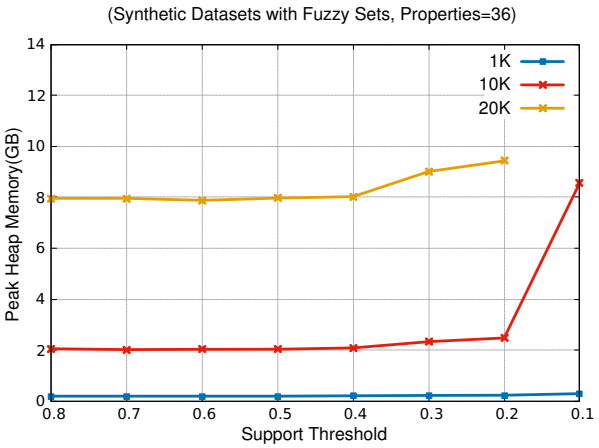

Figure 5.32 – *Memory Utilization with Fuzzy Sets*

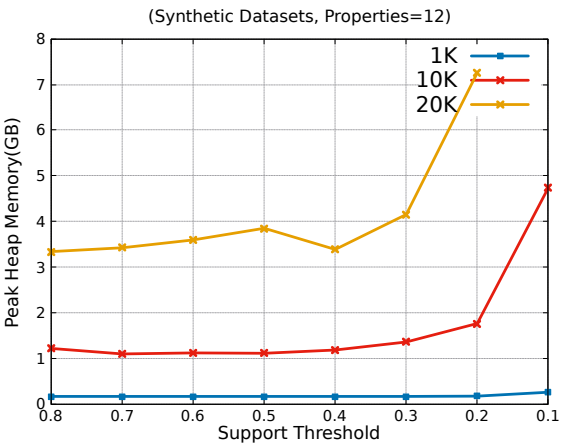

Figure 5.35 – *Memory Utilization for Crisp Data*

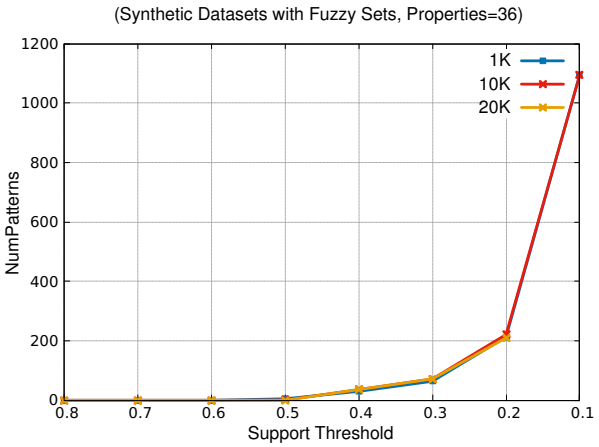

Figure 5.33 – *No. of Patterns with Fuzzy Sets*

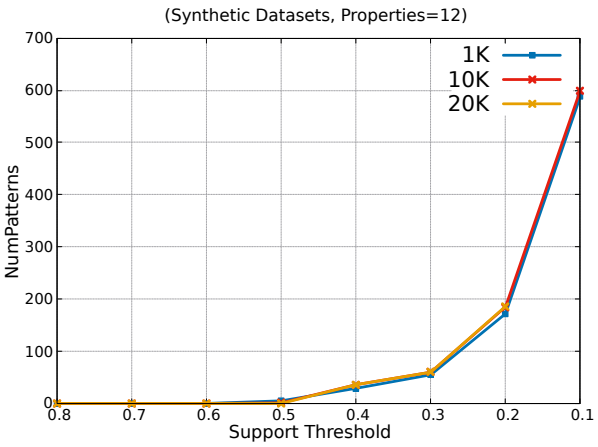

Figure 5.36 – *No. of Patterns for Crisp Data*

5.6 Discussion

It is found from the results of experiments for intra-nodes and node properties with relationships that the percentage of missing data has a relevance with the total number of extracted patterns. That is, with higher missing data percentage in a dataset, we get more patterns.

In the experiments for intra-node gradual patterns, it was observed that imputation method generates more patterns than vdb-based method. This implies that the imputation method generates the patterns that might not actually correlate in reality. Since the Russian-tweet-troll dataset has only 3% missing data, so here is almost no difference between both methods (i.e., vdb and imputation) in terms of patterns being extracted as shown in the NumPatterns Vs Support Threshold plots Figure 5.18. The same fact is visible in Figure 5.21 for Hepatitis dataset. Whereas, in Synthetic dataset plot Figure 5.24, at smaller support thresholds like 0.2, the difference of generated patterns is almost double. This shows that with higher percentage of missing data, the imputation method generates more biased results.

In the experiments for node properties with relationships scenario for the datasets shown in Table 5.2, we observed the similar fact that with more percentage of missing data the imputation method generates more patterns as shown in numPatterns in Figure 5.27. For the imputation method, this fact may be crucial in the case of real graph databases where practitioners or domain experts involvement becomes necessary to remove the false positives (i.e., the excessive patterns). The same is observed in Hetionet graph database at support threshold of 0.3, where we found (chromosome ↑ , PARTICIPATES_GpPW ↑) as an excessive pattern.

For the memory utilization results, with the exception of couple of instances, the vdb based method performs better than imputation method for all the three datasets of intra-node. The program's time utilization results also show that time difference almost doubles for both methods in all the three datasets of intra-node experiments. Whereas, the time utilization in case of node with relationships scenario is almost same as observed in Figure 5.25 and Figure 5.28 respectively.

It is deduced form the results that for both scenarios (intra-node and node properties and relationships count) for the given datasets, when the percentage of missing values is higher the imputation method generates more patterns than vdb approach. We think that it might not reflect the reality because data is being artificially imputed in missing places and as a result it generates more co-variance for the objects that are not actually present.

Chapter 6

Conclusion and Perspectives

6.1 Conclusion

This work addresses the extraction of gradual patterns from property graphs. To extract such patterns, we provide new protoforms of gradual patterns and the corresponding algorithms and architectures to extract them. Experiments have been run over synthetic and real graph databases.

We summarize below the main contributions of this work.

- We have proposed a formal definition of property graphs that is the first one in the literature to completely cover their particularities.

- We have designed different types of approaches for defining gradual patterns from property graphs as discussed in Chapter 3. These approaches provide different perspectives of mining the data depending upon the application domain and requirements of the end user. The first two types of patterns rely on node properties while the other three ones also exploit relationships.

- We also defined the concept of fuzzy gradual patterns from property graphs.

- We have proposed algorithms, methods and architectures in order to operationalize the extraction of such patterns from property graphs.

- In particular, we have investigated a mechanism to handle missing values in NoSQL semi-structured graph database. For such a goal, we propose a method to deal with the missing values to find gradual patterns from property graphs. Our approach considers the concept of valid databases *vdb* presented in [RC98]. We also have demonstrated the feasibility of integrating our work within a graph database engine.

- We have extended the synthetic graph generator application to generate property graph in NoSQL engine like Neo4j. The extended version allows to integrate " properties" in form of *key:value* for nodes and relationships. The existing version was able to generate dummy nodes and relationships for a given input number without properties. To mine gradual patterns, we require data within the nodes and relationships, hence the tool was extended which is publicly available at [Sha19b].

- We implemented our algorithms in Java and used Neoj4 engine for creating property graphs. The approach has been tested on graph data generated by a synthetic graph generator and real dataset. We presented performance comparison and evaluations of results in terms of time and memory to compare the approach of *vdb* with imputation method.

