LIRMM, CNRS - University Montpellier 2, France
University of Science and Technology of Hanoi, Vietnam

Final Master Report

MASTER IN INFORMATION AND COMMUNICATION TECHNOLOGY

SUMMARIZING NoSQL GRAPH DATABASES

Student:
Bui Dinh Duong
dinhduong.bui@gmail.com
Intake 2012-2014

Supervisors:
Dr. Arnaud Castelltort
Prof. Thérèse Libourel
Prof. Anne Laurent
University Montpellier 2

Tutor:
Prof. Muriel Visani
University of La Rochelle

September 2014

Acknowledgement

First of all, I would like to thank Prof. Anne Laurent, Prof. Thérèse Libourel, and Dr. Arnaud Castelltort for their guidance and supervision during my internship research. I am grateful to all my supervisors and colleagues at LIRMM for pointing out my errors, giving me some advice concerning both theory and practical works in this report.
I would like to thank Prof. Muriel Visani, University of La Rochelle, my tutor during the internship.
I also want to thank the University of Science and Technology of Hanoi (USTH), the university where I had a big chance to study with all professors, who taught me during two years of master with amazing courses. USTH has given me a lot of knowledge and vision in my career.
Finally, I want to thank my family, who always stays beside and shares with me all things in the life.

Contents

List of Tables

List of Figures

Abstract

Graph databases are spreading over many fields and have been recently put on the foreground with the emergence of the so called NoSQL graph databases that allow to deal with huge volumes of data (e.g., scientific and social networks). Such graphs often evolve over time and are often very large. It is thus important to provide the users with summaries. Data Summarization is a topic that has been addressed by many authors. In this internship, we consider linguistic summaries of the form *Qy are a*, as for instance . We have worked about the topic of NoSQL graph data summarization which had not been explored before. Three types of summaries have been proposed, namely Structure-Summaries, DataStructure-Summaries and Temporal DataStructure-Summaries. This proposition has been implemented and tested by using the Neo4j NoSQL graph databases engine. The results show the interest of our propositions and have opened many perspectives.

Chapter 1

Introduction

My internship has taken place at the LIRMM laboratory in Montpellier, France. I have worked on Data Summarization for NoSQL Graph Databas, Neo4j.

In this section, I will first present the LIRMM lab and then introduce the scientific objectives of my work.

1.1 LIRMM

The Montpellier Laboratory of Computer Science, Robotics, and Microelectronics (Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, LIRMM) is a cross-faculty research entity of the University of Montpellier 2 (UM2) and the National Center for Scientific Research (CNRS) - Department of Information and Engineering Sciences and Technologies (IEST) (Wikipedia, 2014). These activities are conducted mainly within the three scientific research departments:

- Computer Science (INFO)
- Microelectronics (MIC)
- Robotics (ROB)

LIRMM research activities revolve around information and communication sciences and technologies, reaching from circuit design, to modeling of complex agent-based systems, to algorithmic studies, bio-computing, human-computer interfaces, and robotics (Wikipedia, 2014). With its national and international scientific influence, LIRMM is a key regional economic organization, with an important innovation and technology transfer activity (lirmm.fr).

1.2 Scientific objective

Graphs are known to be efficient for representing data in many applications, from linguistics to chemistry and social networks. For instance, such a graph allows to draw in a very intuitive manner the relationships among people and between these people and the organizations they belong to.

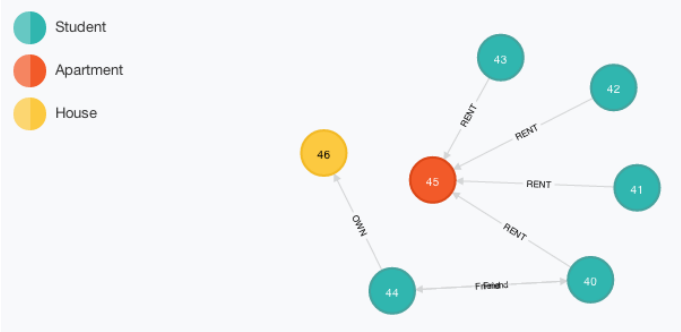

Figure 1.1: Example of a Graph Database

For instance, Fig. 1.1 displays the relations between people and the places they live in. The relation types depict whether they own or rent their housing. The form of housing can be apartment or house, this form being stored as the node type.

Graphs are recognized to play an important role within the pattern recognition field [CFSV04], thus being a key technology for retrieving relevant information, as for fraud detection or in social/biological interactions. Techniques and algorithms can be distinguished depending on the fact that they are meant to mine relevant patterns (data mining) or to retrieve patterns (queries).

Graph databases are specific and cannot always easily be represented as relational data. Comparisons have been made and the current most used databases for managing huge volumes of graph databases is the NoSQL graph database model. Neo4j is the leading commercial graph NoSQL database engine.

The data is increasing very fast, and graph database can meet the requirement of storing and querying huge volume of data. And also, the field of data mining and knowledge discovery become very popular and has been developing rapidly. Many powerful data mining and knowledge discovery techniques are available but they still require a constant human supervision and, in addition, they are not human consistent enough as they practically do not use natural language which is the only fully natural means of communication human beings.

Thus, Summarizing data helps people to automatically sum up the large dataset in similar human language. In in our work, we try to sum up a graph database. For instance, from the database Fig. 1.1, we try to extract summary such as: "Most students follow the schema (Student)-[rents] $\rightarrow$ (Appartement)", where *Most* is a fuzzy modifier.
This summary is closed to the human language. However, it is not so easy to define such summaries on graphs.

The objectives of my work were the following ones:

- Getting familiar with NoSQL graph databases, especially with Neo4j;
- Getting familiar with data summarization;
- Defining NoSQL graph data summaries;
- Proposing queries and algorithms to automatically extract such summaries;

- Implementing and testing on synthetic and real databases such methods;
- Proposing perspectives.

Chapter 2

Materials and Methods

For achieving my work, I have worked on two main topics. The first topic is NoSQL graph databases and the second one is data summarization. These two topics have been the basis for building an innovative solution to summary NoSQL graph databases. They are thus presented in this section.

2.1 NoSQL graph databases

NoSQL is the new generation of database systems which can deal with very big data scalability. The data now is also very complicated. A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the table relations used in relational databases. On the other hand, NoSQL databases provides a schema-less data structure.

2.1.1 Basic Definitions

Graphs have been studied for a long time by mathematicians and computer scientists. A graph can be directed or not. It is defined as follows.

Def 1 (Graph) *A graph G is given by a pair (V, E) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq (V \times V)$.*

Def 2 (Directed Graph) *A directed graph G is given by a pair (V, E) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq \{V \times V\}$. That is E is a subset of all ordered permutations of V element pairs.*

When used in real world applications, graphs need to be provided with the capacity to label nodes and relations, thus leading to the so-called labeled graphs, or property graphs.

Def 3 (Labeled Oriented Graph) *A labeled oriented graph G, also known as oriented property graph, is given by a quadruplet (V, E, α, β) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq \{V \times V\}$, α stands for the set of properties defined over the nodes, and β the set of properties defined over the relations.*

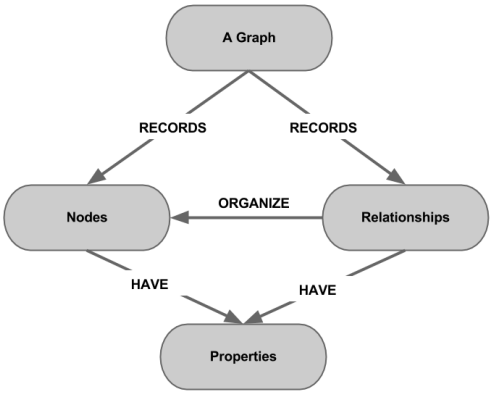

Figure 2.1: Labeled Graph

NoSQL graph databases [RWE13a] are both based on graph and NoSQL concepts.

There are four types of NoSQL database: Document, Column, Key-value and Graph. Graph database stores data in nodes, relationships and properties, as compared in Table 2.1.

In this model, graph objects (nodes and relations) are dealt with. Complex labels on both nodes and relations are managed with $(key, value)$ pairs. Nodes and relations can be stamped with a *type*or *label*.

Operations can be defined for reading and writing data. NoSQL graph databases provide users with several ways to query, whether it be at the programmation level, via API, or though a declarative query language (cypher for Neo4j).

Fig. 2.2 shows a graph and its structure in $(key, values)$ pairs.

In this work, we call L the set of labels and types of a graph, which is thus defined by $G = (V, E, \alpha, \beta, L)$.

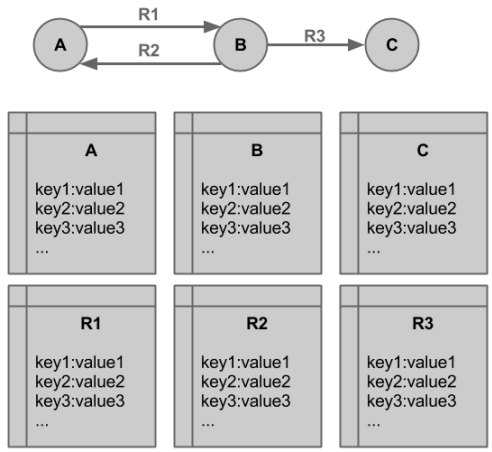

Figure 2.2: Properties of Nodes and Relations

Studies have shown that these technologies present good performances, much better than classical relational databases for representing and querying such large graph databases. Their exist several NoSQL graph database engines (OrientDB, Neo4J,HyperGraphDB, etc.) [AG08].

Database	Column	Document	Key-Value	Graph
Amazon SimpleDB	No	No	Yes	No
BaseX	No	Yes	No	No
Cassandra	Yes	Yes	No	No
CouchDB	No	Yes	No	No
Google Datastore	Yes	No	No	No
HBase	Yes	No	No	No
MemcacheDB	No	No	Yes	No
MongoDB	No	Yes	No	No
Neo4j	No	No	No	Yes
Redis	No	Yes	Yes	No

Table 2.1: Examples of NoSQL Databases

Neo4J is recognised as being one of the top ones regarding performance [Tho13]. Neo4j is an open-source graph database, implemented in Java. It was first released in 2007 by Neo Technology. Neo4j is the most popular graph database [Vai13]. Neo4j provides a robust, scalable and high-performance database, Neo4j is suitable for full enterprise deployment or a subset of the full server can be used in lightweight projects [Neu10]. It also provides both server-side and embedded mode in application for development.
It features:

- True ACID transactions
- High availability
- Scales to billions of nodes and relationships
- High speed querying through traversals

2.1.2 Querying NoSQL Graph Databases

All NoSQL graph databases require the developers and users to use graph concepts to query data. As for any other repository, when querying such NoSQL graph databases, users either require specific focused knowledge (e.g., retrieving Peter's friends) or ask for trend detection (e.g., detecting trends and behaviours within social networks).

Queries are called *traversals*. A graph traversal refers to visiting elements, *i.e.* nodes and relations. There are three main ways to traverse a graph:

- programmaticaly, by the use of an API that helps developers to operate on the graph;
- by functional traversal, a traversal based on a sequence of functions applied to a graph;
- by declarative traversal, a way to explicit what we want to do and not how we want to do it. Then, the database engine defines the best way to achieve the goal.

