Handling Missing Values for Mining Gradual Patterns from NoSQL Graph Databases

Faaiz Hussain Shah, Arnaud Castelltort, Anne Laurent

Handling Missing Values for Mining Gradual Patterns from NoSQL Graph Databases

Faaiz Shah, Arnaud Castelltort, Anne Laurent

LIRMM, University of Montpellier, CNRS, Montpellier, France

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 edges 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 arises 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 because they may 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.

Keywords: Gradual patterns; Property graph, Missing values

1. Introduction

With the provision of ever increasing data rates on Internet and open source technologies to the end users, it has become an 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.

Data mining involves collecting, cleaning, processing and gaining useful insight from data. Its applications are often closely related to one of the four main problems i.e., pattern mining, clustering, classification and outlier analysis [1]. Frequent pattern mining is a process in which the data patterns having more occurrences than a predefined threshold mined. It is one of the most investigated domain in data mining [2]. Frequent pattern mining has rapidly extended from in transactional databases analysis to the analysis of complex structures having numerical attributes such as sequences, trees or graphs.

Association rule mining is generally referred to as frequent itemset (pattern) mining and it is used to discover the association rules ($X \rightarrow Y$) between items in transactional data. For mining quantitative data, [3] proposes a new type of rules to express a kind of "tendency" aka a *gradual dependence between attributes*. [3] present a first interpretation of *"gradual dependency"* 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"*. Therefore, a *gradual dependency* is defined as a pair of gradual itemsets on which a causality relationship is imposed [4].

Gradual pattern mining is an extension of frequent pattern mining. Gradual pattern mining is the process of discovering knowledge from databases as comparable attributes of co-variations. In linguistic expression, it may

*Corresponding author

Email address: faaiz.shah@lirmm.fr (Faaiz Shah, Arnaud Castelltort, Anne Laurent)

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 [4]. These co-variations can be increasing or decreasing and these co-variations 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 pattern is considered as interesting pattern 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 [4, 5] are mainly for relational databases while some other types are emerging, as for instance property graphs.

NoSQL graph database engines are purpose built systems to store nodes and edges natively. Nodes or vertices (data entities) are created and linked through 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"* [6]. 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 [7].

A property graph data model is a graph structure containing nodes and edges with properties/attributes in form of (key:value) pairs. Nodes have *Labels* that define the role of a node. Edges between two nodes have a *Type*; they are directional and may contain (key:value) properties as nodes. As shown in Figure 1, *Patient*, *Doctor*, *Medicine* are node labels, whereas *Takes*, *Visits* and *Advises* are edge types. A node can have one or more labels at the same time such as nodes with label *Patient* and *Doctor*.

Graph modeling is generally an iterative process. For building a property graph model, labels and relevant properties are assigned to nodes. Then we identify and assign the edges between these nodes and the possible properties that they can have. Property graph data model enables us to represent the data in a natural way. It allows the flexibility to incorporate schema changes as and when required because it does not require a fixed schema prior to the database creation. Hence, they are also referred to as

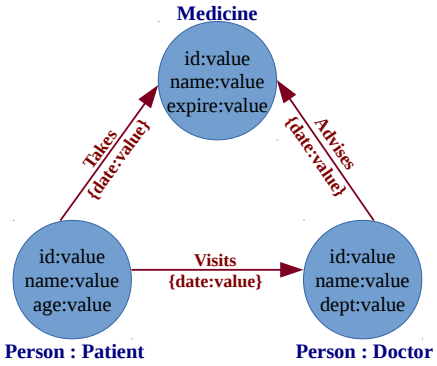


Figure 1: Property Graph Model

semi-structured datasets.

When trying to extract gradual patterns from a property graph, all the properties of a node may not be present. This is due to the reason of semi-structured nature of property graphs and often they do not have predefined schema. For example, Patient's age or lab-tests information may be missing for some nodes. So, regular algorithms can not be applied and the problem can be seen as a problem of handling missing data.

The research is being actively pursued for last one decade in gradual pattern mining such as mining the patterns from large numerical tabular datasets as well as addressing the scalability issues [4, 5, 8, 9, 10, 11, 12, 13]. Also, the existing gradual pattern mining techniques presented in [4, 5, 10] are mainly for tabular databases while some other types are emerging, as for instance property graphs.

To the best of our knowledge, the extraction of gradual patterns from property graphs is a novel idea. The objective of this work is to address the 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.

There may be several ways to define gradual patterns in the context of property graphs. For example, Intra-Node-Label gradual patterns, Inter-Node-Label gradual patterns, Node-properties-with-Edges-count gradual patterns, and Inter-Edge-properties gradual patterns. The focus of this paper is on two types i.e., *Intra-Node-Label* and *Node-properties-with-Edges-count* gradual patterns. The details of pattern extraction for these two scenarios are presented with relevant examples and experimental results

on real and synthetic datasets.

The paper is organized as follows. Section 2 describes the preliminary concepts about gradual patterns, property graphs, missing data types and their handling techniques. Section 3 presents the problem statement. In section 4, we present our proposed approach for mining gradual patterns in presence of missing data, particularly in the context of property graphs. In section 5, we show experimental results and comparisons of proposed approach with imputation technique. Section 6 presents conclusion and future perspectives.

2. Preliminary Concepts

In this section we explain gradual patterns mining and property graphs with relevant examples. Later on, we present briefly about the types of missing data and their treatment approaches in literature.

2.1. Gradual Pattern Mining

The objective of gradual pattern mining is to discover frequent co-variations of the form "the more/less the X_i, ..., the more/less theX_n", between two or more 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"*. Gradual item and gradual pattern (aka gradual itemset) defined in [4] are given as below.

Definition 2.1 (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 $\star$.*

Consequently, a gradual pattern is defined as follows:

Definition 2.2 (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.*

Example 2.1. *For example, let us consider the pattern "the higher the age, the higher the number of medications", formalized by the itemsets from Figure 1 are:*

$$P1 = (Age \uparrow\ ,\ Numberof Medications \uparrow)$$

In [4], a precedence graph based algorithm for gradual pattern mining was proposed. In this method, every pattern of size n being processed is associated with an $(n * n)$ binary matrix representing the order relations between tuples regarding that particular pattern. [4] uses vertical bitmap representation presented in [14] to store the ordering in the binary matrix. It states, *"if there exists an order relation between two tuples a and b, then the bit corresponding to the line of a and the column position of b is set to 1, and to 0 otherwise"*. For instance, Table 1 shows the example data of patients. When considering the pattern $Age \uparrow$, we have from this table that $p3(Age30)$ precedes $p2(Age35)$ which precedes $p5(Age55)$, etc. This results in the fact that the value at the intersection between $p3$ and $p2$ in the matrix from Figure 2 (third line, second column) is 1 (in red bold font).

Id	Age	Lab-tests	Time-in-Hospital
p1	45	13	35
p2	35	7	20
p3	30	5	25
p4	55	10	28
p5	40	14	38

Table 1: Patients

	$p1$	**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

(Age $\uparrow$)

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

(Lab-tests $\uparrow$)

Figure 2: Binary Matrices for Gradual Items of Size-1

When considering gradual patterns with several attributes, as for instance $(Age \uparrow, Lab\text{-}tests \uparrow)$, the binary matrices of $(Age \uparrow)$ and $(Lab\text{-}tests \uparrow)$ are mixed to compute the resulting matrix as the Hadamard product. For instance, Figure 3 shows the precedence graph in which $p3$ (Age 30, Lab-tests 5) precedes $p2$ (Age 35, Lab-tests 7) because we both have $30 < 35$ and $5 < 7$ following the pattern $(Age \uparrow, Lab\text{-}tests \uparrow)$ but $p1$ and $p4$ are incomparable as $45 < 55$ but $13 > 10$. This can also be seen in the matrix representation of Figure 4. Indeed, in the matrix the cells [p1][p4] and [p4][p1] are set to 0 which means that there is no order relation defined between $p1$ and $p4$.