6.2 Perspectives

Considering the contributions mentioned above, we present following perspective works that can be taken as future research directions. We especially discuss improvements related to the integration the algorithms within Neo4j and then present some further works on the data generator.

6.2.1 Integration of the Algorithms in Neo4j

Section 4.4 presents the methodology we used to start integrating our work into Neo4j's graph database engine. The ultimate goal of this work is to provide a full library of our work as a third-party module for Neo4j that we will be shared with the open-source community. To do so, we have identified several key points that we need to address. They are discussed in more details in the rest of this section.

6.2.1.1 Can we extract gradual patterns through Cypher?

This question is intimately linked to that of inductive databases which have been intensively studied some years ago [BB06, Bes+10]. Neo4j engine allows to extend the features by adding "user-defined procedures" and "user-defined functions" to extend the functionality of graph database and Cypher query language.

As explained in Section 4.4.4, with the help of these user-defined functions and procedure we can use Cypher to retrieve gradual patterns directly from graph database engine rather than making an explicit database connection to run the algorithm and processing the data for patterns. These user-defined functions and procedures are written in Java, compiled into a jar file and can be placed in the plugins directory on each standalone or clustered server.

As described in Section 4.4, we have done a proof-of-concept that answer positively to that question. This POC has been created by developing a first user-defined function that allows users to mine gradual patterns on the properties of the nodes that share a same label. Nevertheless, this user-procedure does not address some issues related to the optimization as discussed in more details in the next section.

There are also some expressivity concerns to address: How to define user-friendly function that allow users to express what they want to achieve? For example, when treating the use-case of Inter-Relationships-Properties, how to users can express which relationships they want that the algorithm take into account?

- One idea is to take as input a pattern that describes the shape of the data the user want to be processed.

- Another possibility is to take as parameter a PATH, a segment that combines a relationship in a path with a start and end node that describe the traversal direction for that relationship. This PATH can be obtained as the result of a Cypher query or by using the Traversal API of Neo4j. Having a PATH as parameter will give access to a sub-set of the data (a subgraph) that contains the required data for running our algorithm.

We will have to inspect this leads in our further work to integrate smoothly our work to Neo4j.

6.2.1.2 Handling Categorical Ordinal Properties

In our work, we do not handle categorical ordinal variables for which the possible values may be ordered. [Mar+18] tries to exploit the information provided by the numerical attributes so as to extract knowledge about the categorical ones. For now, we do not provide any mechanism that allows the user to define the order of a property for example, academic ranks (assistant professor, associate professor, professor). In the current implementation, we are not able to manage these ranks.

That is why we have to think about a way to offer the possibility to the end-user to provide comparison function or when it is possible, the order of the elements of an ordinal variable. So that, we could map each categorical value with an index value in order to use them as an attribute in gradual pattern mining.

6.2.1.3 Integrating as an HTTP Service

Neo4j is a graph database engine that can be accessible through HTTP requests. For example, for a local installation of Neo4j, the users can do a cypher query through the REST API available at http://localhost:7474/db/data/cypher

As explained in Section 4.4.4.1, we could also create a server extension that enable new surfaces to be created in the HTTP API.

We could then offer new services to the users through HTTP APIs. Please find below some propositions of extension that could be helpful:

- An orchestrating system for gradual pattern mining. Indeed, some gradual patterns mining could take time to be processed. It could be interesting to have an asynchronous approach: letting the user ask for a gradual pattern mining through a HTTP API and then have a scheduling system that will perform the property-graph mining and send an email to the user so that it can check the results.

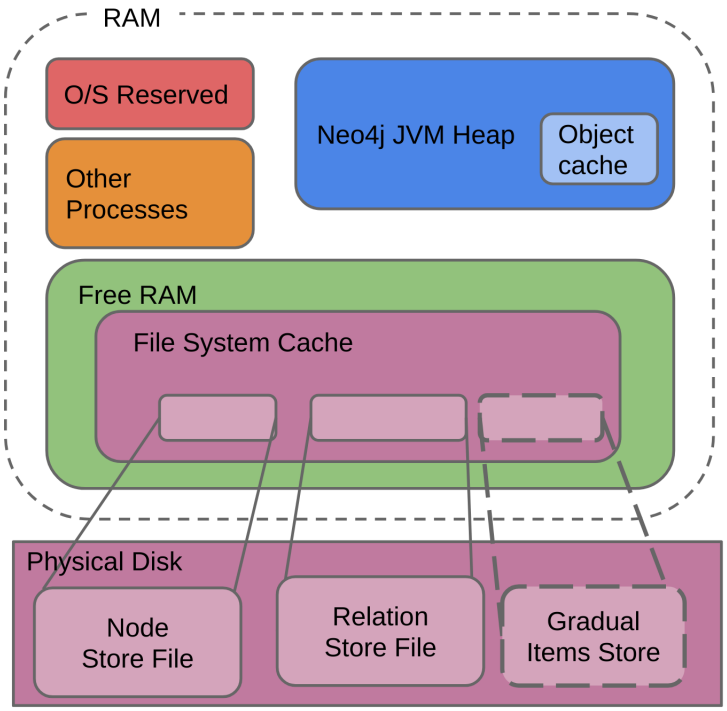

Figure 6.1 – *Cache and Filesystem*

- Categorical ordinal properties management system. A user interface or REST API could be added to let the user annotate the data and manage information on the data model (for example to give information on sortable/unsortable properties and comparable methods for ordinal variables).

6.2.1.4 Storage and Low-level API

Graph databases can have graph native storages which are filesystems that are optimized for graph processing and traversal access to the data:

- For instance, Neo4j uses extensively the offset mechanism for traversing from one node to another. There is also some materialization of relationships information inside nodes (such as the relation type, the ingoing and outgoing node's id) in order to speed up the traversal.
- Beyond this native graph storage system, there is also a caching system layer that optimize the I/O access as discussed in [Vuk+15].

We think that to optimize the execution time we should also use both approaches. That is to say, exploiting cache systems and also try to have an efficient storage system for binary matrices when we need to retrieve some information as presented in Figure 6.1.

Another optimization could be an efficient use of indexation. We could index some data and meta-data useful for our algorithms. For example, we could create some indexes to tell which fields are sortable and not sortable. We could use an index to map ordinal variables to their comparable functions. We could also optimize our process by indexing binary matrices.

6.2.2 Scalable Distributed Implementation of Gradual Pattern Mining

One of our further works is to provide a scalable distributed implementation of gradual pattern mining algorithm compatible with the use of missing values. To do so, there are several leads:

- There are several Graph Processing Systems. The most popular ones are GraphX, Giraph, and GraphLab, which are implementations of the ideas expressed in the Google Pregel[Mal+10] and on Map/Reduce papers like [DG08]. We aim at using GraphX Apache Spark's API, a graph processing system based on Apache Spark In-Memory BigData processing library, for graphs and graph-parallel computation.
- It has been recently announced that based on the achievements of the ongoing Cypher for Apache Spark project, Spark 3.0 users will be able to use the well-established Cypher graph query language for graph query processing, as well as having access to graph algorithms stemming from the GraphFrames project [Men18].
- Morpheus [1] builds on the Spark SQL DataFrame API, offering integration with standard Spark SQL processing and also allows integration with GraphX. Morpheus extends Apache Spark with Cypher. It allows for the integration of many data sources and supports multiple graph querying. It enables you to use your Spark cluster to run analytical graph queries. Queries can also return graphs to create processing pipelines.

The recent announce of several tools and the current traction around property-graph and spark ecosystem to handle them strengthens us in the belief that we going in the right direction.