In this paper, we focus on declarative queries over a NoSQL graph database. The Neo4j language is called Cypher.

For instance on Fig. 2.3, one query is displayed to return the customers who have visited the "Ritz" hotel.They are both displayed in the list and circled in red in the graph.

We consider in this report the Cypher language.

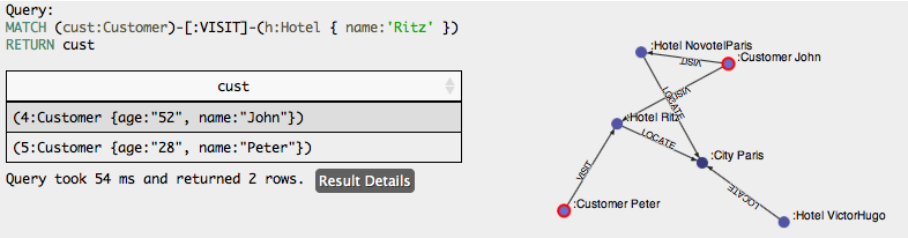

Figure 2.3: Displaying the Result of a Cypher Query

Cypher clauses are similar to SQL ones. It is based on a "ASCII art" way of writing graph elements. For example, directed relations are written using the $-[]->$ symbol. Types and labels are written after a semi-column (:).