The algorithm [5] exploits the rank correlation for gradual pattern extraction. It uses Kendall's tau rank correlation method of concordant and discordant pairs in order to mine gradual pattern. Kendall's tau is one of the most widely used non-parametric test to measure the associa-

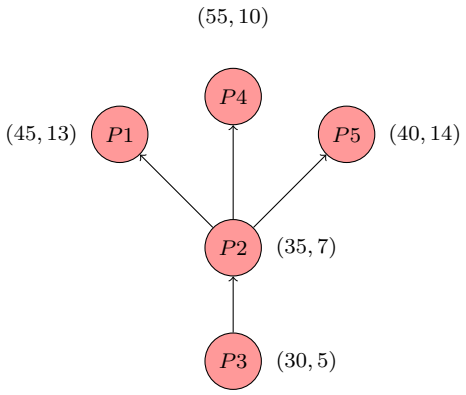


Figure 3: Precedence Graph (Age ↑ , Lab-tests ↑)

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

(Age↑ , Lab-tests↑)

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

(Age↓ , Time-in-hospital↓)

Figure 4: Binary AND of concordant object pairs for gradual patterns

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

(Time-in-hospital ↑)

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

(Time-in-hospital ↓)

Figure 5: Binary matrix representing orders for the dataset shown in Table 1

tion between two variables for rank correlations [15]. The algorithm [5] uses binary matrices for representing orders as shown in Figure 5. It first presents the definition of gradual dependency based on the order concordance [16] in the context of ranking comparison measure and then computes the support. Referring to Theorem 1 in [4] that a gradual pattern P for gradual items e.g., ($i_1 \uparrow$, $i_2 \downarrow$) can be generated, if the following relation holds :

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

The computation of binary matrices representing the set of concordant object pairs for gradual patterns (Age↑ **AND** Lab-tests↑) and (Age↓ **AND** Time-in-hospital↓) is shown in Figure 4. In order to enhance the performance of algorithm [4], a scalable implementation is presented in [10] in which parallelism is exploited by employing multi-core processors for mining gradual patterns.

2.2. Property Graphs

A property graph helps to represent the information and knowledge in a natural way in form of graph-like structure. 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 [17] *"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 [18], Microsoft's Cosmos [19], Titan [20], and Neo4j [21].

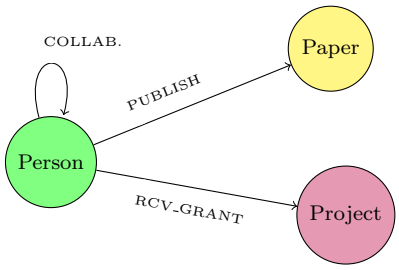


Figure 6: Example Graph Database Schema

Label	Properties	Node Count
Person	id, name, age, desig, expr, sal	7
Paper	id, pr_title, type, year	5
Project	id, prg_title, lab, year	2

Table 2: Nodes Label Summary

Type	Properties	Edge Count
Collaborate	since	14
Publish	date	13
Rec_Grant	date	3

Table 3: Edges Type Summary

Neo4j is an open source NoSQL native graph database engine. Typically, a property graph does not have a predefined schema, therefore, the changes in graph can be incorporated as and when required. To present the definitions, we create a running example property graph in Neo4j as shown in Figure 6. The summary of labels with their properties and node count is given in Table 2 and Table 3 shows the summary about edge types. The definitions for property node, property edge, and property graph introduced in this paper are given as follows.

Definition 2.3 (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 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 2.2. *Λ_N =\{Person, Student, NULL\}, Π_N = \{Age, Expr, Sal\}, n_1 = (2812, \{ Person, Student\}, \{(Age : 32), (Expr : 5), (Sal : 1400)\}) is a property node as shown in Figure 7.*

Definition 2.4 (Property Edge). *Let N be a set of property nodes, Λ_R be a set of edge 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.*

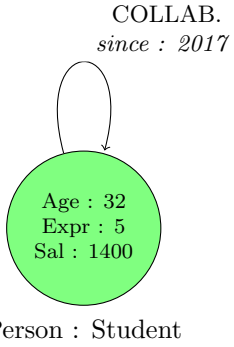


Figure 7: Property Node for Label "Person"

Example 2.3. *Λ_R = \{ PUBLISH, RCV_GRANT, COLAB. \}, Π_R = \{since, NULL\}, r_1 = (4213, \{Person\}, \{Person\}, \{COLAB.\}, \{(since:2017)\} is a property relation as shown in Figure 7.*

Definition 2.5 (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 edge Types with $NULL \in \Lambda_R$, Π_R be a set of properties of edge, 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 edges defined over N, Λ_R and P_R.*

Neo4j uses *"Cypher"*, a declarative query language that uses ASCII-art syntax. Cypher uses specific clauses to perform queries on graph database. An example Cypher query to find a node labeled Person, whose name property value is 'John' and who publishes a journal paper can be written as:

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

Listing 1: Person "John" with Edge "PUBLISH" and Paper

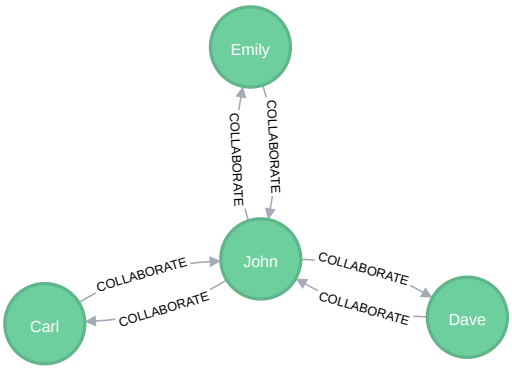


Figure 8: Edges Types Details

The Listing 1 give the details of all papers published by "John" which are of type "journal". To visualize the property graph for edge type "COLLABORATE", we execute the following sample cypher query. The output in Figure 8 shows the "COLLABORATE" edge of user "John" with all users in graph database.

```
1 MATCH (p:Person{name:''John''})-
2         [r:COLLABORATE]->(n:Person)
3 RETURN p,n,r
```

Listing 2: Person "John" with Edge "COLLABORATE"

2.3. Handling Missing Values

Missing data as defined in [22] *"is a statistical problem characterized by an incomplete data matrix"*. The impact of *"missing data bias and error in the reporting of research results"* may vary depending on *"the amount of missing data and the pattern of missing data"* [22]. 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. The types of missing data were first described in [23] 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 the missing data depends on a 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, the 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 reason for missing values in MNAR depends on the missing values themselves, for instance the unobserved data. The example for MNAR can be the people with high income are less likely to report their income [23].

2.3.1. 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.

Ignoring or deleting techniques include list-wise deletion (i.e., ignoring a row/ object which contains missing data) and attribute deletion. Imputation techniques include replacing missing data with fixed constant for instance, replacing with mean of the objects of attribute. Expectation Maximization presented in [24] is a missing data treatment technique in which missing values are estimated (parameter estimation) based on the maximum likelihood method.

We are not using list-wise or pair-wise deletion because in most of the real-case scenarios this could mean removing all the data. Indeed, the properties attached to nodes sharing the same label (e.g. Person) may be different (e.g. "salary and company" on the one side and "school and degree" on the other side). For instance, some nodes may have salary and company information but not school and degree information and vice-versa. In this case, when combining all properties of all nodes, it will result in the fact that no node will have all properties filled in. A list-wise approach will then result in the deletion of all data. Our approach is more parsimonious because it preserves the row as long as possible i.e., even if some attributes are missing, if they are not concerned by the current computation/step, they are disregarded.

2.3.2. 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 [25, 26, 27]. In [28, 29], an association rules based model is used to complete missing data.

The use of graphical models for processing of missing data is presented in [30]. "Graphical models enable an efficient and transparent classification of the missingness"[30] i.e., whether it is MCAR or MAR. The *multiple imputation* techniques such as "Amelia" assume the missingness of data as MAR [31]. The authors in [30] state that "Maximum Likelihood based techniques is a well known estimation method for missing data that require MAR assumption for consistent estimates".

2.3.3. Handling missing data without replacement

[15] 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 has missing values, whereas we consider here the case where single values are missing.

As mentioned in [32], *"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. [33] 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. Hence, the authors redefined the concepts of support and confidence for *vdb* and suggest that these definitions are fully compatible with existing association rule mining algorithms.

3. Problem Statement

NoSQL databases rely on semi-structured data. Due to inherent nature of property graph, a node may not contain all the properties. This data may be missing because it is not available or it does not exist as it may not be applicable (for example no salary, or experience in the case of kids). So, they often contain such missing data where all properties may not be present for nodes and edges. We argue 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) Property graphs are not the same as tabular data in their meaning and application;
(2) Property graphs are semi-structured nature of data.