6.2.3 Improving and Extending the Graph Generator

The evaluation of data mining techniques is crucial for assessing and benchmarking the proposed approaches. Several works exist for many data types (e.g., relational databases and regular graphs) but as for as we know, no tool is available for property graphs. We have therefore developed a graph generator presented in chapter 5.

The existing version of graph generator helps us to synthetically generate nodes and relationships with properties in GUI interface as well in a text file as Cypher statements that can

1. see https://github.com/opencypher/morpheus

used to create property graph. It is not a recommended practice in Neo4j to create property graphs by writing thousands of lines of Cypher in GUI interface or in Cypher-shell utility. Although, the graph generator can generate thousands of lines of Cypher statements but there is a limit to execute them for example 1500 to 2000 statements at a time. Therefore, it requires to load the Cypher statements in chunks.

We present two possible dimensions of work that can be carried out to further extend the graph generator and make it a comprehensive tool for creating property graph.

- Enable the tool to create the data in CSV file format that can be imported in Neo4j using LOAD CSV command. With the data in CSV format, we can load millions of nodes and relationships in few minutes.
- Identify the limits of data that can be generated in GUI interface and to automatically export the data in CSV if the size of input data to be generated is beyond the certain specified limit for instance, more than 100k nodes and relationships. This limit is because the tool is a browser based application and runs in user's browser therefore, specifications of the end user's machine are to be considered for data that is being generated.

Bibliography

[Abd07] Hervé Abdi. « The Kendall rank correlation coefficient ». In: *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA* (2007), pp. 508–510 (cit. on p. 31).

[Agg15a] Charu C Aggarwal. « Association Pattern Mining ». In: *Data mining: the textbook*. Springer, 2015. Chap. 4, pp. 93–132 (cit. on pp. 22–24).

[Agg14] Charu C. Aggarwal. « An Introduction to Frequent Pattern Mining ». In: *Frequent Pattern Mining*. 2014, pp. 1–17. DOI: 10.1007/978-3-319-07821-2_1. URL: https://doi.org/10.1007/978-3-319-07821-2%5C_1 (cit. on p. 22).

[Agg15b] Charu C. Aggarwal. *Data Mining: The Textbook*. Springer Publishing Company, Incorporated, 2015. ISBN: 9783319141411 (cit. on pp. 3–5).

[ABA14] Charu C Aggarwal, Mansurul A Bhuiyan, and Mohammad Al Hasan. « Frequent pattern mining algorithms: A survey ». In: *Frequent pattern mining*. Springer, 2014, pp. 19–64 (cit. on p. 22).

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. « Mining Association Rules Between Sets of Items in Large Databases ». In: *SIGMOD Rec.* 22.2 (June 1993), pp. 207–216. ISSN: 0163-5808. DOI: 10.1145/170036.170072. URL: http://doi.acm.org/10.1145/170036.170072 (cit. on p. 22).

[AS+94] Rakesh Agrawal, Ramakrishnan Srikant, et al. « Fast algorithms for mining association rules ». In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994, pp. 487–499 (cit. on p. 24).

[Ama19] Amazon. *Amazon Neptune: Fast, reliable graph database built for the cloud*. page accessed: Jan 2019. 2019. URL: https://aws.amazon.com/neptune/ (cit. on p. 12).

[Aur19] Titan Aurelius. *TITAN: Distributed Graph Database*. page accessed: Jan 2019. 2019. URL: http://titan.thinkaurelius.com/ (cit. on p. 12).

[Ayo+10] Sarra Ayouni, Sadok Ben Yahia, Anne Laurent, and Pascal Poncelet. « Fuzzy gradual patterns: What fuzzy modality for what result? » In: *SoCPaR: International Conference of Soft Computing and Pattern Recognition*. Paris, France, 2010, pp. 224–230. URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00798797 (cit. on pp. 8, 32, 34, 35).

[AY14] Sarra Ayouni and Sadok Ben Yahia. « Fuzzy set-based formalization of gradual patterns ». In: *2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*. IEEE. 2014, pp. 434–439 (cit. on p. 8).

[AYL11] Sarra Ayouni, Sadok Ben Yahia, and Anne Laurent. « Extracting compact and information lossless sets of fuzzy association rules ». In: *Fuzzy Sets and Systems* 183.1 (2011), pp. 1–25. DOI: 10.1016/j.fss.2011.06.019. URL: https://doi.org/10.1016/j.fss.2011.06.019 (cit. on p. 24).

[Ayr+02] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. « Sequential Pattern Mining Using a Bitmap Representation ». In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: ACM, 2002, pp. 429–435. ISBN: 1-58113-567-X. DOI: 10.1145/775047.775109. URL: http://doi.acm.org/10.1145/775047.775109 (cit. on p. 27).

[BM03] Gustavo E. A. P. A. Batista and Maria Carolina Monard. « An Analysis of Four Missing Data Treatment Methods for Supervised Learning ». In: *Applied Artificial Intelligence* 17 (2003), pp. 519–533 (cit. on p. 38).

[Beb+18] Bradley R. Bebee et al. « Amazon Neptune: Graph Data Management in the Cloud ». In: *International Semantic Web Conference*. 2018, pp. 1–2 (cit. on p. 7).

[Ben+09] Leila Ben Othman, François Rioult, Sadok Ben Yahia, and Bruno Crémilleux. « Missing Values: Proposition of a Typology and Characterization with an Association Rule-Based Model ». In: *Data Warehousing and Knowledge Discovery*. Ed. by Torben Bach Pedersen and A. Min Mohania Mukesh K. and Tjoa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 441–452 (cit. on p. 38).

[Ber+07] F. Berzal, J.-C. Cubero, D. Sanchez, M.-A. Vila, and J. M. Serrano. « An alternative approach to discover gradual dependencies ». In: *Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 15.5 (2007), pp. 559–570 (cit. on p. 26).

[Bes+10] Jérémy Besson, Jean-François Boulicaut, Tias Guns, and Siegfried Nijssen. « Generalizing Itemset Mining in a Constraint Programming Setting ». In: *Inductive Databases and Constraint-Based Data Mining*. Ed. by Saso Dzeroski, Bart Goethals, and Pance Panov. Springer, 2010, pp. 107–126. ISBN: 978-1-4419-7737-3. DOI: 10.1007/978-1-4419-7738-0_5. URL: https://doi.org/10.1007/978-1-4419-7738-0%5C_5 (cit. on p. 111).

[Bez+99] James C. Bezdek, Mikhil R. Pal, James Keller, and Raghu Krisnapuram. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Norwell, MA, USA: Kluwer Academic Publishers, 1999. ISBN: 0792385217 (cit. on p. 34).

[BB06] Francesco Bonchi and Jean-François Boulicaut, eds. *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers*. Vol. 3933. Lecture Notes in Computer Science. Springer, 2006. ISBN: 3-540-33292-8. DOI: 10.1007/11733492. URL: https://doi.org/10.1007/11733492 (cit. on p. 111).

[Bou+10] B. Bouchon-Meunier, A. Laurent, M. Lesot, and M. Rifqi. « Strengthening fuzzy gradual rules through "all the more" clauses ». In: *International Conference on Fuzzy Systems*. July 2010, pp. 1–7. DOI: 10.1109/FUZZY.2010.5584858 (cit. on p. 35).

[Bou18] Bernadette Bouchon-Meunier. « Strengths of Fuzzy Techniques in Data Science ». In: *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy, etc. Methods and Their Applications*. HAL, 2018. URL: https://hal.sorbonne-universite.fr/hal-01676195 (cit. on p. 34).

