Universidade Federal de Pernambuco

Centro de Informática

Departamento de Informática

Pós-graduação em Ciência da Computação

Using OLAP Queries for Data Analysis on Graph Databases

Nicolle Chaves Cysneiros

Dissertação de Mestrado

Recife

01 de setembro de 2017

Universidade Federal de Pernambuco

Centro de Informática

Departamento de Informática

Nicolle Chaves Cysneiros

Using OLAP Queries for Data Analysis on Graph Databases

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Departamento de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: *Profª. Drª. Ana Carolina Salgado*

Recife

01 de setembro de 2017

Ao G e E do meu GEN

Agradecimentos

Agradeço aos meus pais, por terem me apoiado durante todo o período desse projeto e por terem batalhado durante toda a minha vida para me proporcionar oportunidades incríveis como esse Mestrado. Agradeço à Professora Carol, que atuou como minha mentora desde minha graduação, me orientando e me ajudando nos caminhos que me trouxeram até aqui. Agradeço também aos meus amigos Luiz, Tomaz e todo o time Labcodes por terem me escutado e me dado forças durante os momentos mais difíceis dessa jornada.

Resumo

Bancos de Dados (BDs) em Grafo são uma alternativa aos tradicionais BDs Relacionais e permitem uma melhor escalabilidade do sistema, além de uma maneira mais natural de representar dados altamente conectados. Os BDs em Grafo também permitem diferentes tipos de análises em grafos, como medidas de centralidade e algoritmos de detecção de comunidades. Apesar disso, ainda não existem ferramentas disponíveis no mercado para fazer análise multidimensional em grafos, como os sistemas OLAP existentes que operam sobre BDs Relacionais. No meio acadêmico, existem algumas propostas de frameworks que visam a construção de um cubo multidimensional composto por grafos agregados, obtidos a partir da combinação de nós e arestas do grafo original de acordo com as dimensões e medidas analisadas. Contudo, a maior parte das pesquisas são voltadas para a análise de grafos homogêneos, enquanto os trabalhos que se dedicam a grafos heterogêneos realizam a análise multidimensional a partir de um modelo intermediário do dado original. Esse projeto propõe um sistema para a realização de consultas OLAP em um Banco de Dados em Grafo sem a necessidade da geração de um modelo intermediário de dados para realizar análise em grafos heterogêneos. O sistema proposto é capaz de responder consultas OLAP a partir de grafos agregados extraídos do grafo original, além de também realizar análises acerca da topologia do grafo. Neste trabalho são apresentados experimentos mostrando a eficácia do sistema para responder às consultas analíticas e comparações específicas entre o sistema descrito e as soluções existentes.

Palavras-chave: OLAP, Bando de Dados em Grafo, Grafos, Análise de Dados

Abstract

Graph Databases (GDB) are an alternative to traditional Relational Databases and allow a better scalability for the system, in addition to representing highly connected data in a more natural way. GDBs also support different kind of network analysis, such as centrality measures and community detection algorithms. Despite this, there are still no tools available in the market for multidimensional analysis in graphs, such as existing OLAP systems that operate on Relational DBs. In the academic field, there are some framework proposals that aim at the construction of a multidimensional cube composed by aggregate graphs, which are obtained from the combination of vertices and edges of the original graph, according to the dimensions and measures being analysed. However, most part of the researches in this area are focused on the OLAP analysis for homogeneous graphs, while the works dedicated to heterogeneous graphs require an intermediate data model in order to execute the multidimensional analysis. This project proposes a system to execute OLAP queries in a Graph Database without the need to generate an intermediate data model to do multidimensional analysis on heterogeneous graphs. The proposed system is able to answer OLAP queries using aggregate graphs obtained from the original graph, as well as execute analysis about the topology of the graph. In this work, we present experiments showing the effectiveness of the system to answer the analytical queries and some qualitative comparisons between the proposed system and existing solutions.

Keywords: OLAP, Graph Databases, Graphs, Data Analysis

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we will present the motivations for the realisation of this work and give a clear definition of the problem to be addressed. The general and specific objectives of this research will be listed, as well as the remaining structure of the document.

1.1 Motivation

In recent years, our ability to collect data from different sources has increased significantly [**?**]. We can retrieve data from different devices, with different formats and different levels of connection. However, our capability to store, process and analyse these large collections of connected data has still opportunity for improvement.

For this reason, Graph Databases (GDBs) have been gaining attention in the database community due to the good performance when dealing with highly connected data. In comparison to Relational Databases, where the execution performance of a query that requires intensive join operations deteriorates proportionally to data size, Graph Databases performance remains constant with respect to the size of the graph [**?**]. GDBs allow a more natural way to represent data as vertices and edges. Social networks, semantic web pages and recommendation systems are some examples of applications which handle data relationships and could perform better if the data were stored in a GDB [**?**].

GDBs also provide a flexible data model, where the main information stored is the

relationship between entities. This feature allows network processing based analysis to be done, such as pattern detection, edge path analysis and clustering techniques. These different analysis techniques allow the development of solutions for challenging problems, not only those approaches usually applied in traditional relational databases or data warehousing [**?**].

Our main motivation comes from the fact that there are no consolidate tools that can execute both network and multidimensional analysis on a Graph Database. For instance, a social network implemented using a GDB could take advantage of performing analysis over the topology of the network formed by the users, but it would fail in performing multidimensional analysis that would result in Business Intelligence centered reports. In the academic field, there are some framework proposals that aim at the construction of a multidimensional cube composed by aggregate graphs, which are obtained from the combination of vertices and edges of the original graph, according to the dimensions and measures being analysed. However, most part of the researches in this area are focused on the OLAP analysis for homogeneous graphs (graphs with only one type of vertex), while the works dedicated to heterogeneous graphs (graphs with more than one type of vertices) require an intermediate data model in order to execute the multidimensional analysis. The need of a intermediate data model means that the operational data would have to be parsed into the new data model in order to be analysed, including one more step to the analysis process.

1.2 Problem Definition

Given the scenario described in the previous section, we investigate the problem to be addressed from the question: “How can we execute both network and multidimensional analysis on heterogeneous graphs data without the need to generate an intermediate data model?”. Considering the question proposed, the problem can be defined as: given a graph $G = (V, E)$, with a set of vertices V and a set of edges E, what is the architecture and how is the operation of a system that is able to execute network analysis algorithms and OLAP queries over the graph G, without generating an intermediate data model.

1.3 Objectives

The general objective of this work is to build a system that supports the execution of network and multidimensional analysis on a Graph Database, without the generation of an intermediate data model for the graph. In order to achieve the general objective, some specific objectives were considered:

- Generate multidimensional view of the original data without changing the data model adopted by the operational part of the system
- Give support for OLAP operations (roll up, drill down, slice and dice) to be executed over the multidimensional view
- Give support for network analysis algorithms to be executed over the original data
- Define the architecture of the complete system

1.4 Expected Contributions

Once the main problem and the objectives are defined, we expect from this work the following contributions:

- Architecture definition, specification of an algorithm to generate aggregate graphs and implementation of a prototype for a Data Analysis System for Graph Databases.
- Implementation of OLAP operators and network analysis algorithms, providing a comprehensive analysis of the graph data.
- Experiments and qualitative analysis in comparison with existing frameworks.

1.5 Document Structure

The rest of the document is organized as follows:

Chapter 2 Introduces the main concepts related to the theoretical foundation of this work, such as OLAP systems, Graphs and Graph Databases.

Chapter 3 Gives an overview of the state of the art for OLAP system with Graph Databases, presenting the main frameworks proposed in this area.

Chapter 4 Presents the specification of the system proposed, detailing the architecture and the main components of the solution implemented.

Chapter 5 Shows how the system was implemented and describes some experiments and the obtained results.

Chapter 6 Concludes the document, recapitulating the work presented and giving insights for future work.

Chapter 2

OLAP and Graph Databases

This chapter introduces the main concepts that forms the theoretical foundation necessary to better understand the work presented in this dissertation. Initially, we will explore the definition of Data Warehouse, Multidimensional Model and OLAP tools. Then, we will dive into concepts related to Graph and Graph Databases.