Therefore, the focus of this work is on handling missing data and extending the existing algorithms to deal with such missing values for extracting gradual patterns. The main steps of our approach are; (i) Retrieve the graph data, (ii) Treat missing data and (iii) Extract gradual patterns.

For mining the gradual patterns, there can be several ways to define them in the context of property graphs such as; *(i) Intra-Node-Label-Properties*, *(ii) Inter-Node-Label -Properties*, *(iii) Node- Properties-with-Edges-count*, and *(iv) Inter-Edge-Properties*. In the paper, we present *(i) and (ii)*. In *Intra-Node-Label-Properties*, the pattern extraction process is performed among the properties/attributes of the "*same label*" nodes. In the *Inter-Node-Label-Properties*, gradual patterns are extracted from different labels. And, in *Node-Properties-with-edges-count*, the gradual patterns are extracted from node properties with the edges count that is retrieved from property graph.

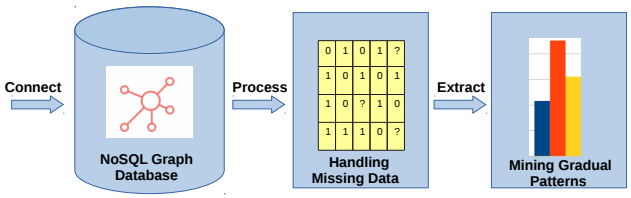


Figure 9: Property Graphs Gradual Pattern Mining Process

4. Mining Gradual Patterns with Missing Values

Mining gradual patterns from property graphs is a novel idea as it involves automatic retrieval of graph data including nodes and edges. It becomes more challenging as the retrieved information often contains missing values due to inherent nature of property graphs. In such a case, the traditional mining techniques can not be applied without treatment of missing values. Therefore, we highlight two novelties of our approach. The first is that no method exists for dealing with missing value when extracting gradual patterns, which is not the same task as regular frequent patterns as the support cannot be computed object by object but requires to consider the ranking of all objects. The second is that we apply this on NoSQL property graphs which are more and more used in real world scalable applications, which is not trivial as all definitions and algorithms must be extended for this particular context.

An overall process flow of gradual pattern mining from a property graph in presence of missing data is shown in Figure 9. The process includes:

(1) Connect to graph database engine and retrieve the graph schema (node labels, edges types and the properties)
(2) Processing the retrieved data from graph for handling missing data
(3) Extract the gradual patterns by computing the support (Section 4.3)

In the first step we retrieve the graph schema from the databases, which means the retrieval of existing node labels, edges types, and their respective count. This information provides a summary of the nodes and edges as well as describes which label is connected with whom and in what frequency (i.e., number of edges). As we know that a property graph is semi-structured and does not have an enforced pre-defined schema, so whenever any deletion or addition of nodes and edges occur in the graph, the data summary for them will also change and the pattern extraction will be performed by accordingly.

```
1 MATCH (Nodes)
2 RETURN labels(Nodes) AS Node_Label,
3 Count (*) AS Node_Count;
```

Listing 3: Graph Node Count with Labels

The Listing 3 shows the details of all the labels for a given property graph with its node count. Similarly, the Listing 4 gives the details about Edges.

```
MATCH () -[ Edge ]-> ()
RETURN type(Edge) AS Edge_Type,
Count (*) AS Edge_Count;
```

Listing 4: Graph Node Count with Labels

Once this data is received, we extract the properties for each label and edge. The process then determines the completeness of data and if the data is missing in properties, then missing data handling takes palace.

In the second step of the process, we consider the approach presented in [33] for treatment of missing values in the context of property graphs. The approach was presented for relational databases to deal with the missing data present in items in order to mine association rules. The approach suggests to cut the dataset into valid database *vdb* i.e., by *partially ignoring missing values* from the data. For instance, the dataset shown in Table 4, with the classical definition of *support*, given the *minimum support threshold* is 40%, the resulting association rules will be:

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

Taking the same *minimum support threshold* of 40% for dataset shown in Table 5, no rules are formed because the support value except $\{X2 = b\}$ for all the remaining items becomes less than 40% as given below:

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

To avoid such a situation due to missing data, the authors [33] propose to partially disable missing data and calculate the valid database for items so as to form the association rules. It is important to recall that for gradual patterns mining we are only interested to compute the support.

In the later part of the section, we explain the *vdb* calculation based on [33] and thereby propose our approach.

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	f
6	b	b	f	d
7	b	c	g	d
8	b	c	h	d

Table 4: Dataset 1 from [33]

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	f
6	b	b	f	?
7	b	c	g	d
8	b	c	h	d

Table 5: Dataset 2 from [33]

The next phase of the process includes mining gradual patterns that involves the matrix representation of items (properties) and support computation as shown in algorithm 1 and 2. To do so, we first begin by defining gradual item in property graph as below.

Definition 4.1 (PG Data 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 edge 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 4.1. *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".*

We explain the concept of *"vdb"* by using the running example of property graph as discussed in Section 2.2. The nodes and edges visualization of property graph is shown in Figure 10. The transformation of label "Person" in tabular representation is given in Table 6. We represent the missing values in "Age" and "Sal" attributes with "?" sign.

The *"vdb"* is the cardinality of an attribute without missing values. For the sake of simplicity, this cardinality is said to be the "cardinality of an attribute". By this, we mean the cardinality of the subset of tuples for which the value on the given attribute is not null. In our case, this attribute is meant to represent a node's property.

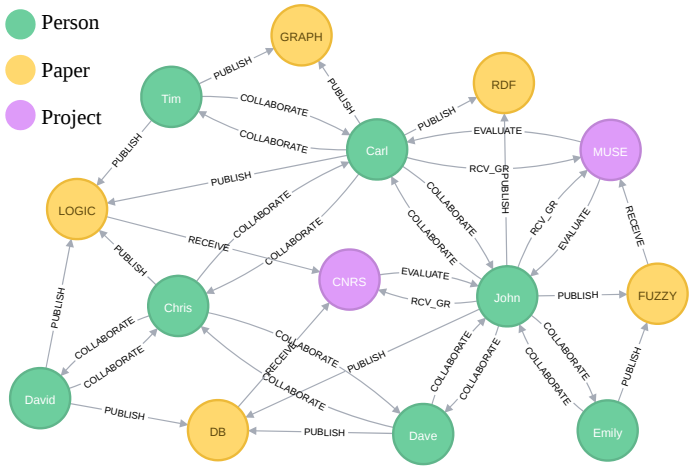


Figure 10: Graph Visualization

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 6: Label "Person" Tabular form

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:

$$|vdb(x)| = |n(x)| - |a_m| \quad (1)$$

Therefore, the *"vdb"* calculation for (Age) is 5 and for (Sal) is 6 as shown in Table 7 and Table 8 respectively.

As discussed in Section 3 that there can be several ways to first define and then mine the gradual patterns in property graphs. For the scope of this paper, we present following two methods:

(1) Intra-Node-Label Gradual Patterns
(2) Node-Properties with Edges Gradual Patterns

We describe the mining process for both of these methods as follows.

4.1. Intra-Node-Label Gradual Patterns

We investigate the scenario of Intra-Node-Label properties for gradual pattern extraction i.e., the pattern extraction process is performed among the properties/attributes of the "same label" nodes. For instance, for a particular label "Person", we try to find the correlation among the properties of nodes. An example of the final extracted pattern can be *"the higher the age, the higher the experience"*. It is important to note that in this scenario there are no edges involved. Considering the label "Person" from graph Figure 10. First, the program retrieves the data from property graph using Noe4j bolt protocol for this label. The response of the Cypher query received from graph database is transformed into tabular representation as shown in Table 6. At this stage, program creates binary order matrix for given properties and then computes the support as described in algorithm [5] for gradual pattern extraction. But, it requires to handle missing values first, because the exiting algorithm does not allow to treat missing data.

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

Table 7: $|vdb\ (Age)|$

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

Table 8: $|vdb\ (Sal)|$

Let us see with an example regarding the missing data and our approach to handle it. 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 [34]. Once the data is received, we perform binary order of matrix as explained in Section 2.1. For example, the binary matrices of $(Age \uparrow)$ and $(Expr \uparrow)$ representing the order are shown in Figure 11. 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 11. 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 12.