[Bow19] Andrew Bowman. *Understanding non-existent properties and working with nulls*. page accessed: Jan 2019. 2019. URL: https://neo4j.com/developer/kb/understanding-non-existent-properties-and-null-values/ (cit. on p. 63).

[CGM07] Toon Calders, Bart Goethals, and Michael Mampaey. « Mining Itemsets in the Presence of Missing Values ». In: *Proceedings of the 2007 ACM Symposium on Applied Computing*. SAC '07. Seoul, Korea: ACM, 2007, pp. 404–408. ISBN: 1-59593-480-4. DOI: 10.1145/1244002.1244097 (cit. on p. 65).

[CL14] Arnaud Castelltort and Anne Laurent. « Fuzzy queries over NoSQL graph databases: perspectives for extending the cypher language ». In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer. 2014, pp. 384–395 (cit. on p. 36).

[CL17] Arnaud Castelltort and Anne Laurent. « Exploiting NoSQL Graph Databases and in Memory Architectures for Extracting Graph Structural Data Summaries ». In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 25 (2017), pp. 81–110 (cit. on p. 52).

[CM18] Arnaud Castelltort and Trevor Martin. « Handling scalable approximate queries over NoSQL graph databases: Cypherf and the Fuzzy4S framework ». In: *Fuzzy Sets and Systems* 348 (2018), pp. 21–49 (cit. on pp. 71, 76).

[CGK06] Renato Coppi, Maria A Gil, and Henk AL Kiers. « The fuzzy approach to statistical analysis ». In: *Computational statistics & data analysis* 51.1 (2006), pp. 1–14 (cit. on p. 34).

[Dan19] Sergio Baranzini Daniel Himmelstein. *Hetnets in biomedicine*. page accessed: Jan 2019. 2019. URL: https://github.com/hetio/hetionet (cit. on pp. 100, 105).

[DG08] Jeffrey Dean and Sanjay Ghemawat. « MapReduce: simplified data processing on large clusters ». In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on p. 114).

[Dem+10] Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. « Dynamic Graph Algorithms ». In: *Algorithms and Theory of Computation Handbook*. Ed. by Mikhail J. Atallah and Marina Blanton. Chapman & Hall/CRC, 2010, pp. 9–9. ISBN: 978-1-58488-822-2. URL: http://dl.acm.org/citation.cfm?id=1882757.1882766 (cit. on p. 6).

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. « Maximum Likelihood from Incomplete Data via the EM Algorithm ». In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875 (cit. on p. 38).

[DLT08] Lisa Di Jorio, Anne Laurent, and Maguelonne Teisseire. « Fast extraction of gradual association rules: A heuristic based method ». In: *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology*. ACM. 2008, pp. 205–210 (cit. on p. 26).

[Do+15] Trong Dinh Thac Do, Alexandre Termier, Anne Laurent, Benjamin Négrevergne, Behrooz Omidvar-Tehrani, and Sihem Amer-Yahia. « PGLCM: efficient parallel mining of closed frequent gradual itemsets ». In: *Knowl. Inf. Syst.* 43.3 (2015), pp. 497–527. DOI: 10.1007/s10115-014-0749-8. URL: https://doi.org/10.1007/s10115-014-0749-8 (cit. on pp. 5, 8, 31, 62).

[Dub80] Didier J Dubois. *Fuzzy sets and systems: theory and applications*. Vol. 144. Academic press, 1980 (cit. on p. 33).

[DBE19] DB-Engines. *Knowledge Base of Relational and NoSQL Database Management Systems*. page accessed: April 2019. 2019. URL: https://db-engines.com/en/ranking/graph+dbms (cit. on p. 12).

[FPS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. « From data mining to knowledge discovery in databases ». In: *AI magazine* 17.3 (1996), pp. 37–37 (cit. on pp. 20, 21).

[Fer+16] Alberto Fernández, Cristobal José Carmona, Mariéa José del Jesus, and Francisco Herrera. « A view on fuzzy systems for big data: progress and opportunities ». In: *International Journal of Computational Intelligence Systems* 9.sup1 (2016), pp. 69–80 (cit. on p. 34).

[FG17] Michael Fire and Carlos Guestrin. « The Rise and Fall of Network Stars: Analyzing 2.5 million graphs to reveal how high-degree vertices emerge over time ». In: *arXiv preprint arXiv:1706.06690* (2017) (cit. on p. 5).

[For04] Bryan Ford. « Parsing expression grammars: a recognition-based syntactic foundation ». In: *ACM SIGPLAN Notices*. Vol. 39. 1. ACM. 2004, pp. 111–122 (cit. on p. 76).

[Fra+18] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. « Cypher: An evolving query language for property graphs ». In: *Proceedings of*

the 2018 International Conference on Management of Data. ACM. 2018, pp. 1433–1445 (cit. on p. 80).

[Ful16] FullAI. *A short history of artificial intelligence*. page accessed: April 2019. 2016. URL: http://www.fullai.org/short-history-artificial-intelligence/ (cit. on p. 3).

[Gam95] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995 (cit. on p. 73).

[Goe03] Bart Goethals. « Survey on frequent pattern mining ». In: *Univ. of Helsinki* 19 (2003), pp. 840–852 (cit. on p. 22).

[Gut84] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*. Vol. 14. 2. ACM, 1984 (cit. on p. 70).

[Hal+14] Patrick Hall, Jared Dean, Ilknur Kaynar Kabul, and Jorge Silva. « An overview of machine learning with SAS® enterprise miner™ ». In: *SAS Institute Inc* (2014) (cit. on pp. 2, 4).

[Han+07] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. « Frequent pattern mining: current status and future directions ». In: *Data Mining and Knowledge Discovery* 15.1 (Aug. 2007), pp. 55–86 (cit. on p. 68).

[HPK11] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011 (cit. on pp. 2, 4, 5).

[Him19] Daniel Himmelstein. *Hetnets in Neo4j*. page accessed: Jan 2019. 2019. URL: https://neo4j.het.io/browser/ (cit. on pp. 95, 100).

[Him+17] Daniel Scott Himmelstein, Antoine Lizee, Christine Hessler, Leo Brueggeman, Sabrina L Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E Baranzini. « Systematic integration of biomedical knowledge prioritizes drugs for repurposing ». In: *eLife* 6 (Sept. 2017). Ed. by Alfonso Valencia, e26726. ISSN: 2050-084X. DOI: 10.7554/eLife.26726. URL: https://doi.org/10.7554/eLife.26726 (cit. on p. 99).

[HKB11] James Honaker, Gary King, and Matthew Blackwell. « Amelia II: A Program for Missing Data ». In: *Journal of Statistical Software* 45.7 (2011), pp. 1–47 (cit. on p. 102).

[HHE03] Eduardo R. Hruschka, Estevam R. Hruschka, and Nelson F. F. Ebecken. « Evaluating a Nearest-Neighbor Method to Substitute Continuous Missing Values ». In: *AI 2003: Advances in Artificial Intelligence*. Ed. by Tamás (Tom) Domonkos Gedeon and Lance Chun Che Fung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 723–734 (cit. on p. 38).

[Hül02] Eyke Hüllermeier. « Association Rules for Expressing Gradual Dependencies ». In: *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*. PKDD '02. London, UK, UK: Springer-Verlag, 2002, pp. 200–211. ISBN: 3-540-44037-2. URL: http://dl.acm.org/citation.cfm?id=645806.670161 (cit. on pp. 24, 25, 33).