More specifically, queries in Cypher have the following syntax[1]:

```
[START]
[MATCH]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

As shown above, Cypher is comprised of several distinct clauses:

- START: Starting points in the graph, obtained via index lookups or by element IDs.
- MATCH: The graph pattern to match, bound to the starting points in START.
- WHERE: Filtering criteria.
- RETURN: What to return.
- CREATE: Creates nodes and relationships.
- DELETE: Removes nodes, relationships and properties.
- SET: Set values to properties.
- FOREACH: Performs updating actions once per element in a list.
- WITH: Divides a query into multiple, distinct parts.

[1]http://docs.neo4j.org/refcard/2.0/
http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html

2.1.3 Historical in Graph Databases

Graph database is becoming popular. However, managing history is not yet possible in an easy way while being critical in many applications. Tracking changes is indeed one of the main functionalities in databases. To deal with this problem, We are developing a plug-in for graph database system, especially for Neo4j. We name it Mnemosyne. To make it with high feasibility in used, it has to meet some requirements. One of them is the performance. How does Mnemosyne affect the time and performance of Neo4j database. How does Mnemosyne work will not be presented in here. In this paper, I express our methods and the results of the test. Finally, we analyze the result and evaluate the Mnemosyne plug-in. This work of my internship is a practical way beside the theory, so i can get familiar with NoSQL graph database.

2.2 Data Summaries

2.2.1 Linguistic Summaries

Data Summarization (also known as linguistic summarization) can be seen as a knowledge discovery task in Databases, which aims to extract knowledge, represented in textual form, from input data (numerical data or time series data). The objective is to automatically generate the same kind of descriptions that a human can provide about the data. The importance of this task because the increasing amount of available data makes it impossible to have human experts doing the job of summary the data.

For instance, classical areas are the generation of weather and pollution reports, reports of the monitoring of people in hospitals and for elder care, energy consumption, traffic, etc, where the data is very huge and people can not sum up the information from the data as human language. For example: Weather system (SumTime project:[2]) has a big amount of data , and it can generate summary of data similar to human language like: S 8-13 increasing 18-23 by morning, then easing 8-13 by midnight. A linguistic summary is expressed in the form:

Q y 's are P (2.1)

or in extended form:

Q R y's are P (2.2)

Where Q is a fuzzy quantifier (for instance *Most*), y represents objects that are summarized in a set of objects Y=$y_1, y_2, ..., y_n$, R is a qualifier, and P is summarizer (e.g., *Low* for attribute Salary).

[2]http://inf.abdn.ac.uk/research/sumtime/

For example: Most of the employees earn low salary

For every summary, a truth value is computed, which is the basic criterion to evaluate the quality of the linguistic summaries. The truth (validity) T of the summary is a number ranging in the interval $[0, 1]$ assessing the truth (validity) of the summary (e.g., 0.7). Therefore, a linguistic summary may be evaluate as:

$$T(most\ of\ employees\ earn\ low\ salary) = 0.7$$

The truth validity is presented in the part of *Quality Measures* in this report.

2.2.2 Fuzzy Set Theory

Linguistic summaries often use fuzzy descriptions, such as fuzzy quantifiers (e.g., most, few) and linguistic terms (e.g., low, high).

Definitions of such concepts are thus provided below.

A fuzzy set A in a universe of discourse $X = x$, denoted by $AinX$, is defined as a set of pairs $(x, \mu_A(x))$[Zim92].

$$A = (\mu_A(x), x) \tag{2.3}$$

where $\mu_A : X \rightarrow [0, 1]$ is the membership function of A and μ_A(x) $\in$ [0, 1] is the grade of membership (or a membership grade) of an element $x \in X$ in a fuzzy set A A fuzzy set A is said to be empty, denoted by $A = \emptyset$, if and only if

$$\mu_A(x) = 0, \forall x \in X \tag{2.4}$$

Two fuzzy sets A and B defined in the same universe of discourse X are said to be equal, denoted by $A = B$, if and only if

$$\mu_A(x) = \mu_B(x), \forall x \in X \tag{2.5}$$

2.2.3 Fuzzy Membership Function

Membership functions (MF) quantify the grade of membership of the element x to the fuzzy set A. MF for fuzzy sets can be defined by many ways. The most common types of membership functions are shown in figure below:

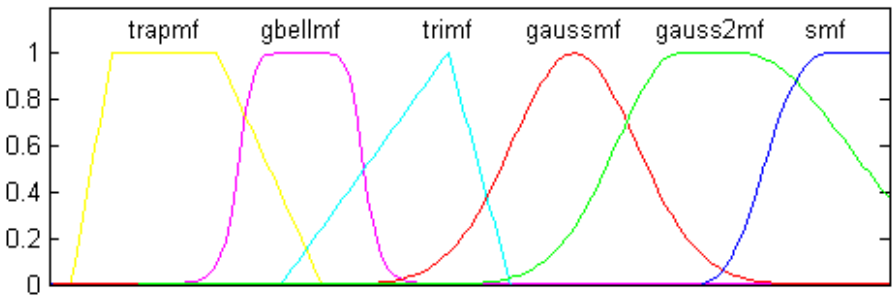

Figure 2.4: Most common membership functions

We consider trapezoidal membership function. A trapezoidal MF is specified by four parameters a, b, c, d as follows:

$$\text{trapezoid}(x; a, b, c, d) = \begin{cases} 0, & x \leq a. \\ \frac{x-a}{b-a}, & a \leq x \leq b. \\ 1, & b \leq x \leq c. \\ \frac{d-x}{d-c}, & c \leq x \leq d. \\ 0, & d \leq x. \end{cases}$$

An alternative concise expression using min and max is:

$$\text{trapezoid}(x; a, b, c, d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right).$$

Data summaries have been proven to provide users with relevant and usefull information from their databases. However, very few works have addressed graph databases summarization, especially in the case of NoSQL databases. In our work, we have thus proposed some approaches to summarize such NoSQL graph databases. The next chapters present these propositions and the experimental work we have done for assessing them.

Chapter 3

Proposition

Our research team has developed three types of graph summaries in the context of NoSQL graph databases:

- Type 1, also called "Structure-Summaries" exhibit the schema of the graph
- Type 2, also called "DataStructure-Summaries" exhibit the differences of the graph schema depending on some data values. They extend the first type of summaries.
- Types 3, also called "Temporal DataStructure-Summaries" allow to exhibit Type 2 summaries with temporal information.

3.1 Defining Summaries

3.1.1 Structure-Summaries

Structure summaries are meant to retrieve the structure of the graph, which could somehow be associated with the schema in relational databases. Such summaries could thus be linked with schema mining from the literature (see Section 2.2).

Def 4 (Structure Summary) *Let $G = (V, E, \alpha, \beta, L)$ be a graph. A Structure summary is defined as a subgraph $a - [r] - > b, Q$ with $a, b, r \in L$ and Q a fuzzy quantifier. where V stands for a set of vertices and E stands for a set of edges with $E \subseteq \{V \times V\}$, α stands for the set of attributes defined over the nodes, and β the set of attributes defined over the relations.*

Example 1 *In the toy example, $Student - [rent] - > Appartment, Most$ is a structure summary.*

3.1.2 DataStructure-Summaries

Data structure summaries are meant to refine structure summaries. They allow to distinguish some cases for the schema which depend on the value of some properties. For instance, it may be the case that employees rent rather appartments or houses depending on their salary.

Def 5 (Data Structure Summary) *Let $G = (V, E, \alpha, \beta, L)$ be a graph. A Data Structure Summary is defined as a subgraph $a - [r] - > b, Q$ with $a, b \in L \bigcup \alpha$, $r \in L \bigcup \beta$ and Q a fuzzy quantifier.*

Example 2 *In the toy example, $Student(young) - [rent] - > Appartment, Most$ is a structure summary.*

3.1.3 Temporal DataStructure-Summaries

Temporal data structure summaries allow to integrate temporal information in the summaries.

Def 6 (Temporal Data Structure Summary) *Let $G = (V, E, \alpha, \beta, L)$ be a graph. A Temporal Data Structure Summary is defined as a subgraph $a - [r] - > b, Q, TD$ with $a, b \in L \bigcup \alpha$, $r \in L \bigcup \beta$, Q a fuzzy quantifier and TD a temporal description.*

Example 3 *In the toy example, "$Student - [rent] - > Appartment, Most, First\ year$" is a temporal data structure summary.*

3.2 Quality Measures

Quality measures are crucial when working on summaries and patterns. They indeed provide information to the end-users about the veracity and quality of the proposed information. Many works have dealt with this topic, for instance in the field of association rule mining [TKS02].

Regarding linguistic summaries, the most used measure is the degree of truth.The degree of truth, validity from [0,1] is the basic measure. It determines the degree to which a linguistically quantified proposition equated with a linguistic summary is true. Truth values for simple protoforms above are defined by:

$$\text{T (Q y's are P)} = \mu_Q \left(\frac{1}{n} \sum_{i=1}^{n} \mu_P(y_i) \right) \quad (3.1)$$

and for extended protoforms, including a qualifier, as:

$$\text{T (Q R y's are P)} = \mu_Q \left(\frac{\sum_{i=1}^{n} \mu_P(y_i) \wedge \mu_R(y_i)}{\sum_{i=1}^{n} \mu_R(y_i)} \right) \quad (3.2)$$

Where n is the number of objects (y_i) that are summarized, and μ_P , μ_R , and μ_Q are the membership functions of the summarizer, qualifier, and quantifier, respectively. Where $\wedge$ is the minimum operation.

In our framework, we have proposed several quality measures:

- Representativity
- Degree of truth
- Diversity

3.2.1 Representativity

Representativity aims at informing the user about the presence of the pattern appearing in the summary in the databases. It could indeed be the case that most of the students rent an appartment but that students and appartments appear only once in a huge database.

For this purpose, we propose to display the ratio of occurrences of the relation being displayed over the number of relations in the graph.

Def 7 (Representativity) *Given a graph database $G = (V, E, \alpha, \beta)$ and a summary S, the representativity of S in G is defined as*

$$Representativity(S) = \frac{count(DistinctS)}{|E|}$$

Roughly speaking, representativity can be compared to the *support* in association rule mining.

3.2.2 Degree of Truth

The degree of truth determines the extent to which the relation appearing in the summary is veridic regarding the fuzzy quantifier. For instance, if the summary mentions that *most of the students rent an appartment,* then the degree of truth describes to which extend a high proportion of students rent an appartement.

Def 8 (Degree of Truth) *Given a graph database $G = (V, E, \alpha, \beta)$ and a summary $S = a - [r] - > b, Q$ or $S = a - [r] - > b, Q, TD$ including the fuzzy quantifier, the degree of Truth of S in G is defined as*

$$Truth(S) = \mu_Q \left(\frac{count(distinct(S)}{count(distinct(a))} \right)$$

Roughly speaking, the ratio appearing the definition of the degree of truth can be compared to the $confidence$ in association rule mining.

3.2.3 Diversity

The **representativity** is a good quality indicator but the summary can be improved with the use of a complementarity measure : the diversity.

The diversity provides information about type 2 or 3 summaries. It is composed of two criterias which are merged.

Def 9 (Diversity) *Given a graph database $G = (V, E, \alpha, \beta)$ and a summary of type 2 $S = a - [r] - > b, Q$ or of type 3 $S = a - [r] - > b, Q, TD$ including the fuzzy quantifier, the degree of Truth of S in G is defined as*

$$Diversity(S) = (coef1 * D1 + coef2 * D2)$$

where $coef1 + coef2 = 1$

D1 metric : diversity of the relation

For a pattern $a-[r]->b$, the diversity of the relation for a couple of node (a,b) is defined as :

$$D1 = \frac{|a-[r]->b|}{|a-[?]->b|}$$

If D1 is small for $a-[r]->b$ it means that on the point of view of the diversity, the weight of r for (a,b) is low.

For example, for a pattern $(Person)-[RENT]->(Place)$ we can have several subpatterns :

- $(Person:Student)-[RENT]->(Place:Appartement)$
- $(Person:Student)-[RENT]->(Place:Home)$
- $(Person:CEO)-[RENT]->(Place:Home)$

If for $D1((Person:Student)-[RENT]->(Place:Home)) < 1\%$ this relation may be declared as insignificant for the summary.

D2 metric : diversity of the target

For a pattern $a-[r]->b$, the diversity of the target b for a couple (a,r) is defined as :

$$D2 = \frac{|a-[r]->b|}{|a-[r]->(?)|}$$

If D2 is high, for a target node, it means that on the point of view of the diversity, the weight of b for (a,r) is high.

On the same example as in the previous section, it looks like the target $Place:Home$ is used 100% of the time for the couple $(Person:CEO, RENT)$ in the actual graph database.

Diversity metric

For the previous example pattern $(Person)-[RENT]->(Place)$ with this subpatterns :

- $(Person:Student)-[RENT]->(Place:Appartement)$
- $(Person:Student)-[RENT]->(Place:Home)$
- $(Person:CEO)-[RENT]->(Place:Home)$

The Diversity metric for $(Person:CEO)-[RENT]->(Place:Home)$ is calculated as this :

$$Diversity(S) = coef1 * \frac{|Person:CEO-[RENT]->Place:Home|}{|Person:CEO-[?]->Place:Home|} + coef2 * \frac{|Person:CEO-[RENT]->Place:HOME|}{|Person:CEO-[RENT]->(?)|} \quad (3.3)$$