When combining several attributes, the value of *vdb* reduces. 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. [33] 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. The *vdb* representations in tabular form when combining attributes are shown in Table 10, and Table 11 respectively. 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

Id	Name	Age	Expr
1	John	45	13
2	Dave	?	7
3	Emily	30	5
4	Tim	?	10
5	Carl	44	14
6	Chris	38	9
7	David	32	5

Table 9: Dataset for Age & 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 11: Binary matrix representing orders for Table 9

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

Figure 12: Hadamard product for Age AND Expr

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 10: $|vdb\ (Age\ \&\ Expr)|$

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 11: $|vdb\ (Age\ \&\ Sal\ \&\ Expr)|$

"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, this *vdb* calculation is an essential part because the value of vdb is then used for support computation (Section 4.3).

4.2. Node-Properties-with-Edges Gradual Patterns

In *"Node-Properties-with-edges- count"* scenario, we try to extract the gradual patterns that involve node properties for a label as well as the corresponding edges 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 collaboration"*.

To perform this mining we first extract graph summaries. *"Graph summarization facilitates the identification of structure and meaning in data"* [35]. There are various summarization techniques depending upon the nature and requirement of the task [7, 35]. These graph summarization techniques include aggregation-based (topology), attribute-based (topology and attributes) [36], application-oriented (e.g., graph pattern matching), and domain specific (e.g., bio-informatics) [7]. In our scenario, program first extracts the graph topology summary including the labels and edges count and further uses that for mining gradual patterns. The output of graph schema summary

SrcLabel	Edge	DstLabel	EdgeCnt
Person	COLLABORATE	Person	14
Person	PUBLISH	Paper	13
Person	RCV_GR	Project	3
Paper	RECEIVE	Project	3
Project	EVALUATE	Person	3

Table 12: Graph Schema Summary with Edge Count

Label "PERSON" Data | **Edges 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 13: Property Graphs Gradual Pattern Mining Process

for running example is shown in Table 12.

After retrieving the graph schema summary, the program extracts summary per label with all its nodes and edges count. The output of this summary in tabular form for label "Person" is shown in Figure 13. From this point the process of handling missing values takes place as shown in Figure 9. After handling missing values, the support computation takes place to find out the gradual pattern that have support greater than the minimum support threshold.

4.3. 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 [37]. 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 4.1. From the resultant matrix, we take the sum of binary '1' bit from the matrices. Consequently, we calculate the support as given below:

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

$$where,\ X_i = itemset$$

In [33], 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:

$$Representativity(X_i) = \frac{|vdb(X_i)|}{|n|} \quad (3)$$

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

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

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

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

$$|Rep(Age\ \&\ Sal\ \&\ Expr)| = 4/7$$

4.4. Algorithm

An overall mining process is shown in Figure 9. Algorithm 1 shows the procedure for computing the support based on the *vdb* method as explained in Section 4.3. After the retrieval of data from the graph data, following are the main steps of Algorithm 1.

- 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.
- 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. 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 11, (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 11 (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 1. If the item's support is less than the minimum support threshold, then it is removed from the list.

Algorithm 1: Mining Gradual Property-based 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
```

Once we have the list of successful gradual items, i.e., those having ($support > minSupport$) the program checks for gradual patterns as defined in Definition 2.2. 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 by 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.

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 [38, 39].

5. Experimental Results

The objective of these experiments is to see the feasibility of mining gradual pattern from property graphs and to compare the proposed approach (Section 4) with imputation of missing values approach. This section is subdivided into three subsections. The first subsection introduces the experimental protocol, the second subsection presents the datasets (synthetic or real data). Finally, we show and discuss the experimental results in the last subsection.

5.1. Experimental Protocol

In this section we briefly describe the program execution environment, the program's logical execution protocol, outlier removal method and memory consumption metrics used to calculate maximum heap utilization.

5.1.1. Environment

All the experiments have been run on the hardware, operating system and software 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

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

```
1: while items in list |L_i| ! = 0 do
2:   for all listItem L_i do
3:     for all listItem L_j = L_i + 1 do
4:       for all rows of matrix M for
         listItem L_i do
5:         for all columns of row of M do
6:           resultM[row][col] =
             L_i.M[row][col]*
             L_j.M[row][col]
7:           if resultM[row][col] == 1
             then
8:             sum ← sum + 1
9:           else if resultM[row][col] == ?
             then
10:            card ← card + 1
11:          end if
12:          if card== resultM.length
             then
13:            flag ← flag + 1
14:          end if
15:        end for
16:      end for
17:      vdb = (L_i resultM.length - flag)
18:      support = sum / (vdb * (vdb-1)/2)

19:      if support < minSupport then
20:        remove L_i
21:      end if
22:    end for
23:  end for
24:  Update the list L
25: end while
26: Output the frequent patterns
```

5.1.2. Experiments execution protocol

The execution protocol describes the steps necessary to produce the results given in the next section.

(1) The java program establishes the connection with graph database using Neo4j's bolt protocol (port 7867)
(2) After the successful connection, the program queries the graph database to retrieve the label's data for the required attributes
(3) Once the attribute data is received, for each attribute the binary matrix initialization is performed as discussed in Section 2
(4) After the matrix initialization and concordant object pairs AND operation, the *vdb* computation is performed as discussed in Section 4
(5) An executable is created and run 5 times. The average value of each metric is used (outliers are removed, as discussed in the next subsection)

The source code and output files are available at git [40]. The program takes as parameter the min_support.

5.1.3. Outlier Removal

Each data point is plotted by taking the average of 5 executions. 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 [41].

5.1.4. Memory Consumption Metric

Choosing a metric for memory consumption is an essential task. Java has both heap and non-heap memory. JVM consider memory pools of type heap or non-heap [42]. For the memory consumption we take peak used size of all the 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. Details of the source code and its usage is given at [40].

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 garbage collector has the unique ability to efficiently and concurrently deal with very large heaps. It can also be configured not to exceed a maximum pause time. It can so easily outperform other state-of-the-art GCs on large heaps that Garbage First (G1). GC is used as default garbage

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

Table 13: Intra-Node Gradual Pattern Mining

S#	Dataset	N	R	P	RT	M
1	Synthetic Graph	15,000	999,268	6	3	10
2	Hetionet(Gene)	20,945	1,289,190	1	4	1

N = Nodes , R = edges, P = Node Properties,
RT= edges Type, M = Missing Values %

Table 14: Nodes with Edges Gradual Pattern Mining

collector in Java 9. We also set the *BiasedLockingStartupDelay* to 0 for improving performance of unaccounted synchronization and we have tuned the Xmx parameter to fix the maximum size of memory allocation pool. To summarize, the jvm flags as arguments for program execution are; -XX:+UseG1GC, -XX:BiasedLockingStartupDelay = 0, and -Xmx10G for intra-node and -Xmx12G for nodes with edges datasets.

5.2. Datasets description

The experiments were performed on two different types of datasets for intra-node gradual pattern mining and nodes with edges gradual pattern mining as shown in Table 13 and Table 14 respectively. It is worth noticing that the proposed approach may be applied to numerical ordinal attributes only. Therefore, in case of intra-node gradual pattern mining, a pre-processing step is performed to opt the attributes of datasets that are numerical. For all these, we have created separated graph databases and run the program executions accordingly, with the exception of Hetionet dataset. The Hetionet is a real graph database which is publicly available at [43].

In each result, for the missing data imputation method, we made a separate 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 [31] for multiple imputation of missing data. The complete details of source code is available at gitlab [40]. The rest of this subsection is devoted to describe more details about each dataset.

5.2.1. The Russian_tweet_troll

The Russian_tweet_troll is a real dataset taken from [44]. This dataset contains more than 200, 000 tweets that Twitter has tied to "malicious activity" from Russia-linked accounts during the 2016 U.S. presidential election.

User account file contains 14 attributes but we have selected 7 relevant attributes (userId, followersCount, statusesCount, timeZone, favouritesCount, friendsCount, and listedCount) for our experiments. This data is used to create a graph DB using cypher import utility. The data is imported using the following command:

```
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: Cypher import command for Russian-tweet-troll