[Hül11] Eyke Hüllermeier. « Fuzzy Sets in Machine Learning and Data Mining ». In: *Appl. Soft Comput.* 11.2 (Mar. 2011), pp. 1493–1505. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2008.01.004. URL: http://dx.doi.org/10.1016/j.asoc.2008.01.004 (cit. on p. 34).

[JB09] Jens Jäkel and Georg Bretthauer. « Fuzzy system applications ». In: *Control Systems, Robotics and AutomatioN–Volume XVII: Fuzzy and Intelligent Control Systems* (2009), p. 107 (cit. on p. 34).

[DLT09] Lisa Di-Jorio, Anne Laurent, and Maguelonne Teisseire. « Mining Frequent Gradual Itemsets from Large Databases ». In: *Advances in Intelligent Data Analysis VIII*. Ed. by Niall M. Adams, Céline Robardet, Arno Siebes, and Jean-François Boulicaut. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 297–308. ISBN: 978-3-642-03915-7 (cit. on pp. 5, 8, 21, 25–29, 31, 35, 62).

[JV13] Salim Jouili and Valentin Vansteenberghe. « An empirical comparison of graph databases ». In: *2013 International Conference on Social Computing*. IEEE. 2013, pp. 708–715 (cit. on p. 80).

[Kam09a] Chandrika Kamath. « The Scientific Data Mining Process ». In: *Scientific data mining: a practical perspective*. Siam, 2009. Chap. 4, pp. 57–66 (cit. on pp. 3, 4, 22).

[Kam09b] Chandrika Kamath. *Scientific data mining: a practical perspective*. Vol. 112. Siam, 2009 (cit. on p. 2).

[KBB17] Arijit Khan, Sourav S. Bhowmick, and Francesco Bonchi. « Summarizing Static and Dynamic Big Graphs ». In: *Proc. VLDB Endow.* 10.12 (Aug. 2017), pp. 1981–1984. ISSN: 2150-8097. DOI: 10.14778/3137765.3137825. URL: https://doi.org/10.14778/3137765.3137825 (cit. on pp. 6, 8, 52).

[KH10] Hyung-Won Koh and Eyke Hüllermeier. « Mining Gradual Dependencies Based on Fuzzy Rank Correlation ». In: *Combining Soft Computing and Statistical Methods in Data Analysis, SMPS 2010, Oviedo, Spain, September 29 - October 1, 2010*. 2010, pp. 379–386. DOI: 10.1007/978-3-642-14746-3_47. URL: https://doi.org/10.1007/978-3-642-14746-3%5C_47 (cit. on p. 35).

[LMD14] Josep Lluís Larriba-Pey, Norbert Martínez-Bazán, and David Domínguez-Sal. « Introduction to graph databases ». In: *Reasoning Web International Summer School*. Springer. 2014, pp. 171–194 (cit. on p. 80).

[LLR09] Anne Laurent, Marie-Jeanne Lesot, and Maria Rifqi. « GRAANK: Exploiting Rank Correlations for Extracting Gradual Itemsets ». In: *Flexible Query Answering Systems*. Ed. by Troels Andreasen, Ronald R. Yager, Henrik Bulskov, Henning Christiansen, and Henrik Legind Larsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 382–393. ISBN: 978-3-642-04957-6 (cit. on pp. 5, 8, 21, 25, 26, 31, 32, 35, 49, 62).

[Lau+12] Anne Laurent, Benjamin Négrevergne, Nicolas Sicard, and Alexandre Termier. « Efficient parallel mining of gradual patterns on multicore processors ». In: *Advances in Knowledge Discovery and Management*. Springer, 2012, pp. 137–151 (cit. on p. 8).

[Liu+18] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. « Graph Summarization Methods and Applications: A Survey ». In: *ACM Comput. Surv.* 51.3 (June 2018), 62:1–62:34. ISSN: 0360-0300. DOI: 10.1145/3186727. URL: http://doi.acm.org/10.1145/3186727 (cit. on pp. 6, 52).

[Log16] Neota Logic. *Artificial Intelligence in Law: The State of Play 2016, Part 1*. page accessed: April 2019. 2016. URL: https://www.neotalogic.com/2016/02/28/artificial-intelligence-in-law-the-state-of-play-2016-part-1/ (cit. on p. 3).

[Lun+07] Max Lungarella, Fumiya Iida, Josh C Bongard, and Rolf Pfeifer. « AI in the 21 st Century–With Historical Reflections ». In: *50 years of artificial intelligence*. Springer, 2007, pp. 1–8 (cit. on p. 2).

[Mal+10] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. « Pregel: a system for large-scale graph processing ». In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 135–146 (cit. on p. 114).

[Mar+18] Christophe Marsala, Anne Laurent, Marie-Jeanne Lesot, Maria Rifqi, and Arnaud Castelltort. « Discovering Ordinal Attributes Through Gradual Patterns, Morphological Filters and Rank Discrimination Measures ». In: *International Conference on Scalable Uncertainty Management*. Springer. 2018, pp. 152–163 (cit. on p. 112).

[Men18] Xiangrui Meng. *SPIP: Property Graphs, Cypher Queries, and Algorithms*. Nov. 2018. URL: https://issues.apache.org/jira/browse/SPARK-25994 (cit. on p. 114).

[Mic19] Microsoft. *Azure Cosmos DB: Globally distributed, multi-model database service*. page accessed: Jan 2019. 2019. URL: https://azure.microsoft.com/en-gb/services/cosmos-db/ (cit. on p. 12).

[Mil13] Justin J Miller. « Graph database applications and concepts with Neo4j ». In: *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*. Vol. 2324. 2013 (cit. on pp. 7, 12).

[MP18] Karthika Mohan and Judea Pearl. « Graphical Models for Processing Missing Data ». In: *arXiv:1801.03583v1 [stat.ME]* (Jan. 2018), pp. 1–34 (cit. on p. 38).

[Muñ+17] Vıéctor Méndez Muñoz, Anna Cohen-Nabeiro, Romain David, Vicente Ivars Camáñez, Alfons Nonell-Canals, Miquel Senar, Denis Couvet, Jean-Pierre Feral, Aurélie Delavaud, and Thierry Tatoni. « Analysis on the Graph Techniques for Data-mining and Visualization of Heterogeneous Biodiversity Data Sets ». In: *Complexis 2017*. 2017, pp. 144–151 (cit. on p. 5).

[Nar19] Balasubramanian Narasimhan. *The Normal Distribution*. page accessed: Jan 2019. 2019. URL: http://statweb.stanford.edu/~naras/jsm/NormalDensity/NormalDensity.html (cit. on p. 94).

[Neo19a] Neo4j. *Neo4j - The Fastest Path To Graph Success*. page accessed: Jan 2019. 2019. URL: https://neo4j.com/ (cit. on p. 12).

[Neo19b] Neo4j. *Neo4j Debian installation*. page accessed: Jan 2019. 2019. URL: https://neo4j.com/docs/operations-manual/current/installation/linux/debian/ (cit. on p. 131).

[Neo19c] Neo4j. *Neo4j is the Go-To Technology for Retail- Neo4j Retail Customers*. page accessed: Jan 2019. 2019. URL: https://neo4j.com/industries/retail/ (cit. on p. 5).

[Neo19d] Neo4j. *Neo4jCommunityVersion*. page accessed: Jan 2019. 2019. URL: https://gite.lirmm.fr/shah/neo4j-community-version-3.4.7/ (cit. on p. 131).

[Neo19e] Neo4j. *The World of Graphs — Powered by Neo4j*. page accessed: Jan 2019. 2019. URL: https://neo4j.com/customers/ (cit. on p. 5).