If Diversity was only composed of D1 criteria, then this information would probably be declared as irrelevant. The D2 criteria is balancing D1.

The coefficients $coef1$ and $coef2$ should be choose carefully about what is the most efficient with respect to the context.

These are three types of Quality Measures we have proposed. In next part presents the way to extract such summaries.

3.3 Extracting the summaries

Extracting summaries is the task to obtain knowledge and export as linguistic sum-up information of data. This section presents the ways to extract the summaries defined previously in this report.

I focus on the basic way to extract type 1, structure summary. The type summary of type 2 and 3 are not enough time during my internship. Thus, Type 1 will be presented in this report.

Considering the following steps:

- Step 1: Query the patterns (relationship), and number of each pattern. The result can be transformed into table. The goal of this phase is to prepossess the data from graph database and represent it in suitable format.
- Step 2: Generate all possible summary, a summary with a pattern and fuzzy quantifier
- Step 3: Evaluate each summary generated

Example: Let say we have the toy database in as in Fig. 1.1.
Step 1: Query the patterns (relationships), and number of each pattern
Query to count the distinct relationships N1←:
MATCH (a)-[r]→(m) RETURN DISTINCT labels(a), type(r), labels(m), count(r)

labels(a)	type(r)	labels(m)	count(r)
Student	RENT	Apartment	4
Student	Friend	Student	2
Student	OWN	House	1

Figure 3.1: Patterns and Count

Query to count all relationships N2←: MATCH (a)-[r]→(m) RETURN count(r)
Query to count all a nodes N3←: MATCH (n:Student) return distinct count(n)

Now, we have a table like this:

Relation	Count(N1)	(N1/N2)	MF_{Most}(N1/N3)	MF_{Few}(N1/N3)	$MF_{VeryFew}$(N1/N3)
Student-Rent-Apartment	4	4/7	$MF_{Most}(4/5)$	$MF_{Few}(4/5)$	$MF_{VeryFew}(4/5)$
Student-Friend-Student	2	2/7	$MF_{Most}(2/5)$	$MF_{Few}(2/5)$	$MF_{VeryFew}(2/5)$
Student-Own-House	1	1/7	$MF_{Most}(1/5)$	$MF_{Few}(1/5)$	$MF_{VeryFew}(1/5)$

Where : N1/N2 is Representativity , MF_Q(N1/N3) is Degree of truth
The membership function MF is calculated follow by trapezoidal MF formula.
Step 2: All possible summaries are:

- Student-Rent-Apartment, Most
- Student-Rent-Apartment, Few
- Student-Rent-Apartment, VeryFew
- Student-Friend-Student, Most
- Student-Friend-Student, Few
- Student-Friend-Student, VeryFew
- Student-Own-House, Most
- Student-Own-House, Few
- Student-Own-House, VeryFew

Step 3: Evaluate each summary, compute the evaluation measure selected .For example, we evaluate by degree of truth of T(Student-Rent-Apartment,Most)=$\mu_{Most}\left(\frac{4}{5}\right)$

Let say, we design the membership function as: μ_{Most}(x,0.6,0.75,0.8,1) we will have:

T(Student-Rent-Apartment,Most)=$\mu_{Most}\,(4/5, 0.6, 0.75, 0.8, 1)$= 1

The universe of definition of a membership function must fit the needs. In this case the needs is [0,1] because the fuzzy set represents a quantifier.

Chapter 4

Experiments and Results

Our work has been implemented and tested on synthetic and real databases. As a first part of this work on the implementation, we have worked on the software environment and on the architectures.

I have been asked to work with Java as a programming language and Neo4j for the NoSQL graph database engine. These choices have been done by my advisors because this is the environement currently used in the research team. Eclipse was chosen as an IDE.

This section is divided into three parts:

1. The first presents the protocol and the general design of experiments;
2. The second deals with experiments on Mnemosyne (an another subject treated by the research team I'm working with) that I have done with this protocol under the supervision of my advisors;
3. The third is about experiments on summaries of graphs that I conducted without being supervised.

4.1 Experiments design

This section presents the test environment and the methodology used.

4.1.1 Test Environment

Setup

The setup used for running the test is :

- Operating System : OS X 10.8.5 64 bits;
- Processor : Intel Core i5 2.4 Ghz;
- Java 1.7.0_45-b18 Java HotSpot(TM) Server VM (build 24.45-b08, mixed mode);
- JVM configuration : Heap is set at 1024m, Perm at 256m

- Database Engine : Neo4J 2.0.1

It should be noted that the version of Java used is Oracle JVM for OpenJDK, not a follow the directions for using Neo4j. Similarly, at the time of this report is written, Neo4j is not certified for Java 8.

DataSet

There is two type of dataset in the experiments of this section :

- dataset 1 is based on the MusicBrainz project. It is an open dataset about music. The database contains about 800 000 artists, 75 000 labels, 1.2 millions releases and more than 12 Millions songs. Datasize is 22.54 Go;
- dataset 2 is a generated dataset to adapt database size to the needs.

Tools

There are severals tools used in the experiments, the more importants are :

- **Eclipse**, an Integrated Development Environment (IDE) that contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications. Expetiments' Java code is written using Eclipse IDE.
- **Apache Maven** is a build automation tool used primarily for Java projects. Maven uses Project Object Model (POM) to define a process for building project. All elements are depended on modules. It predefines targets for initiation tasks, complier, deployment and the order of process to make everything run as good as possible [Mav11]. We use maven to configure our test process, run the test automatically.
- **Gnuplot** is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. It can generate two and three-dimensional plots of functions or data [WK+10]. We use this tool to visualize and from that analyze the output data.

4.1.2 Methodology

This section describes the methodology used to perform the various tests found in this section.

Test Plan

The test plan focuses on the recovery of two metrics: execution time and memory consumption.

Improving the quality of results

To improve results quality, i used the protocol below :

- Each test was run a series of 10 times to limit the side effects of the environment (I/O interruption, CPU usage by another process, etc.);

- deletion of outline datas;
- cleaning environment (generated files, etc.) before each test;
- fork a new JVM foreach test to avoid side effect (like cache systems) and avoid garbage collection run by the JVM.

Some other mesures are specific to each kind of test and will be explain in the related section.

4.2 Mnemosyne experiment

Mnemosyne is a historization (versioning) system for graph databases. The goal is to store historization (the successive state of data and data's action) of the graph in a subgraphe called VersionGraph that has his own kind of modelization (optimized for storing version of nodes, relations and graph versions). Mnemosyne is deployed as a plugin into graph databases engine.

To show that i have well understand the standard protocol, my advisors ask me to do some experiments on Mnemosyne. To do so, I had to refine the methodology.

In this section, I explain in more detail the test plan and specific protocols for this experiments. After that, I show and discuss the results.

Benchmarking

To determine the impact on performance of Mnemosyne on Neo4j, it is necessary to have an experimental calibration. The goal is to have standard metrics of time execution and memory consumption Neo4j in order to compare them with those obtained when Mnemosyne is activated.

Test Plan

To test the performance of Mnemosyne, we write java code that do some modification actions (create, update, delete) on a graph database.

The execution time is tracked when tests are run with and without using Mnemosyne. The results are exported and ploted.

There number of elements to test are: 10, 100, 1000, 10000, 100000, 200000, 400000 and 600000 nodes or relationships, hardware: 2.4Ghz Intel Core i5.

For each test action, we run ten times. We need to run the test several times to avoid some special output because of unusual effect from environment. the final result will be computed as average number. The environment for each test is should be the same. We need to configure the Maven to make environment clean first by using command: **mvn clean -f**

Unit Test

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level,

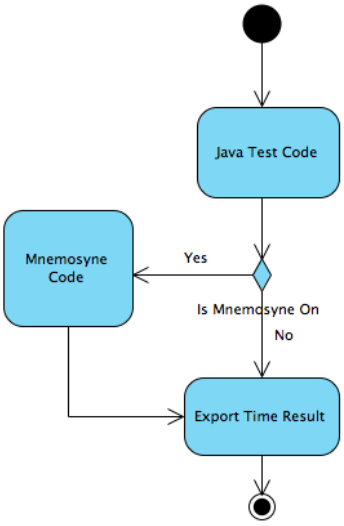

Figure 4.1: Test Branch

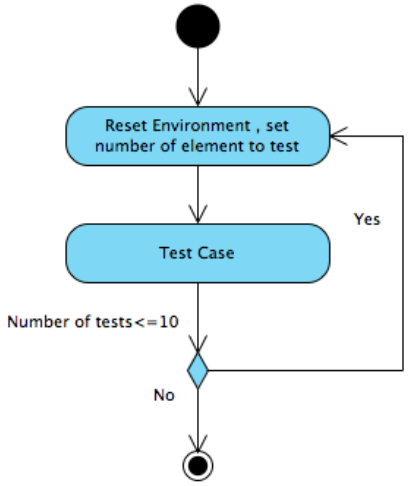

Figure 4.2: Test Plan

and the minimal unit tests include the constructors and destructors [Bin00]. In the unit test level, We use JUnit to test each functions. JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development (TDD) which is a software development process that relies on the repetition of a very short development cycle [BG02].

Test Cases

First of all, we need to validate the java code if it works correctly or not. If it is validated, we can use this for the next phase which is to test Mnemosyne performance. In this section, we present some test cases which are used to test all the java code functions.

Test Case 1

1. Title: Test Create Node

2. Description: This to test if the java code to create node correctly or not

Input	Expected output
Input the value: 10	a string contains "10,time in millisecond", Assert if the output is wrong

Table 4.1: Test case Create Node

Test Case 2

1. Title: Test Update Node

2. Description: This to test if the java code to update node correctly or not

Input	Expected output
Input the value: 10	a string contains "10,time in millisecond",Assert if the output is wrong

Table 4.2: Test case Update Node

Test Case 3

1. Title: Test Delete Node

2. Description: This to test if the java code to update node correctly or not

Input	Expected output
Input the value: 10	a string contains "10,time in millisecond", Assert if the output is wrong

Table 4.3: Test case Delete Node

Test Case 4

1. Title: Test Create Relationship
2. Description: This to test if the java code to create relationship correctly or not

Input	Expected output
Input the value: 10 relationships	a string contains "10,time in millisecond", Assert if the output is wrong

Table 4.4: Test case Create Relationship

Test Case 5

1. Title: Test Update Relationship
2. Description: This to test if the java code to update relationship correctly or not

Input	Expected output
Input the value: 10 relationships	a string contains "10,time in millisecond", Assert if the output is wrong

Table 4.5: Test case Update Relationship

Test Case 6

1. Title: Test Delete Relationship
2. Description: This to test if the java code to delete relationship correctly or not

Input	Expected output
Input the value: 10 relationships	a string contains "10,time in millisecond", Assert if the output is wrong

Table 4.6: Test case Delete Relationship

Algorithms

In this section, We present the algorithms implemented in java code to make the tests.

Algorithm 1: Create Node

Data: Number of node to create $numNode$, transaction size $tranSize$

Result: A string contains number of node created and the time

begin

- $numTrans \longleftarrow$compute number of transaction
- $nodeCreated \longleftarrow 0$
- time start
- **for** $step = 0$ **to** $numTrans$ **do**
  - begin transaction
  - $remainingNode \longleftarrow$numNode−nodeUpdated>=tranSize? $tranSize$: $numNode$ - $nodeUpdated$
  - **for** $step = 0$ **to** $remainingNode$ **do**
    - create a node
    - nodeCreated++
  - end transaction
- time stop
- return nodeCreated and time as string

Algorithm 2: Update Node

Data: Number of node to update $numNode$, transaction size $tranSize$

Result: A string contains number of node updated and the time

begin

- $numTrans \longleftarrow$compute number of transaction
- $nodeUpdated \longleftarrow 0$
- $allNodes \longleftarrow$get all nodes the graph database
- time start
- **for** $step = 0$ **to** $numTrans$ **do**
  - begin transaction
  - $remainingNode \longleftarrow$numNode−nodeUpdated>=tranSize? $tranSize$: $numNode$ - $nodeUpdated$
  - **for** $index = 0$ **to** $remainingNode$ **do**
    - update node at allNodes[index]
    - nodeUpdated++
  - end transaction
- time stop
- return nodeUpdated and time as string

Algorithm 3: Delete Node

Data: Number of node to delete $numNode$, transaction size $tranSize$

Result: A string contains number of node deleted and the time

begin

- $numTrans \longleftarrow$compute number of transaction
- $deletetedNode \longleftarrow 0$
- $allNodes \longleftarrow$get all nodes the graph database
- time start
- **for** $step = 0$ **to** $numTrans$ **do**
  - begin transaction
  - $remainingNode \longleftarrow$numNode−deletetedNode>=tranSize? $tranSize$: $numNode$ - $deletetedNode$
  - **for** $node = 0$ **to** $remainingNode$ **do**
    - delete node at allNodes[index]
    - deletetedNode++
  - end transaction
- time stop
- return deletetedNode and time as string

Algorithm 4: Create Relationship

Data: Number of relationship to create *numRel*, transaction size *tranSize*

Result: A string contains number of relationship created and the time

begin