5.2.2. Hepatitis from UCI Machine Learning Repository

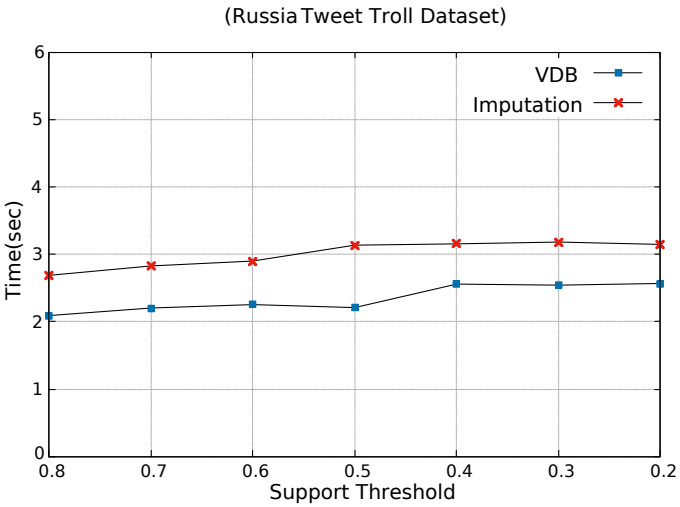
Hepatitis dataset is taken from UCI machine learning repository [45]. For this dataset, there are 20 attributes, but we consider only 6 relevant attributes (BILIRUBIN, ALKphosphate, PROTIME, ALBUMIN, Age, and SGOT]) that have been imported to graph database. Attributes selection is made considering the numerical attributes requirement of the approach.

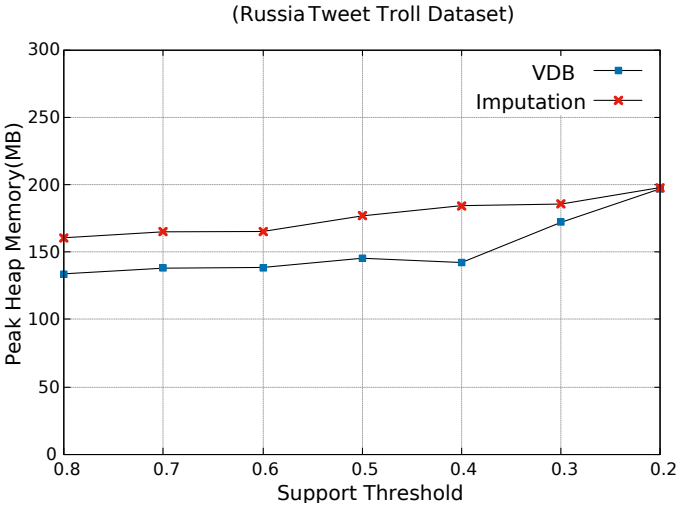
```
LOAD CSV WITH HEADERS FROM
      "file:///hepatitis.csv" AS row
MERGE
      (:Hepatitis {Age:row.Age,
      BILIRUBIN:row.BILIRUBIN,
      ALKphosphate:row.ALKphosphate,
      SGOT:row.SGOT,
      ALBUMIN:row.ALBUMIN,
      PROTIME:row.PROTIME})
```

Listing 6: Cypher import command for Hypatitis dataset

5.2.3. 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 [40]. We generated the dataset for 10, 000 nodes and 5 attributes. 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 given below.





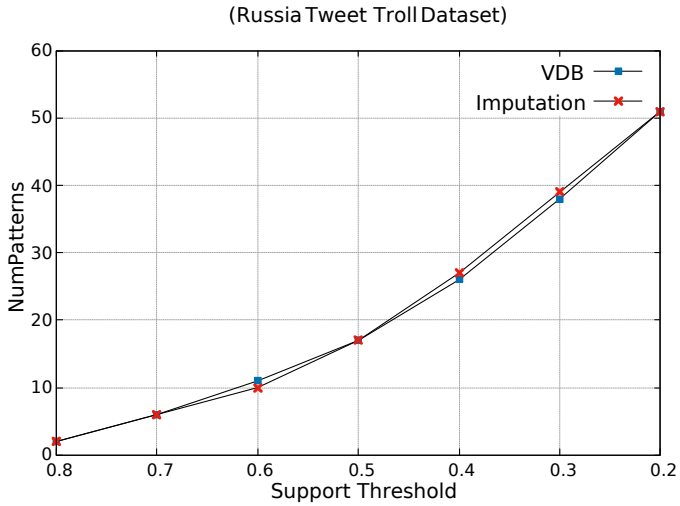
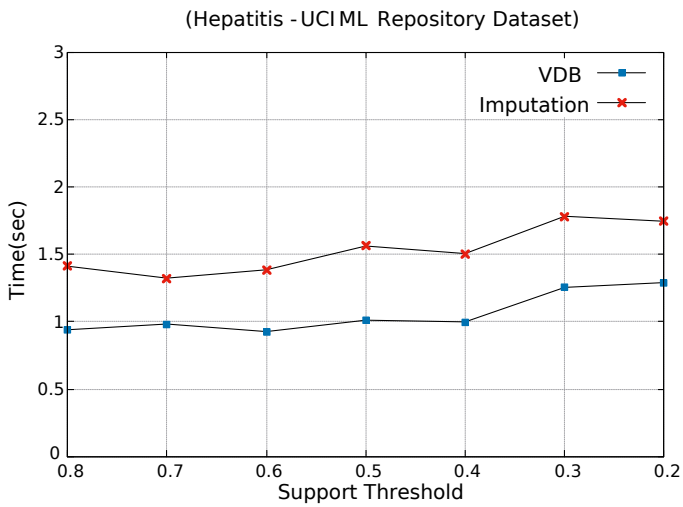
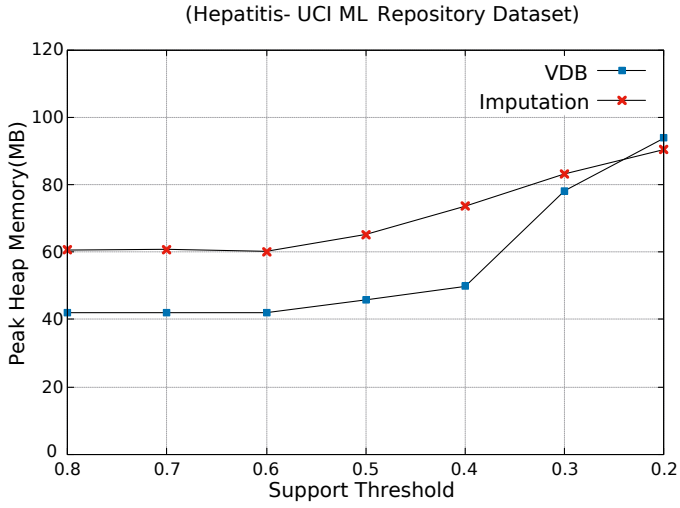


Figure 14: Russia Tweet troll Dataset





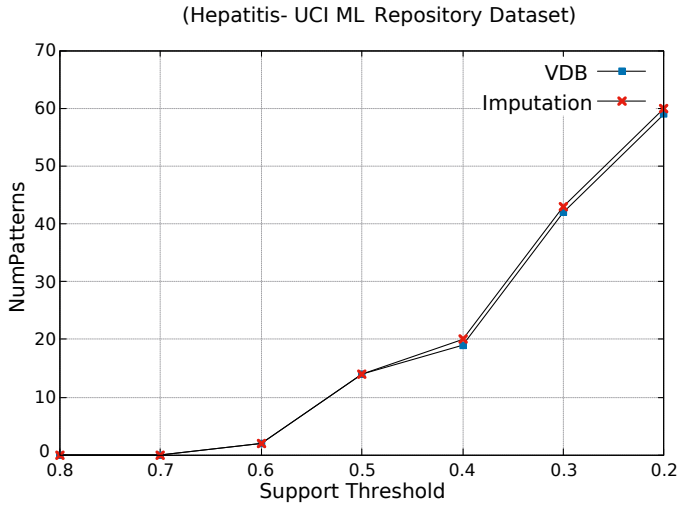


Figure 15: Hepatitis - UCI ML Repository Dataset

```
LOAD CSV WITH HEADERS FROM
    "file:///genData10K.csv" AS row
MERGE
    (:Synthetic {Id:row.Id,
    A99RO:row.A99RO,
    B99RO:row.B99RO,
    C99RO:row.C99RO,
    D99RO:row.D99RO,
    E99RO:row.E99RO})
```

Listing 7: Cypher import command for Synthetic dataset

5.2.4. Synthetic Graph Dataset

We have developed a synthetic graph generator to make property graphs in Neo4j database. This graph generator's web interface is available at [46]. The synthetic graph dataset contains 3 labels namely Label1, Label2, Label3 and 3 edge types R1, R2, and R3 respectively. The nodes and edges details are shown in Table 14. 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 created Neo4j databases and complete source code is available at [40]. The output plots are shown in Figure 17.