2.1 Data Warehouse

According to [**?**], a data warehouse (DW) is "a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions". The first important aspect of a data warehouse is that it is subject-oriented, which means that the information stored in the DW is related to the company's subject. Consider a retail company for example: the main subjects can be product, sale, vendor and customer, therefore the data in the warehouse will be related to these entities.

The second characteristic of a data warehouse is that it is integrated, which means it contains data coming from multiple different sources. During the process of loading the warehouse, the data is converted, formatted, normalised and go through any other process to make the final information stored in the warehouse consistent. Another important aspect of a data warehouse is that it is nonvolatile, which means that the data in the warehouse does not get updated as operational environment. The warehouse is loaded in batches and it stores

a snapshot, creating a history of the data. The final important aspect of a DW is that every record of data contains some sort of a time attribute to mark the moment in which the record is accurate.

Figure **??** shows the basic elements of a data warehouse [**?**]. The Operational Source Systems capture and store the transactions of the business and they are considered elements outside of the data warehouse, since there is no control on the content or the format of the data that they store. The Data Staging Area is where the process of Extract-Transformation-Load (ETL) is made: the data is extracted from the operational source system, then they are transformed in order to integrate all the information in a consistent format, and finally the data is loaded into the presentation area.

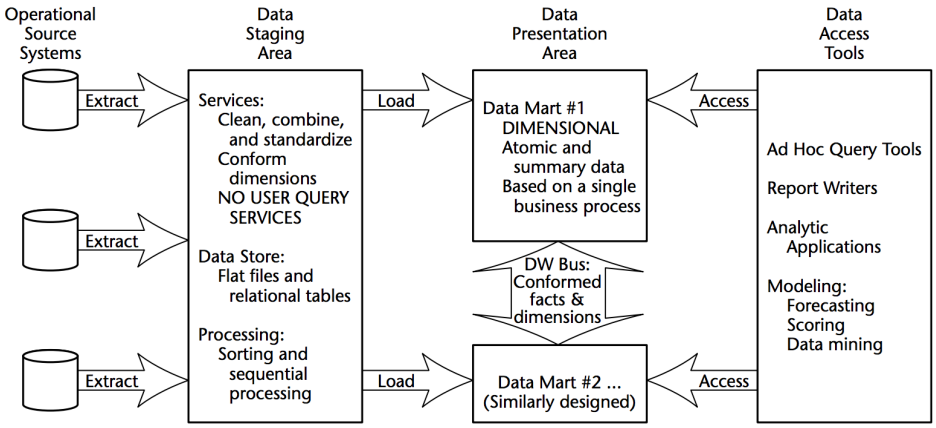


Figure 2.1: Elements of a Data Warehouse [**?**]

The Data Presentation Area is where the data is organised, stored and accessed by analytical applications. According to [**?**], this area should be composed by a series of integrated data marts, which are repositories that present the data for a single process of the organisational business. A data mart stores atomic data and organizes them in a model that is more legible to humans. The most popular technology used to implement data marts adopted by the industry is On-Line Analytical Processing (OLAP), that will be covered in more detail along this chapter.

Finally, the Data Access Tools query the data in the presentation area. This element is formed by a set of different applications, from simple ad hoc queries until complex data mining application.

2.2 Multidimensional Model

The main functionality of a data warehouse is to facilitate multidimensional analysis [**?**]. This type of analysis reduces the number of misinterpretations by aligning the data with the analyst's mental model of the business. The multidimensional analysis provides an easy navigation through the database, showing specific subsets of data in different orientations and executing analytical calculations.

In order to perform multidimensional analysis, the data must be organised in a multidimensional model, where information is stored in a multidimensional array called *hypercube* or *cube* [**?**]. A cube is composed by cells that store measures aggregated by the data dimensions. A *dimension* is a cube's structural attribute formed by a list of properties that are similar to each other according to the user's perception of the data. Measures are the values being analysed by the user. Figure **??** shows an example of the dimensions and measures extracted from a sales application.

As shown in Figure **??** , each dimension (Geography, Time and Item) can be associated with an hierarchy of data aggregation levels. This feature allows user to view the data from different levels of details, for example: the measure *Sales* aggregated by *Region* can be detailed by *Country* or even further detailed by *City*.

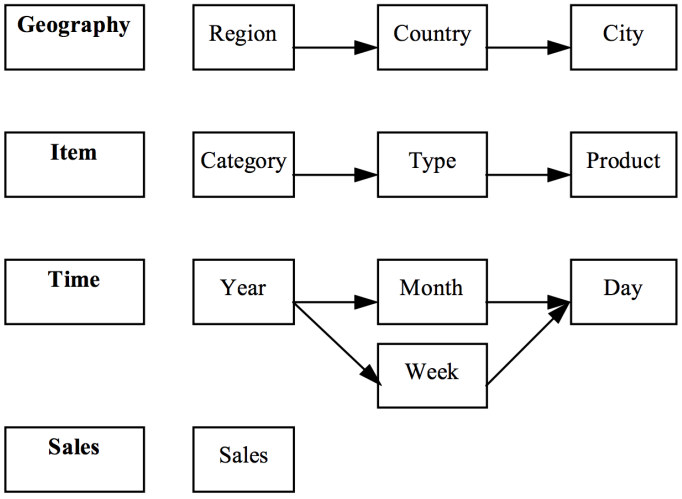


Figure 2.2: Example of dimensions and measure for a sales application [**?**]

2.3 OLAP

On-Line Analytical Processing (OLAP) is a category of technological tools used by analysts and managers to extract new knowledge from consolidated enterprise data [**?**]. In order to achieve this goal, data are organised in cubes and stored following multidimensional models. The access to this data should be fast, consistent and interactive and it should provide various views of information, reflecting the different dimensions of an enterprise as perceived by the user.

Usually, the data loaded into an OLAP system comes from a data warehouse. According to [**?**], the relationship between OLAP systems and data warehouse is complimentary: while OLAP offers control and flexible ways to explore data in different dimensions and hierarchy levels, data warehouse provides a robust data source for the OLAP system, where up-to-date data is available, already extracted and properly integrated.

2.3.1 Types of OLAP Systems

There are two approaches for the physical model of an OLAP system [**?**]: Multidimensional On-Line Analytical Processing (MOLAP) and Relational On-Line Analytical Processing (ROLAP) Architectures.The MOLAP architecture provides a direct multidimensional view of the data. This approach stores the data in a Multidimensional Database Management System (MDBMS), which uses n-dimensional arrays that contains the measures of the cube. This type of DBMS has a better performance than traditional Relational Databases, but it is more difficult to manage updates.

The ROLAP architecture is a multidimensional interface to relational data. This approach uses a traditional Relational Database Management System (RDBMS) to store the data organised in a star or snowflake schema. A star schema is formed by one or more dimension tables and one centralised fact table, which stores the measures of interest for the OLAP system. Figure **??** shows an example of a star schema, where the tables TIME, GEOGRAPHY, ACCOUNT and PRODUCT are dimension tables and SALES is the fact table.

Despite MOLAP architecture has the advantage of relying on a multidimensional na-

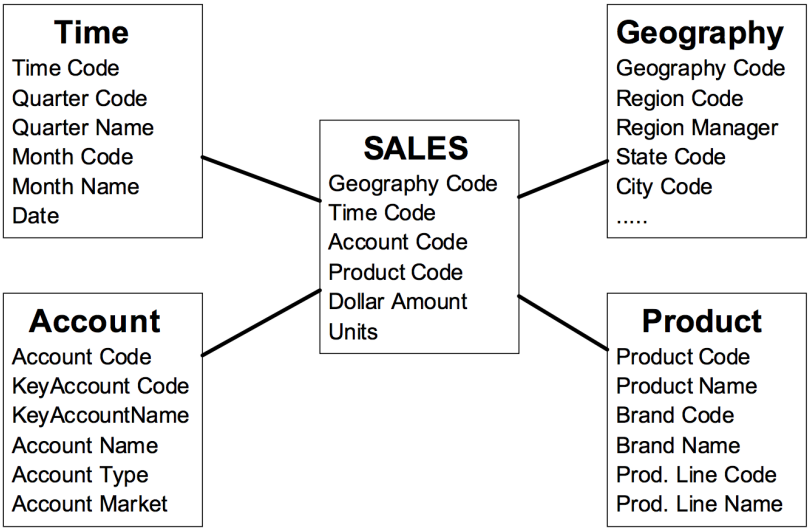


Figure 2.3: Example of a Star Schema [**?**]

tive storage mechanism, the ROLAP approach allows easy integration with existing relational database systems and has the advantage of relational data being more efficiently stored than multidimensional data.