```
numTrans ⟵compute number of transaction
relationshipCreated ⟵0
allNodes ⟵get all nodes the graph database
time start
for tx = 0 to numTrans do
    begin transaction
    remainingRelationship ⟵numRel−relationshipUpdated>=tranSize? tranSize : numRel -
    relationshipCreated
    for index = 0 to remainingRelationship do
        thisNode ⟵allNodes[index]
        forwardNode ⟵allNodes[tx * transactionSize + index]
        if thisNode is not related to forwardNode then /*                    */
            thisNode creates relationship with forwardNode
            relationshipCreated++
    end transaction
time stop
return relationshipCreated and time as string
```

Algorithm 5: Update Relationship

Data: Number of relationship to update *numRel*, transaction size *tranSize*

Result: A string contains number of relationship updated and the time

begin

- *numTrans* ⟵compute number of transaction
- *relationshipUpdated* ⟵0
- *allRelationships* ⟵get all nodes the graph database
- time start
- **for** *step* = 0 **to** *numTrans* **do**
  - begin transaction
  - *remainingRelationship* ⟵numRel−relationshipUpdated>=tranSize? *tranSize* : *numRel* - *relationshipUpdated*
  - **for** *index* = 0 **to** *remainingRelationship* **do**
    - **if** *allRelationships[index] not null* **then** /* */
      - update relationship at allRelationships[index]
      - relationshipUpdated++
  - end transaction
- time stop
- return nodeUpdated and time as string

Algorithm 6: Delete Relationship

Data: Number of relationships to delete *numRel*, transaction size *tranSize*

Result: A string contains number of relationships deleted and the time

begin

numTrans ⟵compute number of transaction

deletetedRelationship ⟵0

allRelationships ⟵get all relationships in the graph database

time start

for *step* = 0 **to** *numTrans* **do**

begin transaction

remainingRelationships ⟵numRel−deletetedRelationship>=tranSize?*tranSize*: *numRel* - *nodeCreated*

for *node* = 0 **to** *remainingRelationships* **do**

if *allRelationships[index] not null* **then** /* */

delete relationship at allRelationships[index]

deletetedRelationship++

end transaction

time stop

return deletetedRelationship and time as string

Query Test

This section present the method to test how does Mnemosyne effect the performance in using cypher query. Cypher is an expressive (yet compact) graph database query language [RWE13b]. In this paper, we only test reading queries.

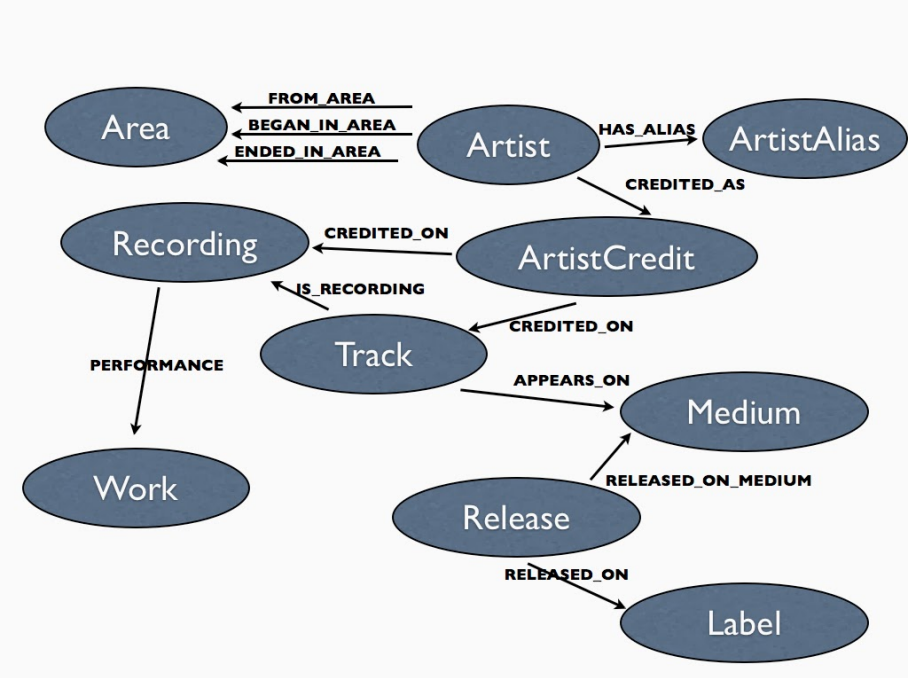

Figure 4.3: MusicBrainz Database Schema

MusicBrainz Database MusicBrainz is a project that aims to create an open content music database. MusicBrainz captures information about artists, their recorded works, and the relationships between them [Swa02]. MusicBrainz currently has around 1000 active users, nearly 800,000 artists, 75,000 record labels, around 1,200,000 releases, more than 12,000,000 tracks, and short under 2,000,000 URLs for these entities. The database size is 22.54 GB. These are four queries be tested. the most basic query test which matches pattern at one level.

Query 1 Alias of Taylor Swift singer

```
START taylor=node:mb_fulltext(name="Taylor Swift")
MATCH (taylor:Artist),(ali:ArtistAlias),
taylor-[:HAS_ALIAS]-ali
RETURN taylor,ali limit 1000
```

START specifies one or more starting points, nodes or relationships in the graph. These starting points are obtained via index lookups or, more rarely, accessed directly based on node or relationship IDs. In

the example query, we are looking up a start node. We ask to find a node with a name property whose value is $TaylorSwift$. The return value from this lookup is bound to an identifier, which is called *taylor* here. This identifier allows us to refer to this starting node throughout the rest of the query.
MATCH at the simple pattern $taylor-[:HAS\_ALIAS]-ali$. This pattern describes a path comprising two nodes, one of which we've bound to the identifier *taylor*, the others to *ali*. These nodes are connected by way of several HAS_ALIAS relationships.
RETURN clause specifies which nodes, relationships, and properties in the matched data should be returned to the client. In our example query, we are interested in returning the nodes bound to the *taylor* and *ali* identifiers. We limit 1000 results from the dataset.
Now, there are three another queries. Which has higher level of pattern in MATCH clause.

Query 2 Alias of all USA artists