5.2.5. Hetnets in bio-medicine

Hetnet is a research outcome published in [47], which shows a network of biology, disease, and pharmacology. It is an integrated network of nodes and edges in which "the knowledge from millions of biomedical studies over the last half century have been encoded into this single system" [47]. The details of the project and its relevant github resource is available at [48].

In this dataset, we choose the "Gene" label and its edges. 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 edges with other labels like chemical compound, biological process, etc. There are 7 edge types of "Gene" Label in Hetionet. The summary of its edges with other labels is shown in Table 15. The dataset has a total of 47,031 nodes and 2,250,197 edges. In the Gene label, there are 20,945 nodes and 1,289,190 edges with 7 edge types. For our experiments, we used all the nodes with 4 edge types that connect with labels other than Gene itself like, PARTICIPATES_GpBP, PARTICIPATES_GpMF, PARTICIPATES_GpPW and PARTICIPATES_GpCC. The output plots are shown in Figure 18.

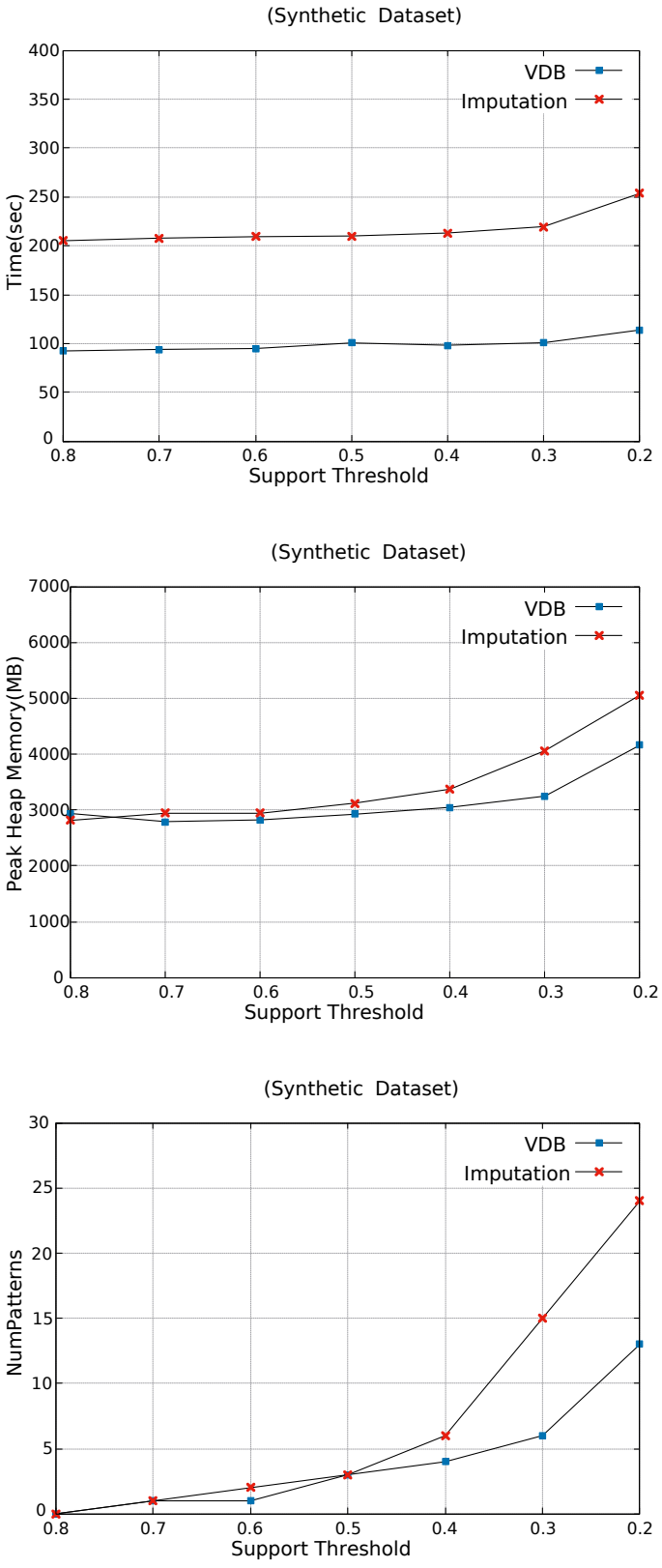
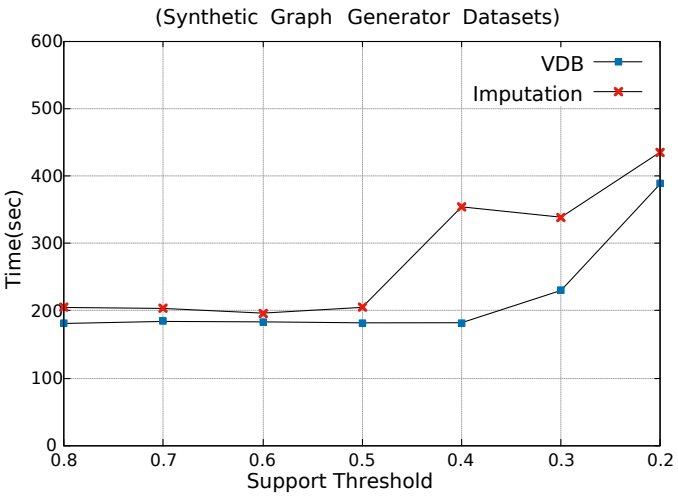
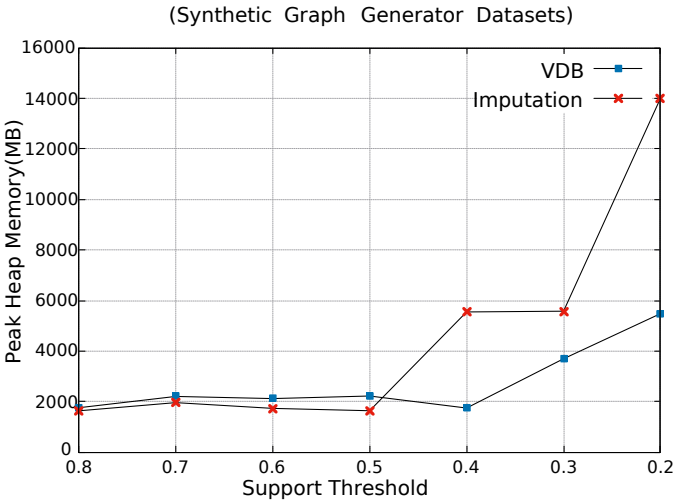


Figure 16: Synthetic Dataset

5.3. Discussion

It is found from the results of both types of experiments i.e., intra-nodes and nodes with edge that the percentage