2.3.2 OLAP Operators

As mentioned earlier in this chapter, an OLAP system should provide a fast and interactive way for the user to explore the data stored in the cube. These are the main OLAP operations that can be used to navigate and manipulate different dimensions of the data [**?**]:

Roll-up Aggregates information along one dimension, summarising data on a higher level in its hierarchy. Consider the dimensions shown in Figure **??** and the measure of total dollar amount of sales per city, we can perform a roll-up operation to obtain the total dollar amount of sale per state.

Drill-down Detail information along one dimension, navigating the data from a higher to a lower hierarchy. Consider the measure of total dollar amount of sales per year, we can drill-down this query to obtain the total dollar amount of sales per month.

Slice Selects a slice of the cube according to user-specified dimension values. For instance, the user can select to view the total dollar amount of sales for the year of 2016.

Dice Selects a subcube from the original data according to user-specified conditions referred to more than one dimension. For instance, the user can select to view the total dollar amount of sales for the year of 2016 in the city of Recife.

Pivot Changes the orientation of dimensions in the cube, i.e. swap a row dimension to a column dimension.

There are other operations that can be performed in a OLAP system, but the ones shown above are considered the basic set of operations to support dynamic multidimensional analysis [**?**].

2.4 Graph

Graph is a data structure formed by a set of vertices $V = \{v_1, \ldots, v_n\}$ and a set of pair of vertices called edges $E = \{v_1v_2, \ldots, v_nv_m\}$. It can be represented graphically (where the vertices are shown as circles and edges are shown as lines, as shown in Figure **??**) or mathematically in the form $G = (V, E)$ [**?**].

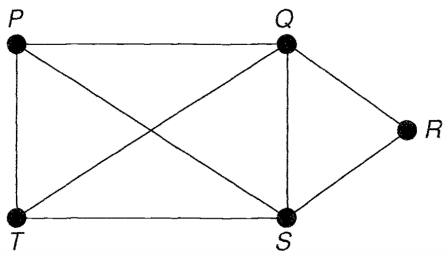


Figure 2.4: Example of graph [**?**]

There are several real world applications that can take advantage of a graph model, specially the ones where the relationship between entities is an important information to be represented [**?**]. Consider, for example, a social network application similar to Twitter, where

a user can follow another user. In this scenario, we can represent a user as a vertex and the relationship between users as an edge, as shown in Figure **??**

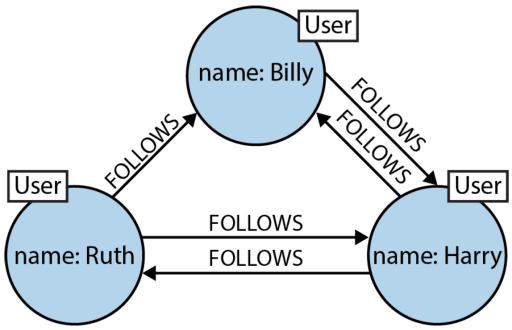


Figure 2.5: Example of graph representation of a social network [**?**]

2.4.1 Graph Theory

Graph Theory is a branch of Mathematics dedicated to the study of graph structures [**?**]. Given the definition of a general graph explained above, there are several classifications and concepts associated with graph structures.

Loop When an edge starts and finishes in the same vertex, as the one shown in Figure **??** starting at vertex T and finishing at vertex T.

Multigraph When a graph allows multiple edges connecting the same pair of vertices, as illustrated in Figure **??** with two edges connecting vertices Q and R.

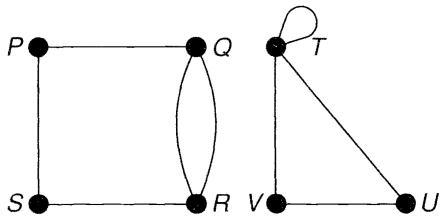


Figure 2.6: Example of a multigraph with a loop [**?**]

Simple Graph When a graph does not have loops or multiple edges, as the one depicted in Figure **??**.

Complete Graph When each pair of distinct vertices are connected to each other, as shown in Figure **??**.

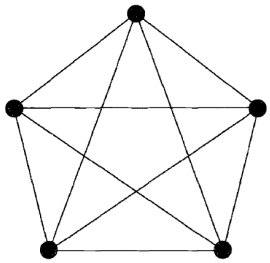


Figure 2.7: Example of a complete graph [**?**]

Directed Graph When the edges have directions expliciting the start and end of the connection, as illustrated in Figure **??**.

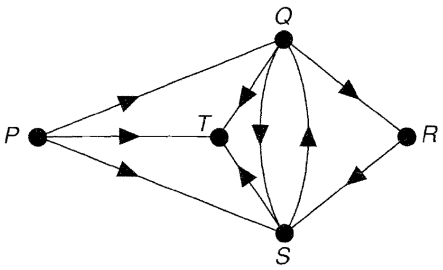


Figure 2.8: Example of a directed graph [**?**]

Homogeneous Graph It is a graph that has only one type of vertex.

Heterogeneous Graph It is a graph that has different types of vertices.

Subgraph It is a graph obtained from another graph *G*, where its vertices are a subset of the vertices of *G* and its edges are a subset of the edges of *G*. Figure **??** shows in **??** a subgraph of the graph in **??**.

Path It is a subgraph obtained by a sequence of adjacent and distinct vertices.

There are other types of classifications of a graph and other concepts associated to it [**?**]. These are the main definitions for the purposes of understanding the work presented here.

(a) Original graph

(b) Subgraph

Figure 2.9: Example of a subgraph [**?**]

2.4.2 Network Analysis

There are different kind of analysis that can be done depending on the nature of the graph data. Due to the work presented here, we will explore the aspects of social network analysis, which focus on the structure of relationships (edges) between entities (vertices) in the graph. Several types of analysis can be performed in a graph, but the measure of a vertex's centrality is historically one of the most studied cases of analysis [**?**]. A common application for a centrality measure is to find out what are the focal points in a social network, i.e. who are the people that are the most connected to other people? There are three types of centrality measures [**?**]:

Degree Centrality This measure is given by the number of adjacent vertices. Formally, the Degree Centrality C_D of a vertex v_k can be given by

$$C_D(v_k) = \sum_{i=1}^{n} a(v_i, v_k), \tag{2.1}$$

where n is the total number of vertices in the graph and $a(v_i, v_k)$ is 1 if and only if v_i and v_k are connected by one edge, or 0 otherwise.

Considering a social network, a person (vertex) that has the greatest number of connections can be considered one of the focal points of the network, since it can directly read the most vertices in the graph. Given a graph with n vertices, the maximum value the degree centrality of a vertex in the graph can be is $n-1$. The degree centrality of a vertex is related to its potential communication activity.

Betweenness Centrality This measure is given by the frequency in which a vertex is present in the shortest path between other vertices. Formally, the Betweenness Centrality C_B of

a vertex v_k is given by

$$C_B(v_k) = \sum_{i \neq j \neq k}^{n} \frac{\sigma_{ij}(v_k)}{\sigma_{ij}}, \tag{2.2}$$

where σ_{ij} is the total number of shortest paths from vertex v_i to vertex v_j and $\sigma_{ij}(v_k)$ is the total number of shortest paths from v_i to v_j that passes through v_k.

Considering a social network, a person with high betweenness centrality is capable of influencing others by intercepting connections between several vertices in the graph. The betweenness centrality of a vertex is related to its potential control of communication.

Closeness Centrality This measure is given by the inverse of the sum distance of the shortest path of a vertex between other vertices in the graph. Formally, the Closeness Centrality C_C of a vertex v_k is given by

$$C_C(v_k) = \frac{1}{\sum_{i=1}^{n} d(v_i, v_k)} \tag{2.3}$$

where $d(v_i, v_k)$ is the number of edges in the shortest path from v_i to v_k.