```
START vn=node:mb_fulltext(name="United States")
MATCH (vn:Country),(ali:ArtistAlias),
(a:Artist)-[:FROM_AREA]-(vn),
(a:Artist)-[:HAS_ALIAS]-ali
RETURN a.name,ali limit 1000
```

Query 3 All recordings of Westlife band

```
START wf=node:mb_fulltext(name="Westlife")
MATCH (wf:Artist),(ac:ArtistCredit),
wf-[:CREDITED_AS]-ac-[CREDITED_ON]-(rc:Recording)
RETURN wf,ac,rc limit 1000
```

Query 4 Listing American artists signed on British record labels

```
START usa=node:mb_fulltext(name="United States"),
gb=node:mb_fulltext(name="United Kingdom")
MATCH (usa:Country), (gb:Country),
(a:Artist)-[:FROM_AREA]-(usa),
(a:Artist)-[:RECORDING_CONTRACT]-(l:Label),
(l)-[:FROM_AREA]-(gb)
RETURN a,l,usa,gb limit 1000
```

Output Validation

Data Validation is a important step in our test. After running all the java code tests, using and without using Mnemosyne, we have a set of results. We need to validate this data, check if there is any wrong data in our data set. This step, we focus on how to detect outliers in output data. For example, we have the output data set S={1,2,3,2,20}, If we still compute average result as (1+2+3+2+20)/5 = 5.6. This can be wrong

data in our case, when the time is not high like that. It maybe causes from some things in environment that make the test to high from the medium . The number 20 can be consider as an outlier in the data set.
To compute the right average value, the first step is to remove outliers. Second step is to compute average value of original set minus outliers set. In the example set above, the outlier is 20. The right average value has to be (1+2+3+2)/4 = 2.
We use Interquartile Range technique to detect and remove outliers in our data set.

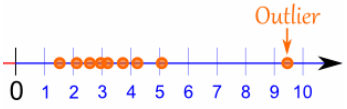

Figure 4.4: Outlier Example

Algorithm 7: Outliers Detection

Data: a set of numbers S
Result: a set of outliers O
begin
  $medianValue$ ⟵compute median of set S
  $quartileQ3$ ⟵compute median of all elements >= $medianValue$
  $quartileQ1$ ⟵compute median of all elements < $medianValue$
  IQR ⟵quartileQ3-quartileQ1
  $highestBorder$ ⟵(quartileQ3 + (1.5*IQR))
  $lowestBorder$ ⟵(quartileQ1 - (1.5*IQR))
  for $element \in S$ **do**
    if $element < lowestBorder$ || $element > highestBorder$ **then** /* */
      O adds $element$
  return O

Memory Consumption

Memory consumption is also important criteria to test. We need to know how many memory mnemosyne need to finish the task. We use Java Management Extensions (JMX) to monitor the resource used by Mnemosyne. JMX is a Java technology that supplies tools for managing and monitoring applications [Per02]. In our test, we track the Heap Memory Usage when enabling and disabling Mnemosyne plug-in, compare the data between two of them and see what is the difference. We track the memory every one second (1000 millisecond) for every task.

4.2.1 Result

Test Cases

Checked List

1. Description: This checked list shows the result of tested cases

Test Case	Actual Result	Remarks
Test Create Node	Meet expected result	Pass
Test Update Node	Meet expected result	Pass
Test Delete Node	Meet expected result	Pass
Test Create Relationship	Meet expected result	Pass
Test Update Relationship	Meet expected result	Pass
Test Delete Relationship	Meet expected result	Pass

Table 4.7: Checked list for all test cases

Overview

Here is the overview of the comparison with all tests: create, update, delete nodes and relationships. We use the same color but different line style for the same action. Scatter lines means using Mnemosyne and continuos lines mean without using Mnemosyne.

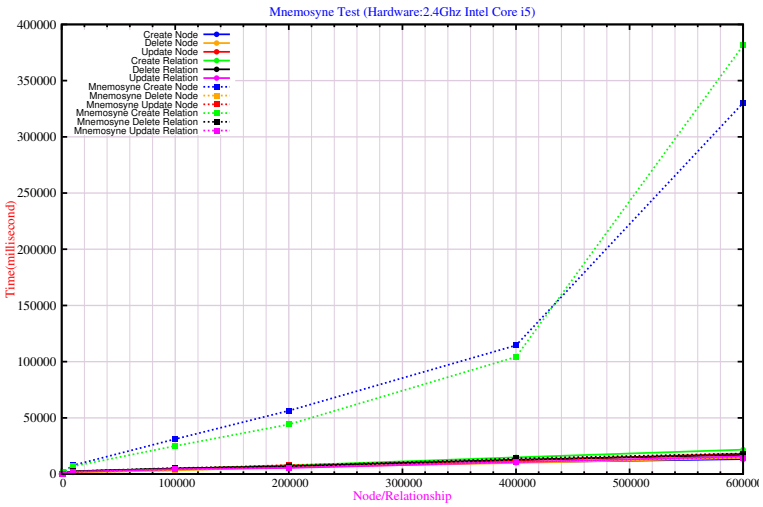

Figure 4.5: Overview of all CRUD tests

From overview, we can see that Mnemosyne plug-in mostly effects in creating node and creating relationships. With Mnemosyne, the time consumption to create nodes (blue scatter line) and create relationships (green scatter line) has very high time, over 300 seconds while the time is just under 30 seconds to do so without using Mnemosyne.
However, with the other tests such as: delete , update nodes and relationships, there is no big difference. it seems to be very similar. We see the graph below to explore in detail.

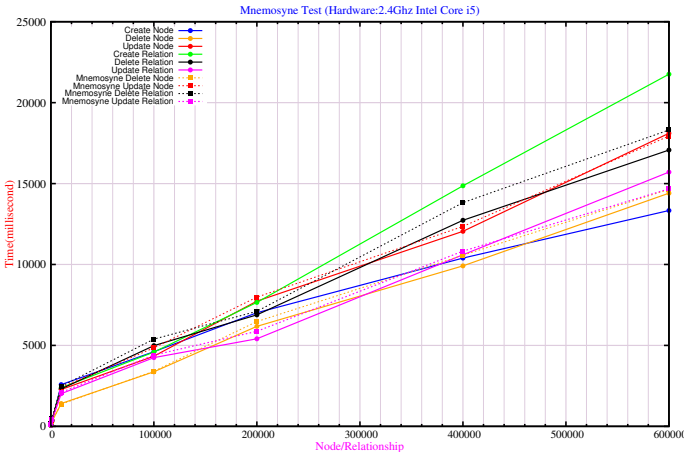

Figure 4.6: Overview without Creating node and relationship test using Mnemosyne

The figure above has removed the lines for creating node and relationship using Mnemosyne where they are too different. Other actions, the time consumption trend is going to be slowly separated for the same test action along the time . Which means that, the gap between two lines of an action is higher and higher when increasing the number of elements.
We see the trend for each test action in more detail.

Creating Node Comparison

The result shows that, the gap between two lines is increasing quite fast, which means that mnemosyne effects a lot the performance. The time comsumption in this case without using mnemosyne grows slowly. We can see at the milestone of 600000 elements, it is just over 10 seconds while mnemosyne line is 330 seconds.

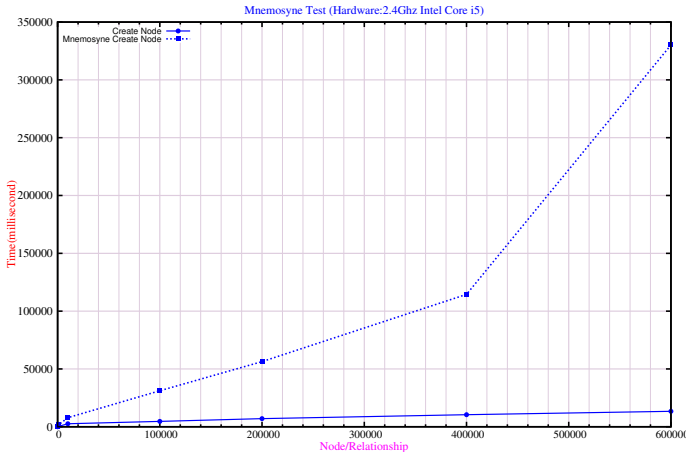

Figure 4.7: Creating Node Comparison

Updating Node Comparison

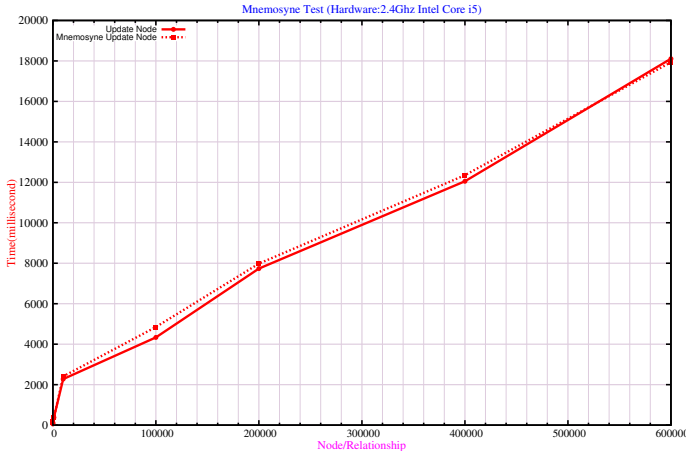

Figure 4.8: Updating Node Comparison

The gap between two lines is not increasing. Sometime mnemosyne line is higher and sometime it is lower. In this test, Mnemosyne effected a bit the performance if we don't want to say that it is not effected.

Deleting Node Comparison

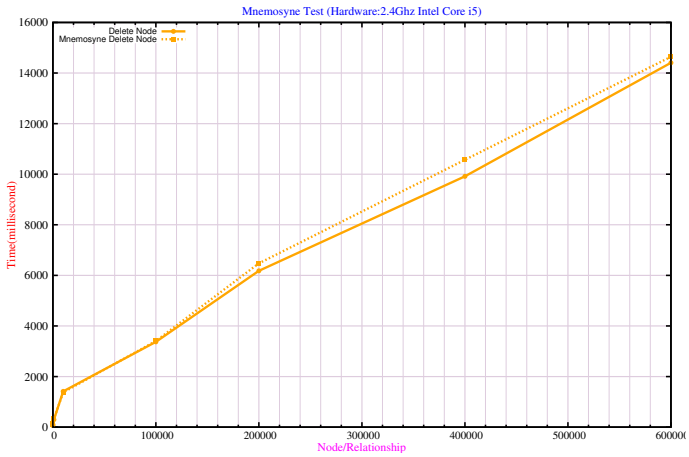

Figure 4.9: Deleting Node Comparison

In this test, we can see that the performance using Mnemosyne is lower a bit, or the time consumption is higher. The scatter lines is alway higher, which mean that the gap is not going to increase. It keeps comparative similar when the number of element is increasing. The gap is too small from 10 to 100000 nodes that why two lines look like one.

Creating Relationship Comparison

We see the figure, there is big gap between two lines. This is the same trend with Creating Node where the gap grows fast for each milestone of elements. The performance after plug Mnemosyne is much more slower.

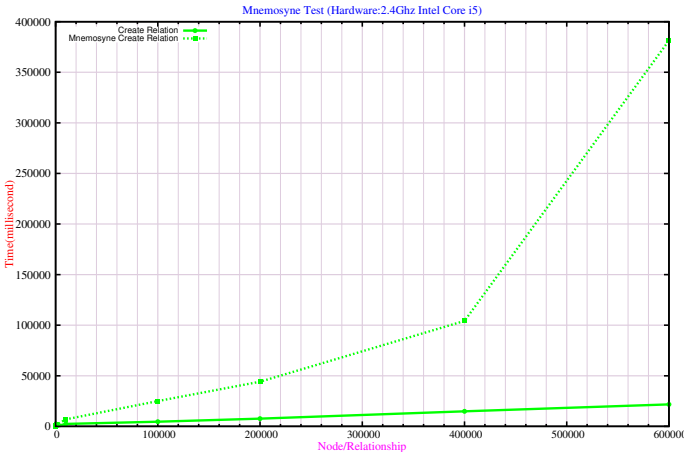

Figure 4.10: Creating Relationship Comparison

Update Relationship Comparison

Updating relationship has the same trend with updating node. Sometime, mnemosyne line go up and sometime it goes down lower than other line. This means that, If we continue going, two lines are still very closed together. On the other hand, we can say that mnemosyne does not effect to the performance in this test.

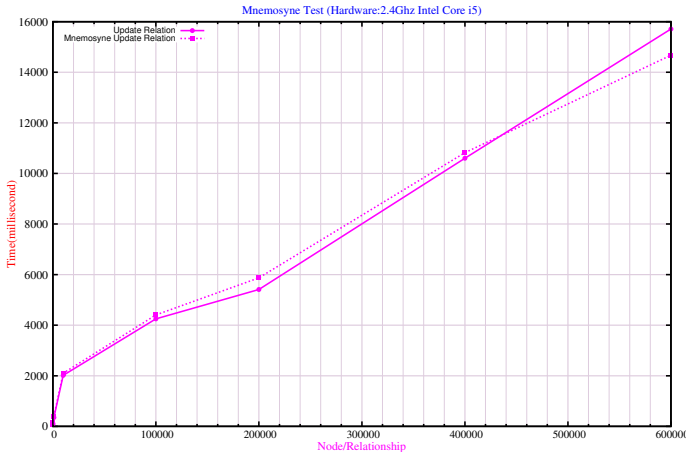

Figure 4.11: Update Relationship Comparison

Deleting Relationship Comparison

For deleting relationship comparison, we the the mnemosyne line is alway higher than other line. The gap between two lines is also increasing during the time. This means, mnemosyne has lower performance and will be increased corresponding with increasing the number of elements. However, The gap does not grow too fast. For example, at milestone of 400000 elements, the gap is 1 second, and with 600000 elements, it is just 1.2 second.

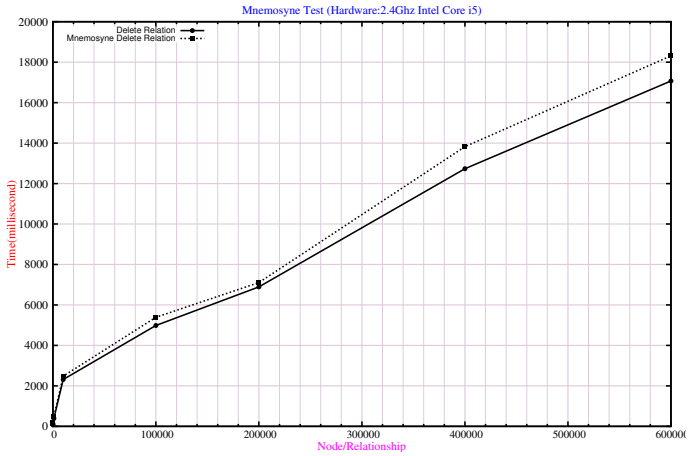

Figure 4.12: Delete Relationship Comparison

Query Result

The time of queries in the graph for using Mnemosyne and without using Mnemosyne are not really different. This mean that Mnemosyne does not effected in queries for reading. From the graph, we also see that, querying data in cypher with many short patterns is still lower than one long pattern . This because in graph database, every node has references to other nodes, it get do some queries like "JOIN" in RDBMS very fast. Actually , NoSQL does have JOIN operation.