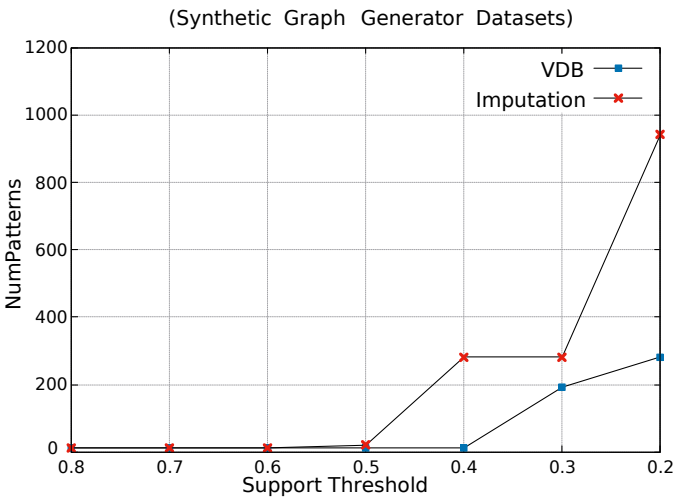
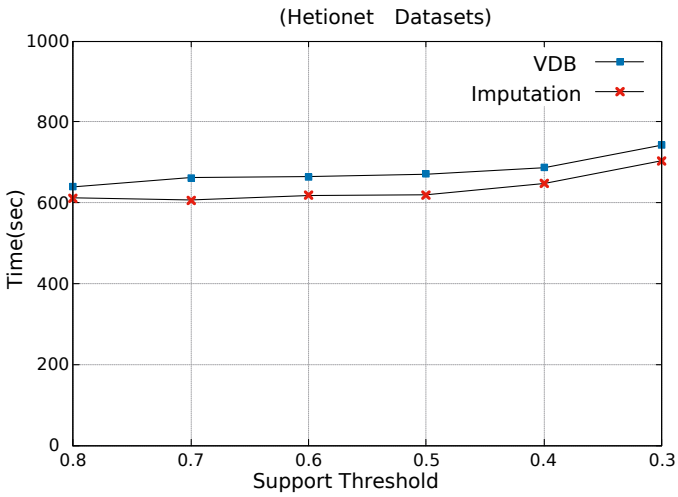
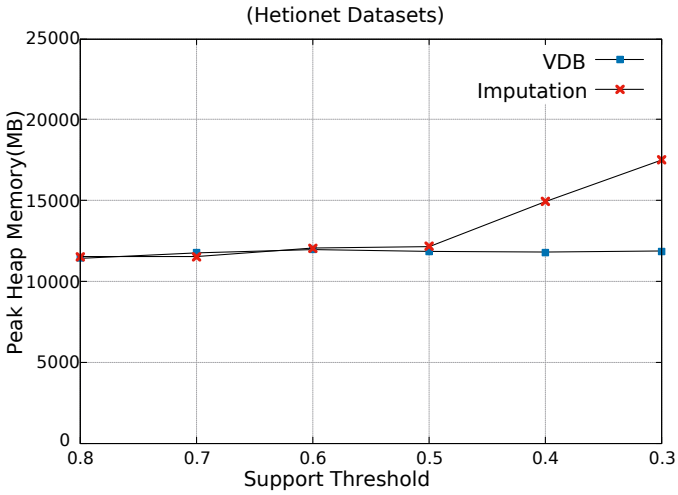


Figure 17: Synthetic Graph Dataset





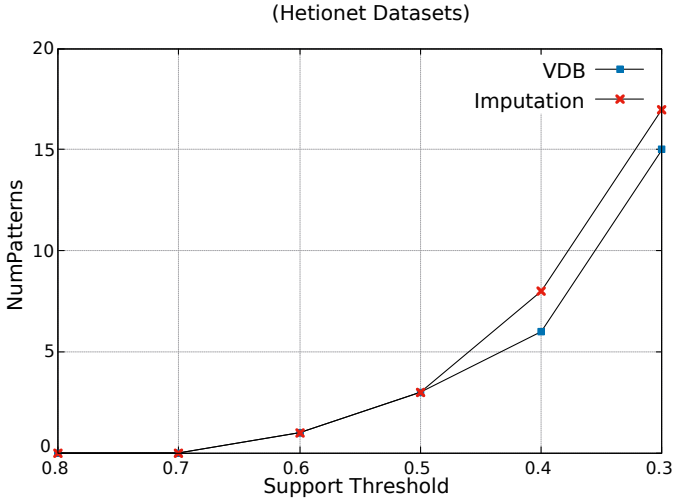


Figure 18: Hetionet(Gene) Dataset

Src	edge	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 15: Hetionet "Gene"edges Summary

of missing data has an impact on the total number of extracted patterns. That is, with higher missing data percentage in a dataset, we get more patterns. In the case of synthetic dataset 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↑). Similarly, for Hepatitis dataset at support threshold of 0.2, (BILIRUBIN ↑, Age↑, SGOT↓) is an excessive pattern. Also, there were some patterns in imputation methods that were not present in vdb such as, (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 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 there 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 14. The same fact is visible in Figure 15 for Hepatitis dataset. Whereas, in Synthetic dataset plot Figure 16, 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 nodes with edges scenario for the datasets shown in Table 14, we observed the similar fact that with more percentage of missing data the imputation method generates more patterns as shown in numPatterns in Figure 17. 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. The program's time utilization results also show that time difference is almost doubles between both methods in all the three datasets of intra-node experiments. whereas, the time utilization in case of node with edges scenario is almost same as observed in Figure 17 and Figure 18 respectively.

It is deduced form the results that when the percentage of missing values is higher the imputation method generates more pattern than vdb approach and these excessive patterns might now reflect to the reality. Because the data is artificially imputed in missing places, the imputation method generates more co-variance for the objects that are not actually present.

6. Conclusion

In this article, we are dealing with mining gradual patterns from NoSQL graph databases, which raises the problem of dealing with missing data. We propose a new approach based on valid databases and show the performance comparison with imputation method.

Further works include the integration of our proposal in the database engine. To achieve this goal, we aim at using GraphX Apache Spark's API for graphs and graph-parallel computation. Indeed, one of our further work is to provide a scalable distributed implementation of gradual pattern mining algorithm compatible with the use of missing values.

Another field of research is to explore the other scenarios for gradual patterns from property graphs like, *Inter-Node-Label-Properties* i.e., gradual patterns are to be extracted from the properties of nodes with different labels and *Inter-Edge-Properties* i.e., gradual patterns are to be extracted from the properties of different edge types.

References

References

[1] C. C. Aggarwal, An introduction to data mining, in: Data Mining: The Textbook, Springer International Publishing, 2015, pp. 1–26 (2015). doi:10.1007/978-3-319-14142-8_1.

[2] C. C. Aggarwal, M. A. Bhuiyan, M. Al Hasan, Frequent pattern mining algorithms: A survey, in: Frequent pattern mining, Springer, 2014, pp. 19–64 (2014).

[3] E. Hüllermeier, Association rules for expressing gradual dependencies, in: Proc. of PKDD'02, 2002, pp. 200–211 (2002).

[4] L. Di-Jorio, A. Laurent, M. Teisseire, Mining frequent gradual itemsets from large databases, in: N. M. Adams, C. Robardet, A. Siebes, J. Boulicaut (Eds.), Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proceedings, Vol. 5772 of Lecture Notes in Computer Science, Springer, 2009, pp. 297–308 (2009). doi:

10.1007/978-3-642-03915-7_26.
URL https://doi.org/10.1007/978-3-642-03915-7_26

[5] A. Laurent, M. Lesot, M. Rifqi, GRAANK: exploiting rank correlations for extracting gradual itemsets, in: T. Andreasen, R. R. Yager, H. Bulskov, H. Christiansen, H. L. Larsen (Eds.), Flexible Query Answering Systems, 8th International Conference, FQAS 2009, Roskilde, Denmark, October 26-28, 2009. Proceedings, Vol. 5822 of Lecture Notes in Computer Science, Springer, 2009, pp. 382–393 (2009). doi:10.1007/978-3-642-04957-6_33.
URL https://doi.org/10.1007/978-3-642-04957-6_33

[6] B. R. Bebee, D. Choi, A. Gupta, A. Gutmans, A. Khandelwal, Y. Kiran, S. Mallidi, B. McGaughy, M. Personick, K. Rajan, S. Rondelli, A. Ryazanov, M. Schmidt, K. Sengupta, B. B. Thompson, D. Vaidya, S. Wang, Amazon neptune: Graph data management in the cloud, in: M. van Erp, M. Atre, V. López, K. Srinivas, C. Fortuna (Eds.), Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th to 12th, 2018., Vol. 2180 of CEUR Workshop Proceedings, CEUR-WS.org, 2018 (2018).
URL http://ceur-ws.org/Vol-2180/paper-79.pdf

[7] A. Khan, S. S. Bhowmick, F. Bonchi, Summarizing static and dynamic big graphs, Proc. VLDB Endow. 10 (12) (2017) 1981–1984 (Aug. 2017). doi:10.14778/3137765.3137825.
URL https://doi.org/10.14778/3137765.3137825

[8] S. Ayouni, S. Ben Yahia, A. Laurent, P. 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 (2010).
URL https://hal-lirmm.ccsd.cnrs.fr/lirmm-00798797

[9] M. Quintero, A. Laurent, P. Poncelet, Fuzzy orderings for fuzzy gradual patterns, in: International Conference on Flexible Query Answering Systems, Springer, 2011, pp. 330–341 (2011).