Considering a social network, a person with high closeness centrality do not depend as much on others to communicate with other people in the network. The closeness centrality of a vertex is related to its potential independency and efficiency to control communication.

2.5 Graph Databases

Graph Databases are an alternative to Relational Database Management Systems (RDBMS), which are commonly used in the industry since the early 1980's. Despite the popularity of RDBMSs, GDBs allow the storage of data in graph model, which is a more natural way to represent information for some applications, such as social networks, semantic web pages and recommendation systems [**?**].

The most popular form of graph model is the labeled property graph model [**?**]. The main characteristics of a labeled property graph are:

- Vertices and edges can contain properties, i.e. key-value pairs
- Vertices can be labeled with one or more labels
- Edges are labeled and directed, i.e. always have start and end vertices

The formal definition for a labeled property graph is given by $G = (V, E, L_V, L_E, A_V, A_E)$, where:

- V is a set of vertices
- $E \subseteq V \times V$ is a set of edges
- L_V is a set of vertices labels and L_E is a set of edge labels
- $A_V = \{a_1, \ldots, a_n\}$ is a set of vertex attributes, where $a_i = (k_i, m_i)$ is a key-value pair, k_i is the attribute key and m_i is the attribute value. Each vertex $v_i \in V$ is associated with a set of attributes. $A_E = \{b_1, \ldots, b_n\}$ is the set of edge attributes defined in the same way as vertex attributes.

An example of a labeled property graph was shown in Figure **??**, where the vertices have the label "User" and the property "name" and the relationships are named "Follows" and have arrows indicating the direction of the edge.

2.5.1 Historical Overview

Scientific research related to graph data models were continuously published between 1980's and the first half of 1990's. Then, the database community attention turned to semistructured data model, due to the emergence of XML and the growth of hypertext document applications [**?**]. From this period, the main focus of the graph databases proposed was to establish a better way to represent data and methods to retrieve and manipulate data modelled as graph.

Recently this area has gained again attention from the community, due to the emergence of trendy projects (chemistry, biology, social network, semantic web, among others)

where the importance of information relies not only in the entities but also in the relationship between them [**?**]. Most recent implementations of graph databases are concerned with handling increasing amount of data and improving performance in retrieving and manipulating data.

The most popular graph database system according to DBEngines[1] website is Neo4J[2]. It is an open source native graph storage database system implemented in Java. Neo4J has its own query language called Cypher, which can be used to create, update and retrieve vertices and edges. This graph database system also provides high availability and scalability for big volume of graph data.

Besides Neo4J, there are other popular graph databases system according to DB Engine site. OrientDB[3] is a NoSQL Multi-Model database, which means it stores data in the form of documents, key-value stores, objects, graph and others. Like Neo4J, OrientDB is also implemented in Java, but it uses an extended version of SQL that includes special operators to manipulate graph. This database system allows the creation of a pre-defined data schema and the definition of complex data type, like dates, decimals and binary objects (BLOB).

TitanDB[4] is a native graph database system implemented in Java. In order to establish connection with the hard disk, Titan needs to be linked to a data storage system - Cassandra[5], HBase[6] or BerkeleyDB[7] - that is suitable for the application. The creation of vertices, edges and the submission of queries can be done through a Java API or through a Gremlin server.

2.5.2 Neo4J

As mentioned, Neo4J is the most popular graph database in the industry according to DB Engines website. In comparison with other DBs from the same category, Neo4J has a good performance considering time to process a query and the amount of memory consumed by the database.

[1] https://db-engines.com/en/system/Neo4j
[2] https://neo4j.com/
[3] http://orientdb.com/orientdb/
[4] http://titan.thinkaurelius.com/
[5] http://cassandra.apache.org/
[6] https://hbase.apache.org/
[7] http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html

The queries submitted to Neo4J are written in Cypher [**?**], which is a declarative query language inspired by SQL and that describes graph patterns using ASCII characters. Figure **??** shows an example of how Cypher represents a relationship in the query syntax. This language also allows to create, update and delete vertices and edges. Since Cypher uses the terms “nodes" and “relationships" to refer to “vertices” and “edges” respectively, we will interchange these words accordingly throughout the text from now on.

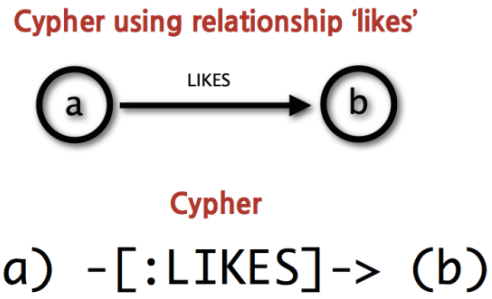


Figure 2.10: Cypher syntax representation of a relationship in the graph [**?**]

Neo4J query language also includes a series of clauses and expressions similar to SQL, such as *WHERE*, *ORDERBY*, *LIMIT*, *AND*, among others. The Cypher query shown in Figure **??** is an example of the general syntax of the language and it shows how it is possible to restrict the results by a certain threshold using the clause *WHERE*. The mentioned query will return a subgraph containing the nodes with labels *Label*1 and *Label*2 that have a relationship of type *TYPE* with a property value above a certain threshold.

```
MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)
WHERE rel.property > {value}
RETURN rel.property, type(rel)
```

Figure 2.11: Example of Cypher Query syntax [**?**]

The data stored in a Neo4J database can be accessed using a Java API or a REST API by default. In order to facilitate the access to the data using a Python application we used an external library called Py2Neo, which wraps REST API requests to execute Cypher commands on the database. In this way, it was possible to implement the algorithm for the Graph Aggregators, where it consumes the original data, generate an aggregate graph and stores

it in another instance of a Neo4J database.

Another interesting feature in Neo4J is its web-based user interface. This interface provides a direct way to submit Cypher queries to the database and a visualisation of the results. Figure **??** is a screenshot of the interface, showing the result of a simple query submitted in the text area at the top of the screen. The submitted query returns a subgraph showing the relationship between the Author Felix Naumann with all his publications present in the database. The interface shows each node as a circle and relationship as an arrow, it also shows the properties and labels of nodes and relationships when they are clicked. The experiments shown in this chapter will be displayed using the graphic interface provided by Neo4J.