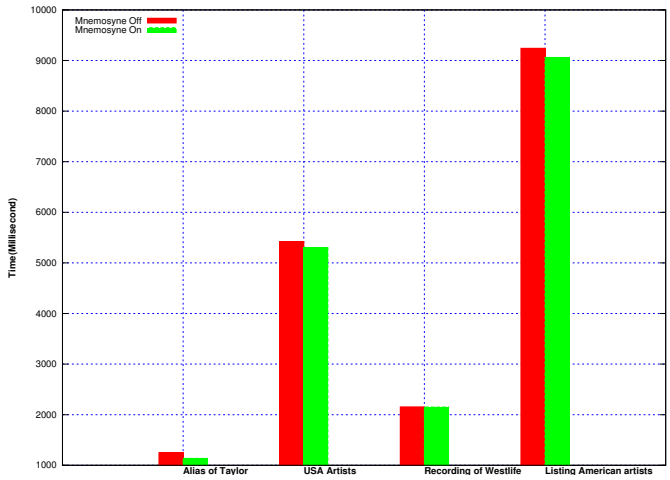

Figure 4.13: Query Comparison

Memory Consumption

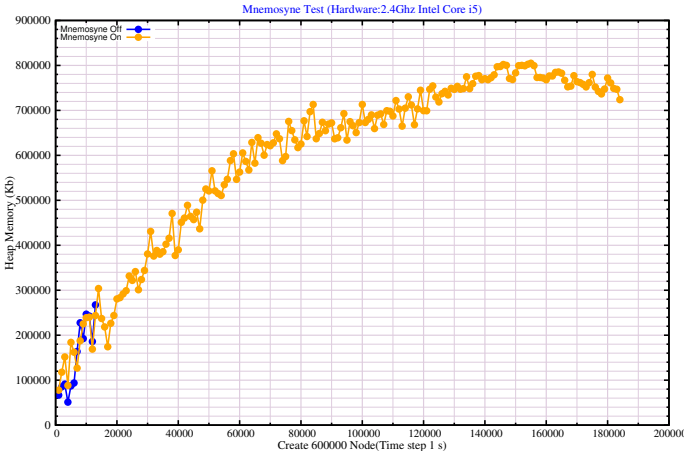

Figure 4.14: Memory Consumption to create node

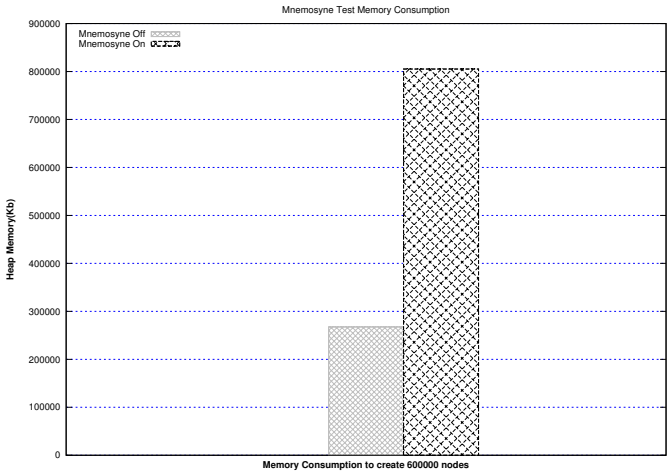

Figure 4.15: Max Memory Consumption to create node

In the graph above, the yellow line and blue line corresponding to Mnemosyne on and off. As we can see, The yellow line is much longer. Which means that Mnemosyne effected a lot of time and memory to finish the task. Both of two lines show that, the longer time, the higher memory has to be used.

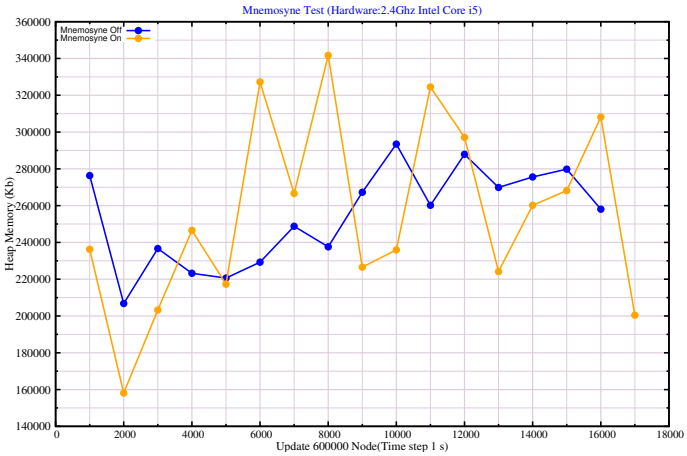

Figure 4.16: Memory Consumption to update node

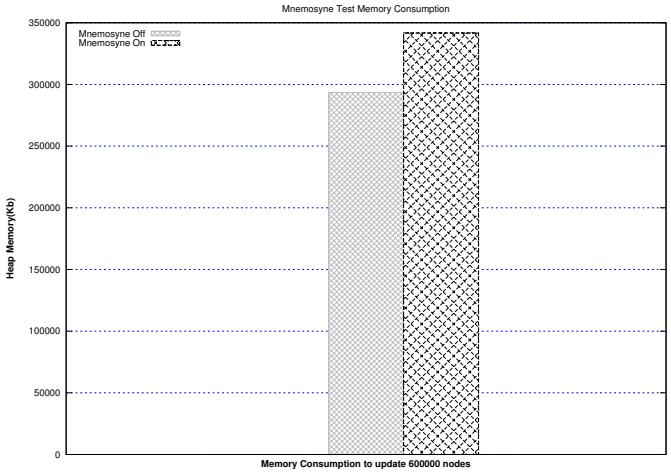

Figure 4.17: Max Memory Consumption to update node

In the task to update node, the memory consumption is similar between using and without using Mnemosyne. So, there is no big effect in memory to use Mnemosyne.

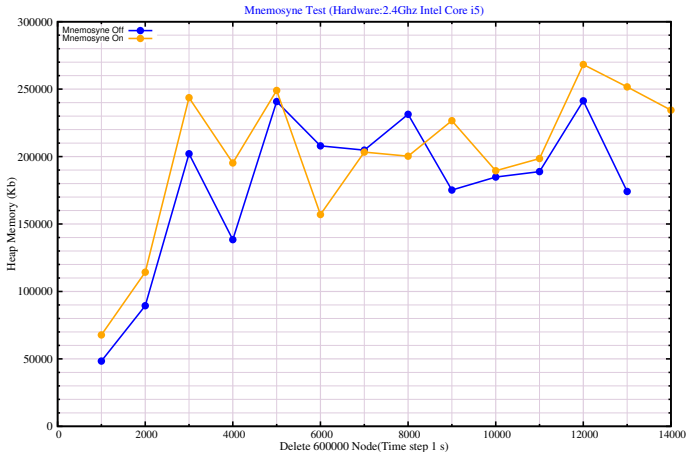

Figure 4.18: Memory consumption to delete node

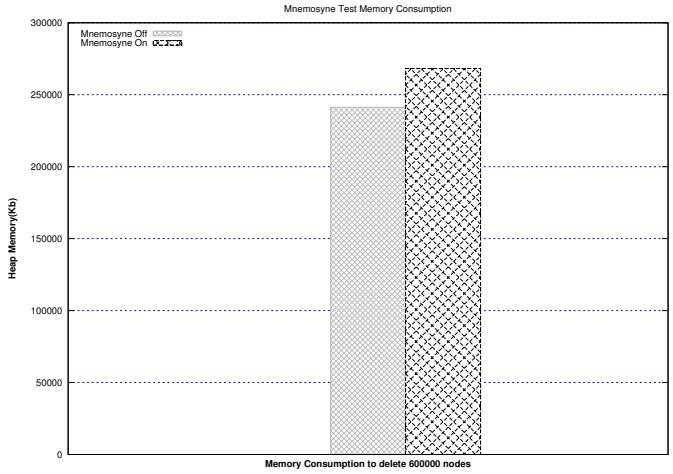

Figure 4.19: Max memory consumption to delete node

In the task of deleting node. the memory consumption is also similar between using and without using Mnemosyne. the yellow line seems higher then blue line means that mnemosyne takes a bit higher memory than usual.

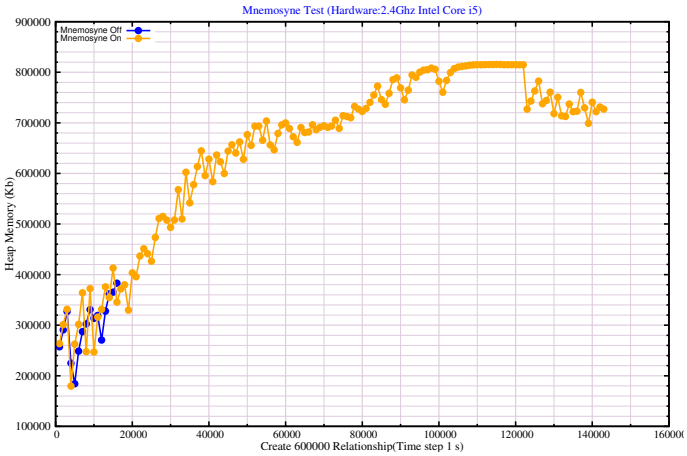

Figure 4.20: Memory consumption to create relationship

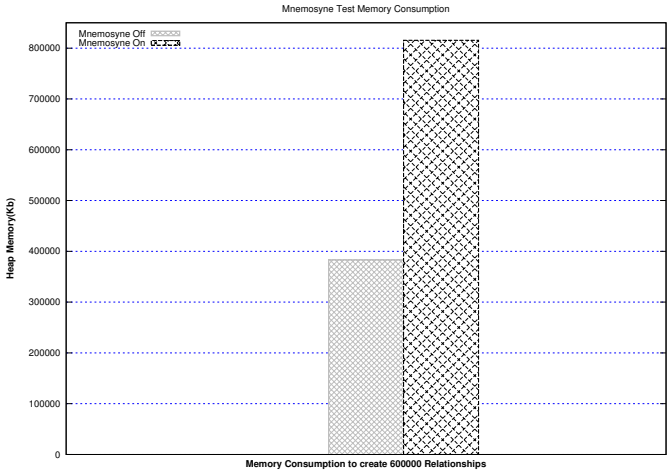

Figure 4.21: Max memory consumption to create relationship

In the test of creating node, we see the graph is similar with creating node. Where the yellow line is much more longer and higher than blue line. it says that, the Mnemosyne takes longer time and higher memory than disabling it.

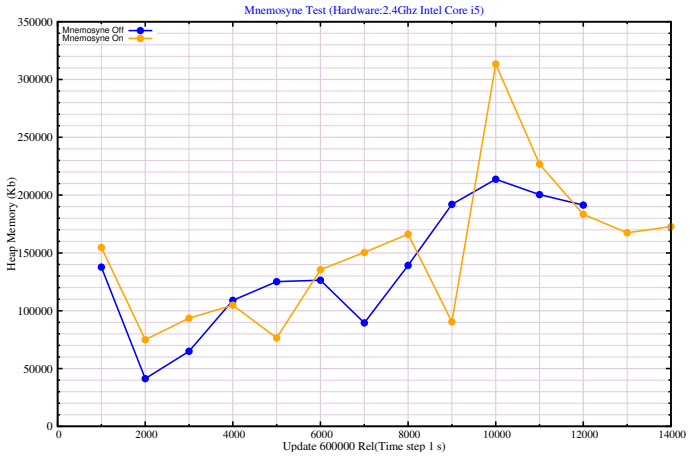

Figure 4.22: Memory consumption to update relationship

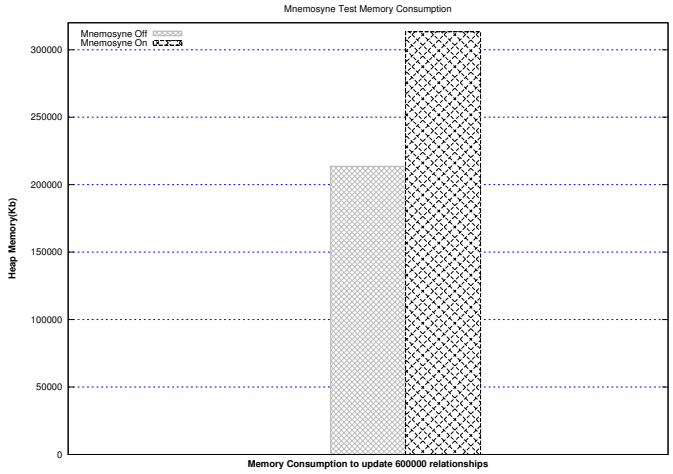

Figure 4.23: Max memory consumption to update relationship

In this test, the yellow line and blue line keep a similar trend. which means that there is no big difference in memory consumption between using and without using Mnemosyne.

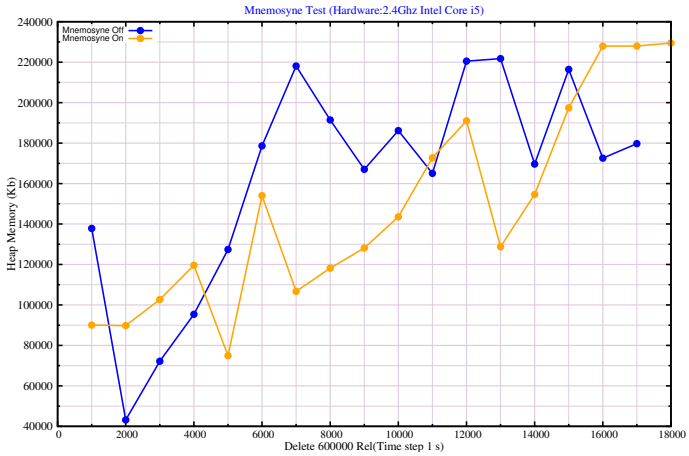

Figure 4.24: Memory consumption to delete relationship

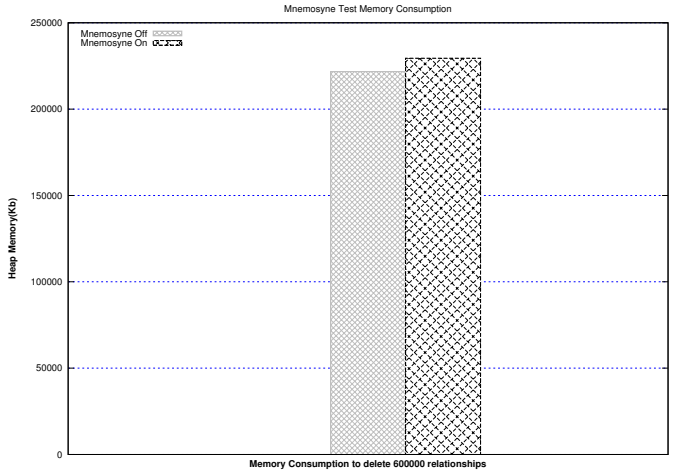

Figure 4.25: Max memory consumption to delete relationship

In this test, it is also similar in memory consumption where yellow line and blue line keep a similar trend.

We have finished the test. In general, We see that there are two big problems in creating nodes and creating relationships using Mnemosyne. The time consumption increases too fast when increasing number of elements. Other actions do not have big difference between using and without using Mnemosyne. In the test of updating node and relationships, the gap between two types of test if not grow if we increase the number of test elements. While deleting nodes and deleting relationships slowly increase. It is similar in memory consumption where there are two big differences in creating node and creating relationship test. Other tests show a very similar result. We can conclude that, Mnemosyne takes longer time and higher memory in creating node and creating relationship. All other actions will not be effected in time and memory.
In graph database, creating nodes and creating relationships are quite frequent operations. the problem that we meet in the test here should be improved in the next version.

4.3 Graph summaries experiment

This part expresses my experiment in extracting type 1 of graph summary. The database will be used is "Cineasts Movies Actors". the sample database contains about 12000 movies, 50000 actors and has size about 12.3 Mb. The database reference is here: [1]

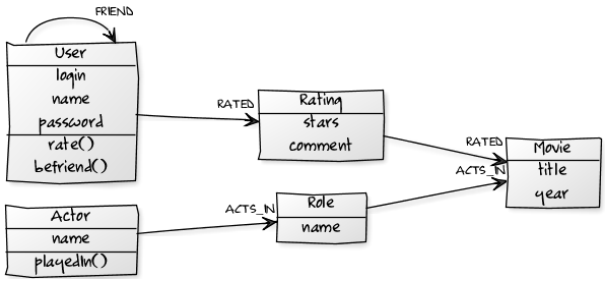

Figure 4.26: Cineasts Movies Actors sample database Model

To extract summaries type 1 from this graph, i have implemented in java all the steps which is presented in part *Extracting Summary*.
After runing the test, we have these results:

T1(Person−[ACTS_IN]−>Movie, Most)=0.74
T2(Person−[ACTS_IN]−>Movie, Few)=0.0
T3(Person−[ACTS_IN]−>Movie, VeryFew)=0.06
T4(User−[FRIEND]−>User, Most)=0.0
T5(User−[FRIEND]−>User, Few)=0.56
T6(User−[FRIEND]−>User, VeryFew)=0.91
T7(Person−[DIRECTED]−>Movie, Most)=0.0
T8(Person−[DIRECTED]−>Movie, Few)=0.19

[1] http://docs.spring.io/spring-data/data-graph/snapshot-site/reference/html/tutorial

$T9(Person-[DIRECTED]->Movie, VeryFew)=0.95$
$T10(User-[RATED]->Movie, Most)=0.72$
$T11(User-[RATED]->Movie, Few)=0.0$
$T12(User-[RATED]->Movie, VeryFew)=0.70$

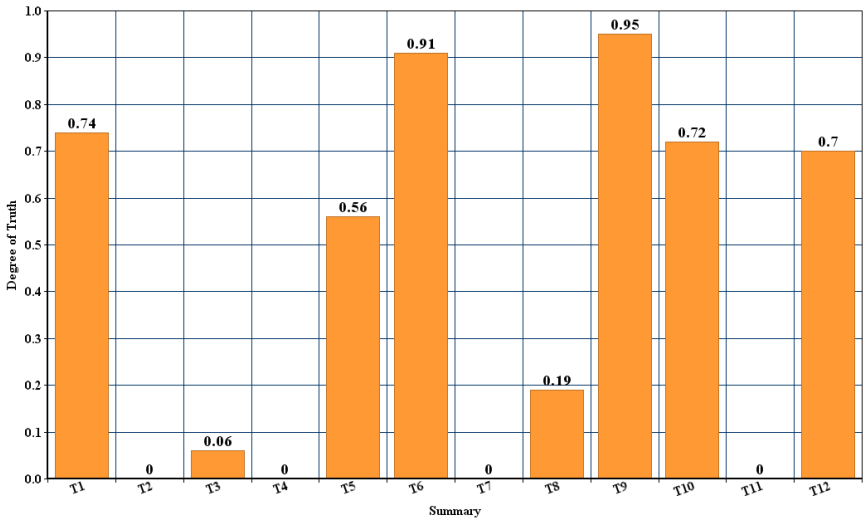

Figure 4.27: Summaries and Degree

To interpret the result as a linguistic summary , let take a example the summary:

T(Person-[ACTS IN]→Movie,Most)=0.74

this can say as linguistic that "Most of Person acts in movies" , and this interpretation has the truth validity which is quite high: 0.74 (74 percent). In this report, we do not consider the process of how to interpret such graph summaries to linguistic summary as human language. we only consider how to extract such structure summaries for graph database.

Chapter 5

Discussion and Conclusion

Nowadays, Data is increasing too fast and it leads to new generation of database. Very few research and proposals exist on this new type of databases, especially in the topic of data summaries. The work I have done in this internship is thus very innovative.

During my research, I have learned about graph database which is dealing with very huge volume of data to be stored and extract. The data can reach to billions of nodes and relationships. More ever, Graph Database embraces well the relationships even better than traditional relational database management systems (RDBMS) [RWE13a]. After the research course, i also know how to model graph database application. These are just some advantages of graph database, that is why, today, more and more organizations choose this new generation of database. After being expert in graph database, I have started researching about graph data summarization with my team.

My work on Neo4j engine and my research on data summarization have been very interesting. It was difficult becoming accustomed to these technologies and methods, but I am now knowledgeable on both theoretical and practical skills.

In the research, i have contributed my ability in the work of testing and using graph database system, graph summary and Mnemosyne system which is developed by our research team, a versioning system for Neo4j graph database engine. Some of works have been researched by our research team from this extension such as: Fuzzy Historical Graph Pattern Matching, Fraud Ring Detection for Bank.

In this work, we have proposed three types of summaries for NoSQL graph databases and we have tested the proposal on several databases. This work is a first attempt and will be continued to other aspects, to the other kinds of data such as complex data, spatial information, open data, master data.

My future work is to continued in graph summaries. Because in this internship, it is not full experimented. it still lacks second type and third type. In addition, it needs to be experimented in large database by high performance computers.

Bibliography

[ABR13] Renzo Angles, Pablo Barceló, and Gonzalo Ríos. A practical query language for graph dbs. In *7th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, 2013.

[AG08] Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.

[AH07] Sören Auer and Heinrich Herre. A versioning and evolution framework for rdf knowledge bases. In *Proceedings of the 6th international Andrei Ershov memorial conference on Perspectives of systems informatics*, PSI'06, pages 55–69, Berlin, Heidelberg, 2007. Springer-Verlag.