[New19] NBC News. *Russian troll tweets*. 2019. URL: https://www.nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731 (cit. on pp. 96, 97).

[Ngo+18] Tu Ngo, Vera Georgescu, Anne Laurent, Thérèse Libourel, and Grégoire Mercier. « Mining Spatial Gradual Patterns: Application to Measurement of Potentially Avoidable Hospitalizations ». In: *International Conference on Current Trends in Theory and Practice of Informatics*. Springer. 2018, pp. 596–608 (cit. on p. 8).

[Ora19] Oracle. *Java SE 8 installation*. page accessed: Jan 2019. 2019. URL: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html (cit. on p. 131).

[ora18] oracle. *Java Memory Types*. 2018. URL: https://docs.oracle.com/javase/7/docs/api/java/lang/management/MemoryPoolMXBean.html (cit. on p. 94).

[OW72] T. Orchard and M. A. Woodbury. « A missing information principle: theory and applications ». In: *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*. 1972, pp. 697–715 (cit. on p. 38).

[PL84] T. Papaioannou and S. Loukas. « Inequalities on Rank Correlation with Missing Data ». In: *Journal of the Royal Statistical Society Series B (Methodological)* 46.1 (1984), pp. 68–71 (cit. on pp. 31, 38).

[PG98] Witold Pedrycz and Fernando Gomide. *An introduction to fuzzy sets: analysis and design*. Mit Press, 1998 (cit. on p. 33).

[Piv+16] Olivier Pivert, Olfa Slama, Grégory Smits, and Virginie Thion. « SUGAR: A graph database fuzzy querying system ». In: *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. IEEE. 2016, pp. 1–2 (cit. on p. 71).

[QLP11] Malaquias Quintero, Anne Laurent, and Pascal Poncelet. « Fuzzy orderings for fuzzy gradual patterns ». In: *International Conference on Flexible Query Answering Systems*. Springer. 2011, pp. 330–341 (cit. on pp. 8, 34, 35).

[RC98] Arnaud Ragel and Bruno Crémilleux. « Treatment of Missing Values for Association Rules ». In: *Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD-98, Melbourne, Australia, April 15-17, 1998, Proceedings*. 1998, pp. 258–270 (cit. on pp. 38, 40, 62, 63, 65, 110).

[Rep19] UCI Machine Learning Repository. *Hepatitis Data Set*. 2019. URL: https://archive.ics.uci.edu/ml/datasets/hepatitis (cit. on p. 98).

[Reu16] Thomson Reuters. *Artificial Intelligence: How will it affect legal practice – and when?* page accessed: April 2019. 2016. URL: https://blogs.thomsonreuters.com/answerson/artificial-intelligence-legal-practice/ (cit. on p. 3).

[Reu19] Thomson Reuters. *Early adoption is an investment that pays dividends*. page accessed: April 2019. 2019. URL: https://legal.thomsonreuters.com/en/insights/infographics/early-adoption-is-an-investment-that-pays-dividends (cit. on p. 2).

[RWE15] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases: New Opportunities for Connected Data*. 2nd. O'Reilly Media, Inc., 2015. ISBN: 9781491930892 (cit. on pp. 5–7, 14).

[Rod16] Marko A. Rodriguez. *Graph Morphisms*. page accessed: Jan 2019. 2016. URL: https://github.com/tinkerpop/blueprints/wiki/Graph-Morphisms (cit. on p. 5).

[RN10] Marko A Rodriguez and Peter Neubauer. « Constructions from dots and lines ». In: *Bulletin of the American Society for Information Science and Technology* 36.6 (2010), pp. 35–41 (cit. on pp. 5, 6).

[RN12] Marko A Rodriguez and Peter Neubauer. « The graph traversal pattern ». In: *Graph Data Management: Techniques and Applications*. IGI Global, 2012, pp. 29–46 (cit. on pp. 5–7).

[RUB76] DONALD B. RUBIN. « Inference and missing data ». In: *Biometrika* 63.3 (1976), pp. 581–592. DOI: 10.1093/biomet/63.3.581. URL: http://dx.doi.org/10.1093/biomet/63.3.581 (cit. on pp. 36, 37).

[SA11] Sherif Sakr and Ghazi Al-Naymat. « Querying Graph Databases: An Overview ». In: *Advanced Database Query Systems: Techniques, Applications and Technologies*. IGI Global, 2011, pp. 304–322. DOI: 10.4018/978-1-60960-475-2.ch013 (cit. on p. 12).

[Sha18] Faaiz Shah. *Source Code for Synthetic Graph Generator Extension*. page accessed: Jan 2019. 2018. URL: https://github.com/faaizshah/graphs (cit. on pp. 86, 137).

[Sha19a] Faaiz Shah. *Gradual Pattern Extraction- Java code*. page accessed: Jan 2019. 2019. URL: https://gite.lirmm.fr/shah/handlingmissingdata (cit. on pp. 93, 95, 98, 99, 102, 103).

[Sha19b] Faaiz Shah. *GUI Interface for Synthetic Graph Generator*. page accessed: Jan 2019. 2019. URL: https://faaizhussain.github.io/graphs/ (cit. on pp. 80, 84, 99, 101, 110, 134, 137).

[SRA17] Waseem Shahzad, Qamar Rehman, and Ejaz Ahmed. « Missing Data Imputation using Genetic Algorithm for Supervised Learning ». In: *International Journal of Advanced Computer Science and Application (IJACSA)* 8.2 (Feb. 2017), pp. 438–445 (cit. on p. 38).

[Sic+13] Nicolas Sicard, Yogi Aryadinata, Federico Del Razo Lopez, Anne Laurent, and Perfecto Flores. « Multi-core parallel gradual pattern mining based on multi-precision fuzzy orderings ». In: *Algorithms* 6.4 (2013), pp. 747–761 (cit. on p. 35).

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. « Mining Quantitative Association Rules in Large Relational Tables ». In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. SIGMOD '96. Montreal, Quebec, Canada: ACM, 1996, pp. 1–12. ISBN: 0-89791-794-4. DOI: 10.1145/233269.233311. URL: http://doi.acm.org/10.1145/233269.233311 (cit. on p. 24).

[Str15] Umberto Straccia. « All about fuzzy description logics and applications ». In: *Reasoning Web International Summer School*. Springer. 2015, pp. 1–31 (cit. on p. 34).

[Swa18] Alvira Swalin. *How to Handle Missing Data*. Jan. 2018. URL: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4 (cit. on p. 37).

[Tru94] Richard J Trudeau. *Introduction to graph theory*. Dover Books on Mathematics. Mineola, NY: Dover, 1994. URL: http://cds.cern.ch/record/2009885 (cit. on p. 6).

[Van14] Rik Van Bruggen. *Learning Neo4j*. Packt Publishing Ltd, 2014 (cit. on p. 6).

[Vuk+15] Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. *Neo4j in action*. Vol. 22. Manning Shelter Island, 2015 (cit. on p. 113).

[Wik19] Wiki-Angular. *About AngularJS*. page accessed: Jan 2019. 2019. URL: https://en.wikipedia.org/wiki/AngularJS (cit. on p. 80).

[WHR98] Graham Williams, Markus Hegland, and Stephen Roberts. « A data mining tutorial ». In: *IASTED Int. Conf. on Parallel and Distributed Computing and Networks, PDCN*. Vol. 98. 1998, p. 14 (cit. on p. 2).

[Win18] Mégane Wintz. *Graph Generator*. page accessed: Jan 2019. 2018. URL: https://fastgraph-generator.herokuapp.com/ (cit. on pp. 80, 89, 137).