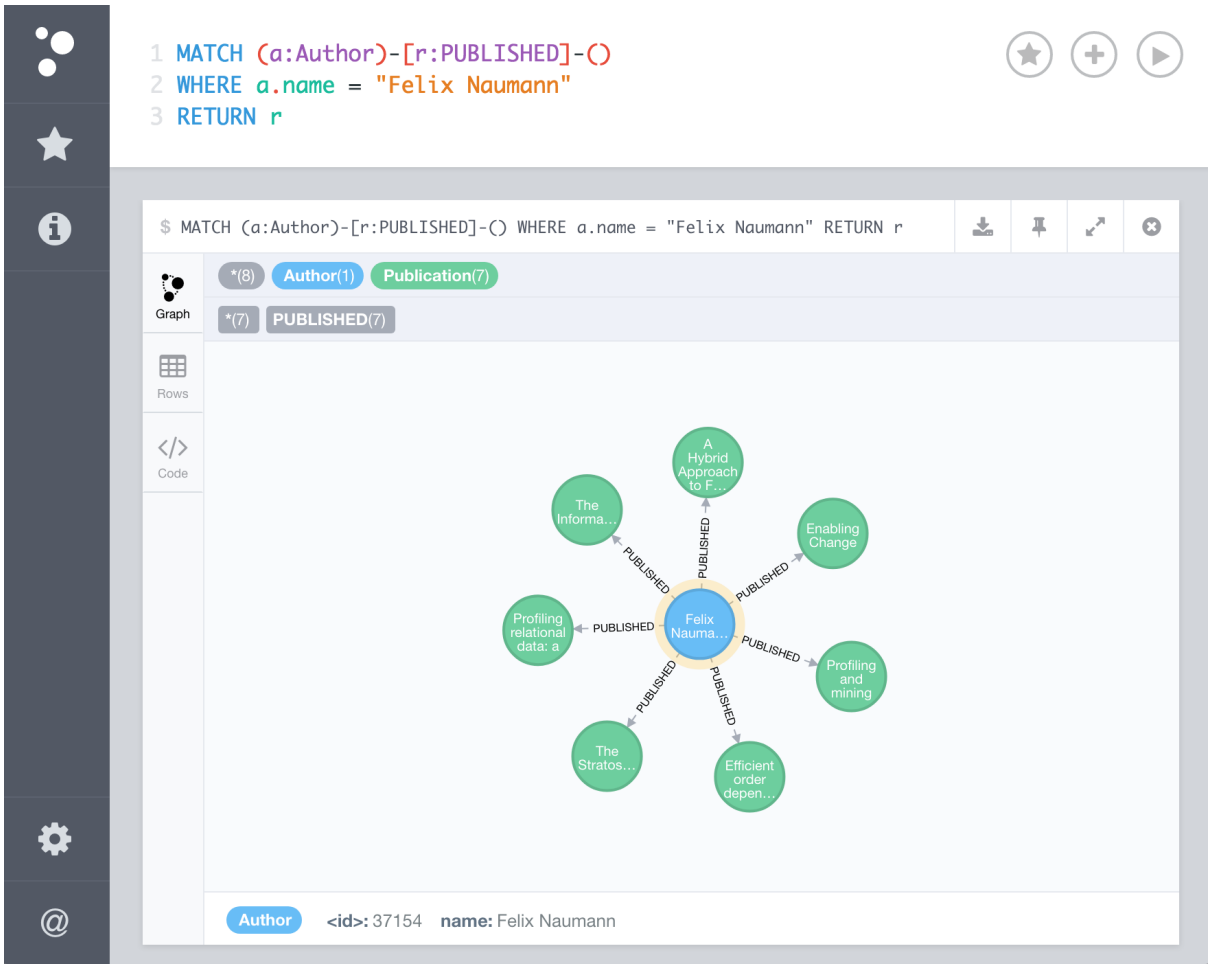


Figure 2.12: Neo4J User Interface

2.6 Final Considerations

In this chapter, we discussed the main concepts related to the work presented in this dissertation. Initially, the definition of Data Warehouse and Multidimensional Model were introduced, followed by a detailed view on different types and operators an OLAP system can have. Then, we covered the main components of a graph and how it can be classified according to different characteristics. Finally, we explained the definition of a Graph Database and went through a historical overview of such systems, as well as a more specific description of Neo4J database. In the next chapter, we will review the most important research published that implements an OLAP system using Graph Databases.

Chapter 3

Graph Cubes: State of the Art

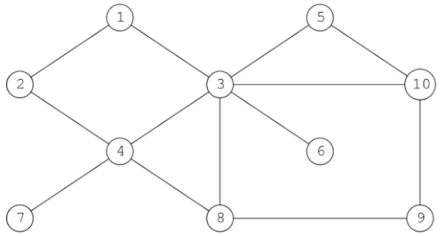
This chapter presents the most important research works published in the area of OLAP systems implemented using graph databases. The first work presented here dates back to 2011 and introduces several concepts - such as graph cube, aggregate graphs and cuboid queries - that serve as basis for other papers published since then.

3.1 Graph Cube: On Warehousing and OLAP Multidimensional Networks

One of the main works in graph analysis using OLAP methods is described in [**?**], and it introduces several concepts that will be used by other authors to describe their work in this area. Graph Cube is one of those concepts and it is defined as a multidimensional model that extends multidimensional networks to provide decision support features. A multidimensional network is a graph $N = (V, E, A)$, where V is a set of vertices, E is a set of edges and A is a set of vertex-specific attributes. Each vertex in the graph is a multidimensional tuple and the attributes of the vertex define the graph cube dimensions.

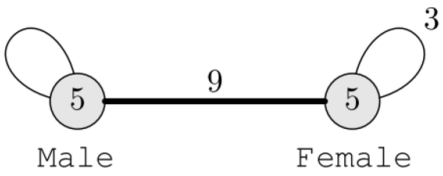
A graph cube is formed by all possible aggregate networks calculated from the original multidimensional network. An aggregate network (often called cuboid) is a summarisation of the original graph with respect to one or more dimensions, which is calculated by applying an aggregate function (e.g. COUNT, SUM, AVERAGE, among others) on the vertices attributes.

Consider the multidimensional network illustrated in Figure **??**.



ID	Gender	Location	Profession	Income
1	Male	CA	Teacher	$70,000
2	Female	WA	Teacher	$65,000
3	Female	CA	Engineer	$80,000
4	Female	NY	Teacher	$90,000
5	Male	IL	Lawyer	$80,000
6	Female	WA	Teacher	$90,000
7	Male	NY	Lawyer	$100,000
8	Male	IL	Engineer	$75,000
9	Female	CA	Lawyer	$120,000
10	Male	IL	Engineer	$95,000

Figure 3.1: Multidimensional network [**?**]



Gender	COUNT(*)
Male	5
Female	5

Figure 3.2: Aggregate Network by *Gender* dimension [**?**]

The vertices of the graph presented in Figure **??** represent individuals and the edges represent the relationship between these individuals. The table on the right side of Figure **??** describes the attributes of each vertex: an unique *ID*, *Gender*, *Location*, *Profession* and *Income*. Figure **??** shows the cuboid obtained by applying the function COUNT on the attribute *Gender*: two vertices represent the possible values for the attribute (Male and Female) and they contain the number of vertices in the original graph with the respective *Gender* value. It is important to notice that the relationship between individuals was also aggregated in the resulting network, e.g. the aggregate network show that there are 9 relationships between male and female individuals in the original graph, which is represented by an edge with weight of 9.

The dimension of a cuboid is given by the set of non-aggregate dimensions of the cuboid. For instance, the cuboid in Figure **??** has the dimension $\{Gender\}$. A cuboid A' is an ancestor of another cuboid A" if $dim(A')$ contains $dim(A'')$. Given these definitions, all possible cuboids of a graph cube can be organised in a graph cube lattice, ordering the cuboids according

to its ancestors. A multidimensional network N with n dimensions has 2^n cuboids in the graph cube. Figure **??** shows the graph cube lattice for the multidimensional network introduced in Figure **??**, considering only *Gender*, *Location* and *Profession* as dimensions.

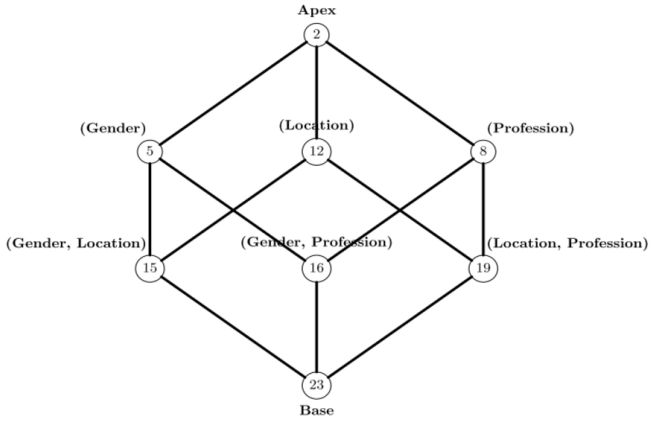


Figure 3.3: Graph Cube Lattice [**?**]

The paper proposes two OLAP query models:

Cuboid Query Aggregate vertices and edges based on the dimension requested in the query and can work with any aggregate function (SUM, AVG, etc). For instance, consider a graph with vertices containing the attributes (*Gender*, *Location*, *Profession*). A cuboid query for this graph can be $(Gender, *, *)$ which will result in an aggregate graph showing all the vertices with the same Gender value aggregated and the edges also aggregate by the function COUNT.

Crossboid Query Return the aggregate network between two or more cuboids structures. An example of a crossboid query can be the aggregate network between an user with $ID = 3$ and all the locations.

Given that the size of the graph cube lattice is exponential with respect to the number of dimensions of the original multidimensional network, the paper proposes a partial materialisation in order to process queries. The partial materialisation is implemented using a greedy algorithm that selects k cuboids $(k < 2^n)$ to be materialised according to the benefit of those cuboids to improve the cost for query evaluation.