[AKJ03] Peter Jeavons Andrei Krokhin and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of allen's interval algebra. *Journal of ACM 50(5)*, pages 591–640, 2003.

[Bas14] Kenny Bastani. In *Bank Fraud Detection*, 2014.

[BDM09] Jie Bao, Li Ding, and Deborah L. Mcguinness. Semantic history: Towards modeling and publishing changes of online semantic data. In *CEUR*, 2009.

[BG02] Kent Beck and Erich Gamma. Junit cookbook. *Available on-line at: http://JUnit. sourceforge. net/doc/cookbook/cookbook. htm*, 2002.

[Bin00] Robert Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional, 2000.

[BLC04] T. Berners-Lee and D. Connolly. Delta: an ontology for the distribution of differences between rdf graphs. http://www.w3.org/designissues/diff, 2004.

[Bon76] John Adrian Bondy. *Graph Theory With Applications*. Elsevier Science Ltd, 1976.

[Bro09] F. Brouard. Historisation des données partie 1 mode colonne, 2009.

[BT11] Pablo Barceló and Val Tannen, editors. *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management, Santiago, Chile, May 9-12, 2011*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[Cat10] Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2010.

[CAW98] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proc. of the ICDE Conf.*, 1998.

[CFSV04] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004.

[CGM97] S. S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *SIGMOD*, 1997.

[CL13a] A. Castelltort and A. Laurent. Representing history in graph-oriented nosql databases: A versioning system. In *Proc. of the Int. Conf. on Digital Information Management*, 2013.

[CL13b] A. Castelltort and A. Laurent. Rabbithole pullrequest 59, https://github.com/neo4j-contrib/rabbithole/pull/59, July 2013.

[CL14] A. Castelltort and A. Laurent. Fuzzy queries over nosql graph databases: Perspectives for extending the cypher language. In *International Conference on Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer, 2014.

[GGR01] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Mining and monitoring evolving data. *IEEE TKDE*, 13(1), 2001.

[GS96] R George and R Srikanth. Data summarization using genetic algorithms and fuzzy logic. *Genetic Algorithms and Soft Computing*, pages 599–611, 1996.

[Gue13] Giovanna Guerrini, editor. *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13, Genoa, Italy, March 22, 2013, Workshop Proceedings*. ACM, 2013.

[Gye01] Attila Gyenesei. A fuzzy approach for mining quantitative association rules. *Acta Cybern.*, 15(2):305–320, 2001.

[HGW04] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the 10t ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004), August 22-25, 2004, Seattle, WA, USA*, pages 158–167. ACM Press, New York, NY, USA, 2004.

[HHLD11] Jing Han, E. Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Proc. of the 6th International Conference on Pervasive Computing and Applications (ICPCA)*, pages 363–366, 2011.

[HLW01] Tzung-Pei Hong, Kuie-Ying Lin, and Shyue-Liang Wang. Mining fuzzy sequential patterns from multiple-item transactions. In *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, volume 3, pages 1317–1321. IEEE, 2001.

[HP13] Florian Holzschuher and René Peinl. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In Guerrini [Gue13], pages 195–204.

[HS08] Huahai He and Ambuj K. Singh. Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 405–418, New York, NY, USA, 2008. ACM.

[JJZ13] Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors. *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. IEEE Computer Society, 2013.

[KD13] Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In Jensen et al. [JJZ13], pages 997–1008.

[KENM06a] A. Kupfer, S. Eckstein, Karl Neumann, and B. Mathiak. Keeping track of changes in database schemas and related ontologies. In *7. Int. Conf. on Databases and Information Systems*, pages 63–68. IEEE, July 2006.

[KENM06b] A. Kupfer, S. Eckstein, Karl Neumann, and B. Mathiak. Keeping track of changes in database schemas and related ontologies. In *7. Int. Baltic Conference on Databases and Information Systems*, pages 63–68. IEEE, 2006.

[KFW98] Chan Man Kuok, Ada Fu, and Man Hon Wong. Mining fuzzy association rules in databases. *ACM Sigmod Record*, 27(1):41–46, 1998.

[KK01] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society.

[LCK10] Ki Yong Lee, Yon Dohn Chung, and Myoung Ho Kim. An efficient method for maintaining data cubes incrementally. *Inf. Sci.*, 180(6):928–948, March 2010.

[LTVT12] Anna Leontjeva, Konstantin Tretyakov, Jaak Vilo, and Taavi Tamkivi. Fraud detection: Methods of analysis for hypergraph data. In *ASONAM*, pages 1060–1064. IEEE Computer Society, 2012.

[Mav11] In Maven. Apache maven project, 2011.

[MSB11] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[Neo] Neo4j. Cypher - refcard, http://docs.neo4j.org/refcard/1.9/.

[Neu10] Peter Neubauer. Graph databases, nosql and neo4j, 2010.

[Per02] J Perry. *Java Management Extensions*. O'Reilly Media, Inc., 2002.

[PFF+13] Vicky Papavasileiou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. High-level change detection in rdf(s) kbs. *ACM Trans. Database Syst.*, 38(1):1:1–1:42, April 2013.

[pro] OpenSource project. Rabbithole web-based console repository, https://github.com/neo4j-contrib/rabbithole.

[PTC07] P. Plessers, O. D. Troyer, and S. Casteleyn. Understanding ontology evolution: A change detection approach. *J. Web Sem.*, 5(1), 2007.

[PVW13] Jonas Partner, Aleksa Vukotic, and Nicki Watt. *Neo4j In Action*. Manning, 2013.

[RN10] Marko A. Rodriguez and Peter Neubauer. The graph traversal pattern. *CoRR*, abs/1004.1001, 2010.

[RT11] Juan L. Reutter and Tony Tan. A formalism for graph databases and its model of computation. In Barceló and Tannen [BT11].

[RWE13a] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly, 2013.

[RWE13b] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, 2013.

[SCK08] Steven Schockaert, Martine De Cock, and Etienne E. Kerre. Fuzzifying allen's temporal interval relations. *IEEE T. Fuzzy Systems*, 16(2):517–533, 2008.

[Sen92] Stephanie Seneff. Tina: A natural language system for spoken language applications. *Computational linguistics*, 18(1):61–86, 1992.

[SR14] G. Sadowski and P. Rathle. Fraud detection: Discovering connections with graph databases. In *White Paper - Neo Technology - Graphs are Everywhere*, 2014.

[Swa02] Aaron Swartz. Musicbrainz: A semantic web service. *Intelligent Systems, IEEE*, 17(1):76–77, 2002.

[TB09] J. Tappolet and A. Bernstein. Applied temporal rdf: Efcient temporal querying of rdf data with sparql. In *ESWC*, 2009.

[Tho13] ThoughtWorks. Technology advisory board, http://thoughtworks.fileburst.com/assets/technology-radar-may-2013.pdf, May 2013.

[TKS02] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 32–41, New York, NY, USA, 2002. ACM.

[Vai13] Gaurav Vaish. *Getting Started with NoSQL*. Packt Publishing Ltd, 2013.

[W3C09] W3C. Allegrograph rdfstore web 3.0's database, 2009.

[Wik] Wikipedia. Pattern matching, http://en.wikipedia.org/wiki/pattern_matching.

[WK+10] Thomas Williams, Colin Kelley, et al. Gnuplot 4.4: an interactive plotting program. *Official gnuplot documentation, http://sourceforge. net/projects/gnuplot*, 2010.

[Woo12] Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, April 2012.

[WZC94] J.T.L. Wang, K. Zhang, and G.W. Chirn. The approximate graph matching problem. *Pattern Recognition, Proceedings of the 12th International Conference on IAPR*, 94:284–288, 1994.

[YH02] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 721–, Washington, DC, USA, 2002. IEEE Computer Society.

[Zim92] HJ Zimmermann. *Fuzzy Set Theory and Its Applications Second, Revised Edition.* Springer, 1992.