[WWC04] Chih-Hung Wu, Chian-Huei Wun, and Hung-Ju Chou. « Using association rules for completing missing data ». In: *Hybrid Intelligent Systems, 2004. HIS '04. Fourth International Conference on*. Dec. 2004, pp. 236–241. DOI: 10.1109/ICHIS.2004.91 (cit. on p. 38).

[Wu+14] Y. Wu, Z. Zhong, W. Xiong, and N. Jing. « Graph summarization for attributed graphs ». In: *2014 International Conference on Information Science, Electronics and Electrical Engineering*. Vol. 1. Apr. 2014, pp. 503–507. DOI: 10.1109/InfoSEEE.2014.6948163 (cit. on p. 52).

[Yan04] Guizhen Yang. « The complexity of mining maximal frequent itemsets and maximal frequent patterns ». In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. 2004, pp. 344–353. DOI: 10.1145/1014052.1014091. URL: https://doi.org/10.1145/1014052.1014091 (cit. on p. 68).

[Zad65] Lotfi A Zadeh. « Fuzzy sets ». In: *Information and control* 8.3 (1965), pp. 338–353 (cit. on pp. 33, 34).

[Zad08] Lotfi A Zadeh. « Is there a need for fuzzy logic? » In: *Information sciences* 178.13 (2008), pp. 2751–2779. DOI: https://doi.org/10.1016/j.ins.2008.02.012. URL: http://www.sciencedirect.com/science/article/pii/S0020025508000716 (cit. on p. 33).

Publications

- Journal Publication (Accepted)

  Faaiz Shah, Arnaud Castelltort, Anne Laurent, "Handling Missing Values for Mining Gradual Patterns from NoSQL Graph Databases", Future Generation Computer Systems (FGCS), Elsevier, Oct 2019. [Impact Factor: 5.768]

- Conference Proceeding
  - Faaiz Shah, Arnaud Castelltort, Anne Laurent. "Extracting Fuzzy Gradual Patterns from Property Graphs" FUZZ-IEEE, June 2019, New-Orleans, United States.

Appendix A

Flowchart and Program Execution

A.1 Flowchart

The following two flowchart show the overall execution of program from connection to graph database to the final result.

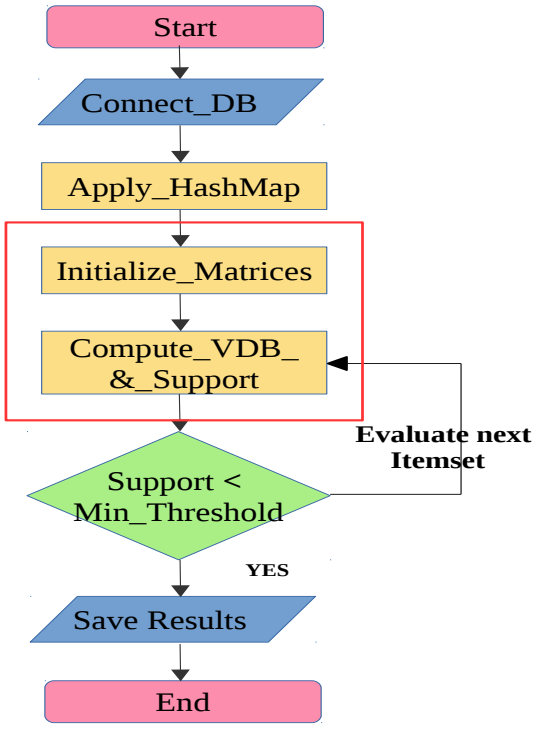

Figure A.1 – *(vdb) approach*

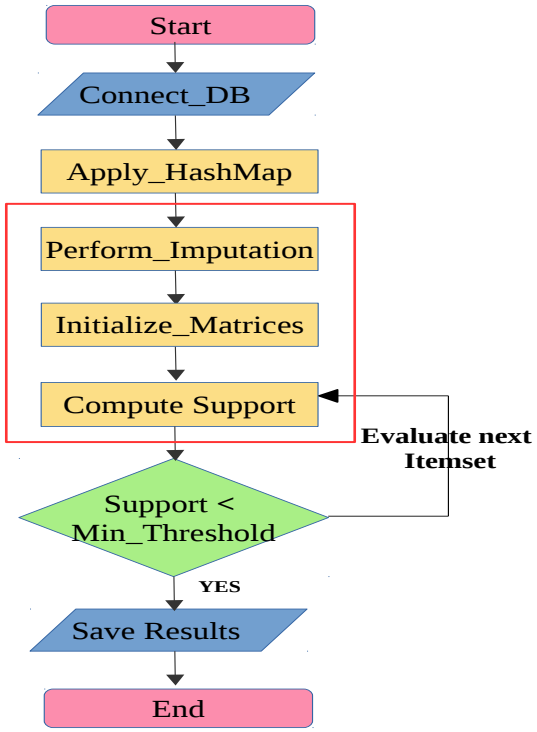

Figure A.2 – *Imputation Method*

A.2 Program Execution

Following is summary of the steps for this program execution.

1. Getting Started
   (a) System Specifications
   (b) Prerequisites
   (c) Installations
2. Running the program
   (a) Individual program execution
   (b) Multiple executions of program for plotting charts

A.2.1 Getting Started

This program provides the details about how to run this Java program for extraction of gradual patterns from property graphs. We show the necessary tool and commands to generate output files.

A.2.1.1 System Specifications

For running these experiments, following were our system specification.

- Hardware: Intel Core i7-4610M, 3.00 GHz, quad core
- Processor, 16 GB RAM
- Operating System: Linux generic kernel 4.4.0-134, Ubuntu 16.04 LTS

Note:

It is not necessary to have exactly same or higher hardware specifications to run this program. If you choose to run this program for a database smaller of nodes and relationships, for example upto 5000 nodes and 5000 relationships, then Linux OS with 4 GB RAM and Core i3/i5 processor can easily run this program. An example setup for a database with 1000 nodes and 1000 relationships, as show in following setup. The same setup can execute higher number of nodes and relationships e.g. 20,000 nodes and 20,000 relationships with the above mentioned hardware specifications or with higher specifications.

A.2.1.2 Prerequisites

To run this program, you need to have following softwares.

Mandatory:

- Java version 1.8.0_181, Java(TM) SE Runtime Environment (build 1.8.0_181-b13) or higher
- Neo4j graph database version 3.4.7 or higher

Optional (if you want to execute scripts to reproduce results):

- Python 2.7.12
- Linux Ubuntu 16.04.5 LTS

A.2.1.3 Installations

We tested this program execution with neo4j community version 3.4.7. The installation instructions for neo4j on debian are available at [Neo19b]. Java 8 installation package is available at [Ora19]. If you want to download the community version that we used for running these experiments, then it is available at [Neo19d].

After the successful installations, following are the outputs of version commands for installations verification:

```
$ java -version

Output:
    java version "1.8.0\_181"
    Java(TM) SE Runtime Environment (build 1.8.0\_181-b13)
    Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13)

$ readlink -f $(which java)

Output:
    /usr/lib/jvm/jdk1.8.0_181/bin/java
```

Python may be installed in Ubuntu by executing following command:

```
$ sudo apt-get install python-minimal

$ python --version

Output:
    Python 2.7.12
```

A.2.2 Running the Program

Steps

1. Download the project in any format (tar or zip)
2. Go to the download location in your host machine and untar the downloaded file "gpnr-master.tar.gz"

```
$ tar -zxvf gpnr-master.tar.gz
```

3. Go to the directory and check files as shown below

```
$ cd gpnr-master/GPNR/

$ ls -l
```

The above output contains the "src" directory which contains complete source code of this program. It also contains the "pom.xml" file to see the packaging and dependencies details.