The graph cube implementation described above is also studied by other works. In [**?**], a distributive approach is proposed using map reduce jobs to calculate each step of the aggregation process. In [**?**], a new algorithm to compute the Graph Cube (iGraphCubing) is proposed, using a new prunning method based on the Structural Significance measure.This measure takes into account three factors:

- The diversity of the attribute value in the neighborhood for each vertex
- The clustering coefficient
- The density around each vertex

The work in [**?**] presents experiments evaluating the effectiveness and the efficiency of the method proposed. The effectiveness was evaluated by analysing OLAP queries submitted to the graph cube. The efficiency was evaluated by analysing the graph cube performance depending if it was fully or partially materialised in disk. In conclusion, the paper proposes an implementation of a graph cube obtained from an homogeneous attributed graph, but it does not consider heterogeneous networks neither attributed edges.

3.2 Graph OLAP: Towards Online Analytical Processing on Graphs

The Graph OLAP framework proposed in [**?**] takes as input a set of network snapshots, where each snapshot contains a graph and a set of informational attributes that describes the snapshot. In this scenario, the paper defines two dimension types: informational dimensions (formed by the informational attributes) and topological dimensions (formed by the attributes of the vertices and edges of the graph). The authors distinguish different semantics for OLAP operations in graphs, so these operations are categorised into Informational OLAP (I-OLAP) and Topological OLAP (T-OLAP).

The framework gives as output an aggregate graph with a summarised view of the snapshots set. The type of the aggregate graph returned by the framework can also be cat-

egorised in Informational Aggregate Graph (I-aggregate graph) and Topological Aggregate Graph (T-aggregate graph).

Informational Aggregate Graph is computed based on a set of network snapshots that have informational dimensions with same values. Figure **??** shows an example of an i-aggregate graph composed by snapshots describing the co-author relationship between individual authors. The co-author relationship is grouped by a certain conference (*sigmod*, *vldb*, *icde*), by a class of conferences (*db-conf*), by a specific period of time (*2004*, *2005*) and by a group of time periods (*all-years*). It is important to notice that the graph in Figure **??** shows different levels of aggregations for each co-author relationship.

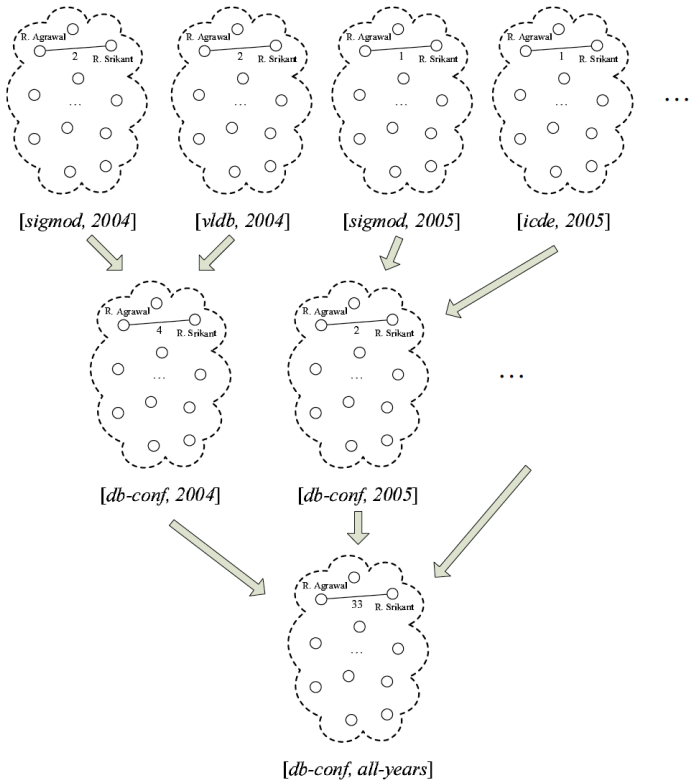


Figure 3.4: Example of Informational Aggregate Graph [**?**]

Classic OLAP operations in an i-aggregate graph can be interpreted as follows:

Roll-up Aggregate multiple snapshots to form a higher level summary of information

Drill-down Return to lower-level snapshots from aggregate graph

Slice / dice Select a subset of snapshots based on informational dimensions

Topological Aggregate Graph is obtained based on a single network, where the vertices are the result of applying the aggregate function to the vertices of the original network with the same attribute value. Figure **??** shows an example of a t-aggregate graph where the information about co-author relationship between individual authors in one snapshot was aggregated into co-author relationship between the institutions the authors belong to.

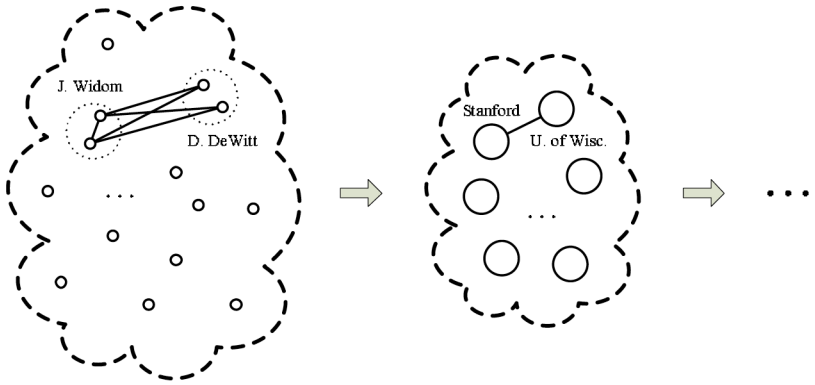


Figure 3.5: Example of Topological Aggregate Graph [**?**]

Classic OLAP operations in an t-aggregate graph can be interpreted as follows:

Roll-up Merge topological elements (vertices or edges) and replace them by corresponding higher-level elements

Drill-down Split merged elements into lower-level elements

Slice / dice Select a subgraph of a snapshot based on topological dimensions

The Topological OLAP is further explained in [**?**]. This work takes into consideration two properties (T-Distributiveness and T-Monotonicity) used to classify how different measures can be performed in an OLAP Graph. A measure function is considered T-Distributive if the result of the function applied to high-level vertices from the graph can be obtained by the computation of pre-computed results of the same function applied to lower-level vertices from the same graph. On the other hand, a measure is considered T-Monotone, if the data search space

can be pruned given a user-defined threshold, by dropping vertices pairs with measures that do not satisfy the threshold. The paper shows experiments using common constraint function, proven to be T-Distributives and/or T-Monotones, such as SUM, MIN, MAX, Density, Degree Centrality, Closeness Centrality, among others.

3.3 HMGraph

An Heterogeneous and Multidimensional Graph OLAP framework (HMGraph OLAP) is proposed in [**?**]. This framework uses a graph model similar to Graph OLAP [**?**], but it adds the concept of Entity Dimensions due to the heterogeneity of the input graphs (Graph OLAP framework only handles homogeneous graphs). Figure **??** shows an example of a heterogeneous multidimensional network, highlighting the entity attributes of the graph.

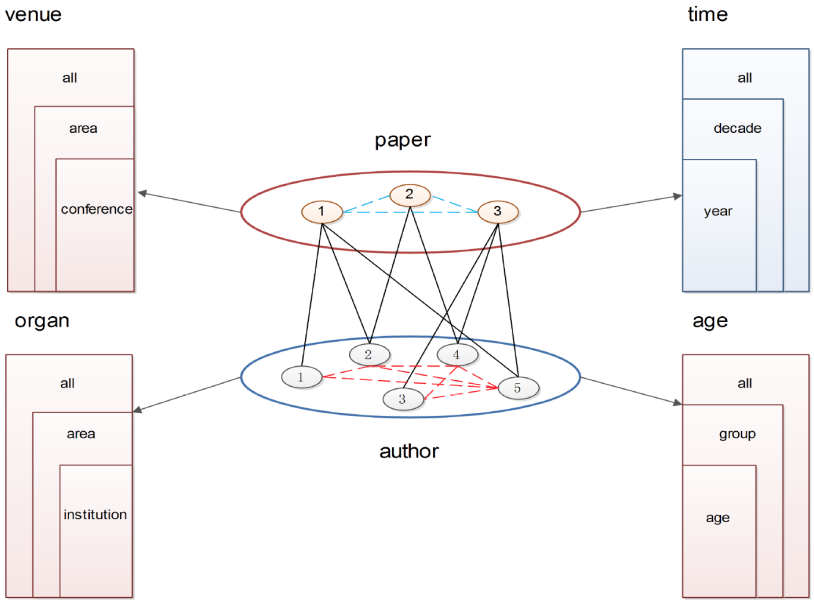


Figure 3.6: Example of a heterogeneous multidimensional network [**?**]

Entity attributes are the attributes that describe the characteristics of an entity. In the graph illustrated in Figure **??**, organ and age are entity attributes of author entity. Entity Dimension is related to the types of vertices in the graph.