[10] T. D. T. Do, A. Termier, A. Laurent, B. Négrevergne, B. Omidvar-Tehrani, S. Amer-Yahia, PGLCM: efficient parallel mining of closed frequent gradual itemsets, Knowl. Inf. Syst. 43 (3) (2015) 497–527 (2015). doi:10.1007/s10115-014-0749-8.
URL https://doi.org/10.1007/s10115-014-0749-8

[11] S. Ayouni, S. B. 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 (2014).

[12] A. Laurent, B. Négrevergne, N. Sicard, A. Termier, Efficient parallel mining of gradual patterns on multicore processors, in: Advances in Knowledge Discovery and Management, Springer, 2012, pp. 137–151 (2012).

[13] T. Ngo, V. Georgescu, A. Laurent, T. Libourel, G. 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 (2018).

[14] J. Ayres, J. Flannick, J. Gehrke, T. 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, ACM, New York, NY, USA, 2002, pp. 429–435 (2002). doi:10.1145/775047.775109.
URL http://doi.acm.org/10.1145/775047.775109

[15] T. Papaioannou, S. Loukas, Inequalities on Rank Correlation with Missing Data, Journal of the Royal Statistical Society Series B (Methodological) 46 (1) (1984) 68–71 (1984).

[16] F. Berzal, J.-C. Cubero, D. Sanchez, M.-A. Vila, J. M. Serrano, An alternative approach to discover gradual dependencies, Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 15 (5) (2007) 559–570 (2007).

[17] S. Sakr, G. Al-Naymat, Querying graph databases: An overview, in: Advanced Database Query Systems: Techniques, Applications and Technologies, IGI Global, 2011, pp. 304–322 (2011). doi:10.4018/978-1-60960-475-2.ch013.

[18] Amazon, Amazon neptune: Fast, reliable graph database built for the cloud, page accessed: Jan 2019.
URL https://aws.amazon.com/neptune/

[19] Microsoft, Azure cosmos db: Globally distributed, multi-model database service, page accessed: Jan 2019.
URL https://azure.microsoft.com/en-gb/services/cosmos-db/

[20] T. Aurelius, Titan: Distributed graph database, page accessed: Jan 2019.
URL http://titan.thinkaurelius.com/

[21] Neo4j, Neo4j - the fastest path to graph success, page accessed: Jan 2019.
URL https://neo4j.com/

[22] D. A. Newman, Missing data: Five practical guidelines, Organizational Research Methods 17 (4) (2014) 372–411 (2014). doi:10.1177/1094428114548590.
URL https://doi.org/10.1177/1094428114548590

[23] D. B. RUBIN, Inference and missing data, Biometrika 63 (3) (1976) 581–592 (1976). arXiv:/oup/backfile/content_public/journal/biomet/63/3/10.1093/biomet/63.3.581/2/63-3-581.pdf, doi:10.1093/biomet/63.3.581.
URL http://dx.doi.org/10.1093/biomet/63.3.581

[24] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the em algorithm, Journal of the Royal Statistical Society. Series B (Methodological) 39 (1) (1977) 1–38 (1977).
URL http://www.jstor.org/stable/2984875

[25] G. E. A. P. A. Batista, M. C. Monard, An analysis of four missing data treatment methods for supervised learning, Applied Artificial Intelligence 17 (2003) 519–533 (2003).

[26] E. R. Hruschka, E. R. Hruschka, N. F. F. Ebecken, Evaluating a nearest-neighbor method to substitute continuous missing values, in: T. T. D. Gedeon, L. C. C. Fung (Eds.), AI 2003: Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 723–734 (2003).

[27] W. Shahzad, Q. Rehman, E. Ahmed, Missing data imputation using genetic algorithm for supervised learning, International Journal of Advanced Computer Science and Application (IJACSA) 8 (2) (2017) 438–445 (feb 2017).

[28] C.-H. Wu, C.-H. Wun, H.-J. Chou, Using association rules for completing missing data, in: Hybrid Intelligent Systems, 2004. HIS '04. Fourth International Conference on, 2004, pp. 236–241 (Dec 2004). doi:10.1109/ICHIS.2004.91.

[29] L. Ben Othman, F. Rioult, S. Ben Yahia, B. Crémilleux, Missing values: Proposition of a typology and characterization with an association rule-based model, in: T. B. Pedersen, M. K. Mohania, A. M. Tjoa (Eds.), Data Warehousing and Knowledge Discovery, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 441–452 (2009).

[30] K. Mohan, J. Pearl, Graphical models for processing missing data, arXiv:1801.03583v1 [stat.ME] (2018) 1–34 (jan 2018).

[31] J. Honaker, G. King, M. Blackwell, Amelia ii: A program for missing data, Journal of Statistical Software 45 (7) (2011) 1–47 (2011).

[32] T. Orchard, M. A. Woodbury, A missing information principle: theory and applications, in: Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, 1972, pp. 1: 697–715 (1972).

[33] A. Ragel, B. Crémilleux, Treatment of missing values for association rules, in: X. Wu, K. Ramamohanarao, K. B. Korb (Eds.), Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD-98, Melbourne, Australia, April 15-17, 1998, Proceedings, Vol. 1394 of Lecture Notes in Computer Science, Springer, 1998, pp. 258–270 (1998). doi:10.1007/3-540-64383-4.
URL https://doi.org/10.1007/3-540-64383-4

[34] A. Bowman, Understanding non-existent properties and working with nulls, page accessed: Jan 2019.
URL https://neo4j.com/developer/kb/understanding-non-existent-properties-and-null-values/

[35] Y. Liu, T. Safavi, A. Dighe, D. Koutra, Graph summarization methods and applications: A survey, ACM Comput. Surv.

51 (3) (2018) 62:1–62:34 (Jun. 2018). doi:10.1145/3186727.
URL http://doi.acm.org/10.1145/3186727

[36] Y. Wu, Z. Zhong, W. Xiong, N. Jing, Graph summarization for attributed graphs, in: 2014 International Conference on Information Science, Electronics and Electrical Engineering, Vol. 1, 2014, pp. 503–507 (April 2014). doi:10.1109/InfoSEEE.2014.6948163.

[37] T. Calders, B. Goethals, M. Mampaey, Mining itemsets in the presence of missing values, in: Proceedings of the 2007 ACM Symposium on Applied Computing, SAC '07, ACM, New York, NY, USA, 2007, pp. 404–408 (2007). doi:10.1145/1244002.1244097.

[38] G. 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 (2004). doi:10.1145/1014052.1014091.
URL https://doi.org/10.1145/1014052.1014091

[39] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, Data Mining and Knowledge Discovery 15 (1) (2007) 55–86 (Aug 2007).

[40] F. Shah, Gradual pattern extraction- java code, page accessed: Jan 2019.
URL https://gite.lirmm.fr/shah/handlingmissingdata

[41] B. Narasimhan, The normal distribution, page accessed: Jan 2019.
URL http : / / statweb . stanford . edu / ~naras / jsm / NormalDensity/NormalDensity.html

[42] oracle, Java memory types (2018).
URL https://docs.oracle.com/javase/7/docs/api/java/lang/management/MemoryPoolMXBean.html

[43] D. Himmelstein, Hetnets in neo4j, page accessed: Jan 2019.
URL https://neo4j.het.io/browser/

[44] NBC-News, Russian troll tweets (2018).
URL https : / / www . nbcnews . com / tech / social-media / now-available-more-200-000-deleted-russian-troll-tweets-n844731

[45] UCI-Machine-Learning-Repository, Hepatitis data set (1988).
URL https://archive.ics.uci.edu/ml/datasets/hepatitis

[46] F. Shah, Synthetic graph generator, page accessed: Jan 2019.
URL https://faaizshah.github.io/graphgenerator/

[47] D. S. Himmelstein, A. Lizee, C. Hessler, L. Brueggeman, S. L. Chen, D. Hadley, A. Green, P. Khankhanian, S. E. Baranzini, Systematic integration of biomedical knowledge prioritizes drugs for repurposing, eLife 6 (2017) e26726 (sep 2017). doi:10.7554/eLife.26726.
URL https://doi.org/10.7554/eLife.26726

[48] S. B. Daniel Himmelstein, Hetnets in biomedicine, page accessed: Jan 2019.
URL https://het.io/