If you see the "GPNR-Node-Rel-1K-Nodes.jar" and "graph-1K-GG.db.tar.gz", then you are ready to execute the program.

A.2.2.1 Individual program execution

Now, first we have to configure the neo4j.conf file. Open the config file

```
$ nano /etc/neo4j/neo4j.conf
```

In the case, you are using community version, go the directory of neo4j config directory and open the config file

```
$ cd neo4j-community-3.4.7/

$ nano ./conf/neo4j.conf
```

Comment the following line and add a new line as shown in neo4j.conf file

```
#dbms.active_database=graph.db
dbms.active_database=graph-1K-GG.db
```

Save the file Ctrl+o and exit Ctrl+x

Untar the "graph-1K-GG.db.tar.gz" and place it into neo4j database folder

```
$ cd ~/gpnr-master/GPNR/
```

Copy the database file into /neo4j-home/data/databases/ folder. In our case, we are using community version. So the command will be as follows:

```
$ cp graph-1K-GG.db.tar.gz ~/neo4j-community-3.4.7/data/databases/

$ tar -zxvf graph-1K-GG.db.tar.gz
```

This will result an folder with name "graph-1K-GG.db" in location /neo4j-community-3.4.7/data/databases/

Start the neo4j service

```
$ service neo4j start
```

In case you are using community version,

```
$ ./neo4j console
```

On successful start of the service, you will see the output on console, something similar to following:

```
INFO  ======== Neo4j 3.4.7 ========
INFO  Starting...
INFO  Bolt enabled on 0.0.0.0:7687.
INFO  Started.
Remote interface available at http://localhost:7474/
```

Test that neo4j is working correctly and having nodes and relationships. Open http://localhost:7474/ in browser to verify. The username is "neo4j" and password is "lirmm" without quotes. Please note that username and password must be same as mentioned. These credentials are used by java program to communicate with graph database.

Note: If you do not want to use this test database and want to create a new neo4j database using our synthetic graph generator [Sha19b]. It generates the cypher to create graphs in Neo4j.

After the successful start and verification of Neo4j on host machine, now we will run our java program.

The program requires two inputs:

1. Input jar file (in our test case i.e., GPNR-Node-Rel-1K-Nodes.jar)

2. Minimum support threshold (normally in range from 0.1 to 0.9)

It is better to save the output of the program in a text file so as to view it separately. Therefore, the command will be:

```
$ java -jar GPNR-Node-Rel-1K-Nodes.jar 0.2 >
Result-Support-0.2.txt
```

For experiments, sometime it is better to allocate static memeory (e.g., 2G as shown below) to program, efficient G1 garbage collection and zero "0" biased locking delay. We use following JVM flags to run the individual execution.

```
–Xmx2G –XX:+UseG1GC –XX:BiasedLockingStartupDelay=0
```

Therefore, the final command with jvm flags would be:

```
$ java –Xmx10G –XX:BiasedLockingStartupDelay=0 –XX:+UseG1GC
–jar GPNR–Node–Rel–1K–Nodes.jar 0.2 > Result–Support–0.2.txt
```

The successful execution of the program will generate a text file with name "Result-Support-0.2.txt" in the current directory, which is /gpnr-master/GPNR/. You will also notice that after the program execution, a folder with name "csv" is created that contains the data files retrieved from graph database. The final lines of the output file may look like:

```
List of Patterns with Valid_Database:
_____________________________________________

Total Number of Patterns : 56

[[2+], [1+]] 0.3338108643867335
[[1+], [2–]] 0.3096400193233496
[[4+], [1+]] 0.5149839249720978
[[4–], [1+]] 0.4654095384051573
[[5+], [1+]] 0.21682130899035498
[[5–], [1+]] 0.23287967883260316
[[6+], [1+]] 0.4866985390881378
.
.
.
.

[[8+], [6–], [1+]] 0.20677649880894872
[[6+], [8–], [4+]] 0.2336292915327081

************************************************
```

```
------- Run Time & Peak Heap Memory Result are: -------

Total Heap Peak Used : 282
Program Run time : 10.083783252 seconds
```

A.2.2.2 Multiple executions of program for plotting charts

In this section we will describe the details of multiple executions of program in order to plot the charts. As we can see in Step 3, the output of the command "ls -l" shows two script files namely, "GPNR-Node-Rel.sh" and "resultsPythonScriptGPNR.py". Please note, to run this script, it must be in the directory where the jar file and python script is present. In our case currently, it is " /gpnr-master/GPNR/".

Program execution: graph database with 1,000 nodes and 1,000 relationships:

This database contains 3 node labels, 3 relationship types, and 9 node properties per label. In order to execute given .jar file (i.e., GPNR-Node-Rel-1K-Nodes.jar) for minimum support threshold values ranging from 0.1 to 0.8, we will execute the bash script as follows:

```
$ ./GPNR-Node-Rel.sh
```

The script will ask you to provide the input jar file as shown below:

```
$ ./GPNR-Node-Rel.sh

Bienvenue et Bonjour !

entrez le nom de fichier avec l'extension␣.jar␣:
```

Now, type the input file i.e., GPNR-Node-Rel-1K-Nodes.jar. The script will start executing the program and you will see the output on console like:

```
$ ./GPNR-Node-Rel.sh

Bienvenue et Bonjour !
```

```
    entrez le nom de fichier avec l'extension .jar :
    GPNR-Node-Rel-1K-Nodes.jar


    WELCOME TO GRADUAL PATTERN EXTRACTION PROGRAM

    Input file : GPNR-Node-Rel-1K-Nodes.jar
    Minimum support : 0.1

    AvgTimeConsumption : 10.4684838968
    AvgMemoryUtilization : 258.4


    WELCOME TO GRADUAL PATTERN EXTRACTION PROGRAM

    Input file : GPNR-Node-Rel-1K-Nodes.jar
    Minimum support : 0.2

    AvgTimeConsumption : 6.0477500466
    AvgMemoryUtilization : 182.6
    .
    .
    .

    Program ends
```

This script runs the program five (5) times keeping an interval of 15 seconds in each execution. It will create a new directory namely "Output-Results" in the current path that contains the results of all the 5 executions along with an average result csv file for each minimum support threshold ranging from 0.1 to 0.8.

A.3 Synthetic Graph Generation Tool

We have developed a synthetic graph generator to make property graphs in Neo4j database. This synthetic graph generator [Win18] was first developed by Mégane Wintz, as a thesis internship at polytech Montpellier, France. Later on, to add the node properties feature, it was extended for which the GUI is available at [Sha19b] and the source code is available [Sha18]. The project was created using Angular JS. The basic configuration details are as given below.

```
Graph Generator Configuration details:
---------------------------------------

Angular CLI: 6.1.3

Node: 10.14.0

OS: linux x64 (Ubuntu 16.04)

Angular: 5.2.11


Pre-requisities:
-----------------

Node.js

$ sudo apt-install nodejs
$ sudo apt-install npm

Check versions for verifying successful install:
-------------------------------------------------

$ node -v
$ npm -v

Install FileSaver.js, for saving files on the client-side:
-----------------------------------------------------------

$ npm install file-saver --save


Install node modules:
----------------------

$ npm install

Open the Application:
----------------------

$ ng serve --open
```

```
It will open the application on http://localhost:4200/
```