Like Graph OLAP, HMGraph can perform I-OLAP and T-OLAP operations, but it can also perform rotate and stretch operations. The rotate operation is done by changing ver-

tices into edges and edges into vertices, as shown in Figure **??**. The stretch operation is done by changing edges into entities and adding edges between the recently created entity and the vertices previously connected to the transformed edge, as shown in Figure **??**.

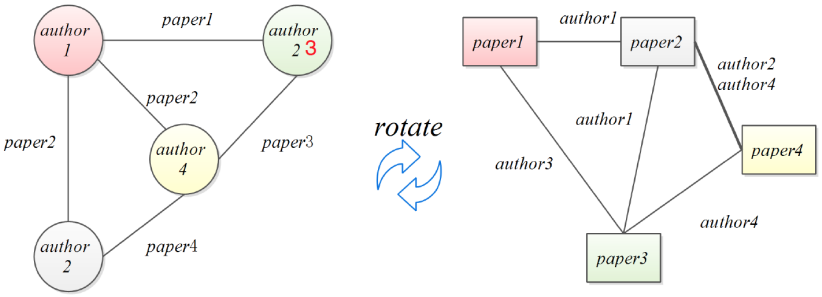


Figure 3.7: Example of rotate operation [**?**]

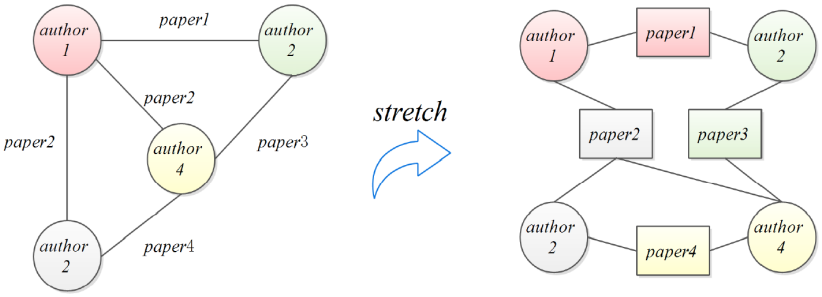


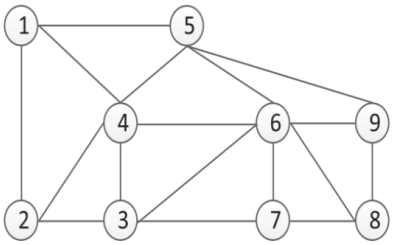
Figure 3.8: Example of stretch operation [**?**]

Even though the work presented in [**?**] draws attention to the importance of heterogeneous networks in real world application, the paper does not provide further implementation detail on how the framework can be used with real data.

3.4 Pagrol: PArallel GRaph OLap over Large-scale Attributed Graphs

The work presented in [**?**] proposes a parallel Graph OLAP system, adopting the Hyper Graph Cube model that extends attributed graphs to support decision making services. The model proposed in this paper is similar to the Graph Cube described in [**?**], with the main difference

being the presence of attributes also in the graph edges. In this scenario, there are two types of dimensions: vertex dimensions and edge dimensions. Figure **??** shows an example of an attributed graph.



(a) Graph Structure

ID	Gender	Nation	Profession
1	male	USA	professor
2	female	USA	doctor
3	male	China	engineer
4	male	SG	engineer
5	female	SG	professor
6	male	SG	doctor
7	female	China	engineer
8	female	China	doctor
9	male	USA	doctor

(b) Vertex Attribute Table

sV	tV	Date	Type	Strength
1	2	2008	Family	9
1	4	2010	Friend	7
1	5	2011	Colleague	9
2	3	2011	Friend	7
2	4	2011	Friend	4
3	4	2008	Friend	8
3	7	2011	Colleague	8
3	6	2012	Family	3
4	5	2009	Family	9
4	6	2010	Friend	6
5	6	2010	Family	8
5	9	2012	Friend	8
6	7	2008	Friend	7
6	8	2012	Colleague	9
6	9	2012	Friend	5
7	8	2011	Friend	5
8	9	2012	Colleague	8

(c) Edge Attribute Table

Figure 3.9: Example of attributed graph [**?**]

Given an attributed graph with n vertex dimensions and m edge dimensions, the Hyper Graph Cube will contain 2^{n+m} aggregate graphs obtained as described by the work of [**?**]. This Hyper Graph Cube can be seen as the cartesian product between all the vertex-aggregate networks (when one or more vertex dimensions are aggregated) and the edge-aggregate networks (when one or more edge dimensions are aggregated). This cube arrangement can support the following categories of queries:

Category 1 Queries answered by information stored either in a vertex or in an edge attributes. For example:"How many relationships appeared in 2012?" or "What is the percentage of users in each different profession in this network?"

Category 2 Queries answered by integrating the knowledge stored at both vertex and edges attributes. For example: "What is the trend of the number of relations appearing between USA and SG (Singapore) in the last 3 years?"

Category 3 Queries answered by an aggregate graph, that provides a summarised view of the data along some dimensions. For example: "What is the graph structure as grouped by users' gender as well as relationship type?"

The Hyper Graph Cube also supports roll-up and drill-down operations, along both vertex and edge dimensions. For instance, if we have an aggregate graph along Location dimension according to City value, we can roll up to obtain an aggregate graph according to State value.

The materialisation for the Hyper Graph Cube is done using Map-Reduce(MR) jobs. Since vertex and edges are stored in two different tables in the distributed file system (DFS), the materialisation is done in two steps: first the two tables are joined into one flat table containing all the dimensions and the second step performs the cube computation. This process is optimised using self-contained joins and batching techniques.

3.5 GRAD Graph Cubes

One of the most recent works in this area is presented in [**?**]. This paper proposes a new technique for extracting multidimensional concepts and building OLAP cubes from heterogeneous property graphs. The authors propose a new classification of graph measures based on the aggregation type and computation algorithm:

Content-based measure Calculated based on graph's vertices and edges attributes. They are similar to traditional OLAP measures.

Graph-based measure Obtained by applying graph algorithms. They capture topological properties of the graph.

Graph as measure Different aggregation levels of a graph can be considered measures.

Given a property graph with two distinct classes of vertices, the authors explore candidate dimensions, measures and cubes that can be obtained from the graph. The example used throughout the paper is a movie graph: it has movie vertices linked to vertices representing the

actors that acted in the movie, as shown in Figure **??**. The dimensions obtained by a subset of vertices and edges attributes are called inter-class dimensions.

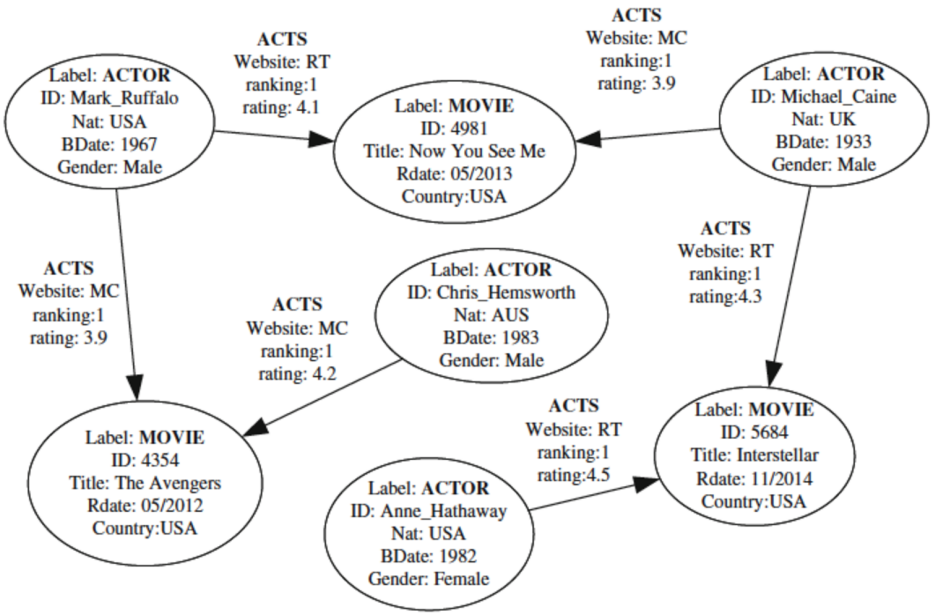


Figure 3.10: Original movie graph [**?**]

Once the dimensions are selected, a graph lattice is defined by all possible OLAP aggregations obtained by aggregating the intra-class dimensions. The inter-class measures fall back in one of the aforementioned categories (content-based, graph-specific or graph as measure).

Figure **??** shows the aggregate graph obtained by grouping movies by their release date and actors by their birth date and gender. The graph shows the average ranking and rating of the ACTS relationship between grouped actors and movies.

This paper also proposes a technique for building OLAP cubes extracted from a graph modelled according to the analysis-oriented graph database model GRAD. This model provides advanced graph structures, integrity constraints and graph algebra. According to the authors, traditional property graphs only support OLAP analysis of inter-class dimensions, while additional capabilities brought by GRAD allows the analysis of information stored within each vertex.

The GRAD model consider heterogeneous, attributed, labelled graphs and supports

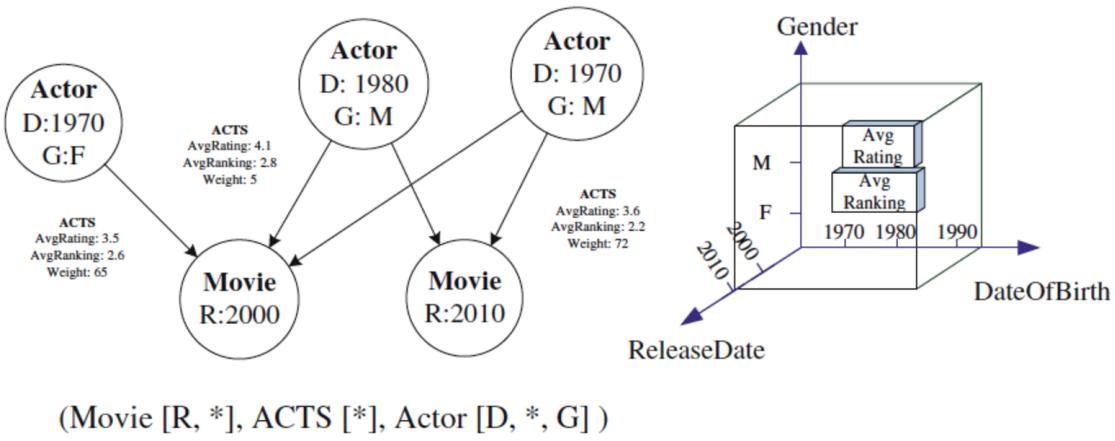


Figure 3.11: Aggregate Graph for inter-class dimensions [**?**]

complex type attributes on the vertices. This model introduces special analytical structures called hypernodes, that represent real world entities grouped by classes. Each hypernode is a subgraph formed by an entity vertex - which contains the label and the identifier attributes - attributes vertices - linked to the entity vertex and represent the non-identifier attribute - and literal vertices - which stores the effective value of its attribute vertex. Figure **??** shows an example of a movie graph modelled with GRAD.

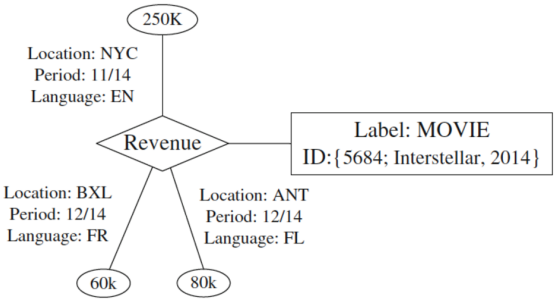


Figure 3.12: Movie graph on GRAD model [**?**]

Based on this model, the paper defines Intra-class Dimensions as a subset of attributes vertices. The Intra-class Measures are identified by the attribute vertex label and are calculated in a similar way as the measures in property graph model. Figure **??** shows the result of applying aggregation function to the original GRAD graph in order to calculate the revenue measure, aggregating the Location according to the Country Name (CN) attribute.

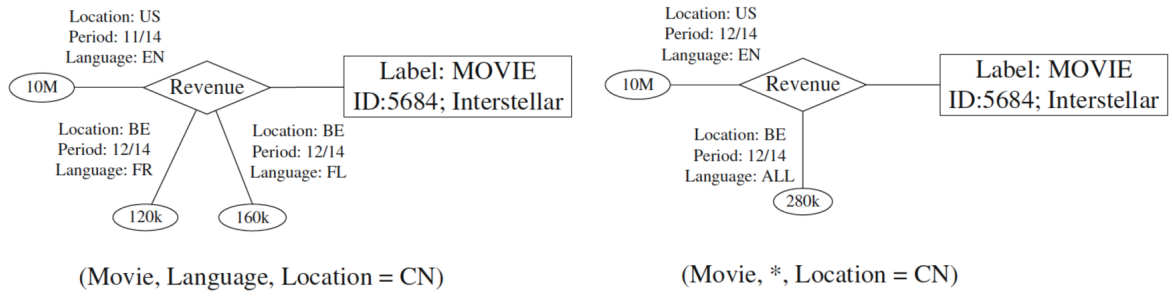


Figure 3.13: Aggregate Graph for intra-class dimensionl [**?**]

This framework's implementation used Neo4J for the graph storage and HDFS (Hadoop Distributed File System) for distributed processing. The architecture is also composed by a middleware layer that is responsible for computing the aggregate graph and measures, using GraphX[1] library to calculate graph-specific measures.

3.6 Comparative Analysis

The first works done in OLAP analysis on graph focused on homogeneous graph datasets and defined ground concept of this area, such as aggregate graphs and graph lattice. Several operators were proposed and, in general, three types of measures were taken into consideration:

Informational / Content-based / Attribute-based measure Similar to traditional OLAP measures, this information is obtained by applying an aggregate function on the vertex's attributes.

Topological / Aggregate Graph measure This type of measure gives information about the topology of the graph and is obtained by applying aggregate function on vertices and edges, generating a graph as a measure.

Graph-based / specific measure This type of measure is based on graph analysis theory and can be represented by a number or a subgraph.

[1] https://spark.apache.org/graphx/

One relevant characteristic of the works presented so far is the little explanation given on how the framework was indeed implemented, which made their understanding rather difficult. The Table **??** shows a comparison between all the frameworks presented in this chapter, regarding the type of graph, dimensions and operations supported by each one.

Table 3.1: Comparison of studied frameworks

Framework	Graph	Dimensions	Operations
Graph Cube	Homogeneous	Vertex Attributes	Cuboid and Crossboid Query
Graph OLAP	Homogeneous	Informational and Topological	I-OLAP and T-OLAP Operations
HMGraph	Heterogeneous	Informational, Topological and Entity	I-OLAP, T-OLAP, Rotate and Stretch Operations
Pagrol	Homogeneous	Vertex and Edge Attributes	3 Query Category and Roll-up/Drill-down Operations
GRAD Graph Cubes	Heterogeneous	Inter-class and Intra-class	-

The work proposed here will be focused in heterogeneous graph datasets and will support the three types of measures described by the work on GRAD Graph Cubes, since those measures represent a compilation of all the other measures proposed by other authors. In addition to that, this work will also explore OLAP operations and network analysis on graph databases without the need to define a new graph model, as suggested by previous works, eliminating the extra step of parsing operational data to the new model.

3.7 Final Considerations

In this chapter, we presented the main research works published related to OLAP system using Graph Databases. The majority of the works available only supported homogeneous graph, but they introduced important concepts of the area, such as graph cubes and aggregate graph. The implementations that actually gave support to heterogeneous graph, proposed different graph

models in order to answer analytical queries. In the next chapter, we will specify a simple OLAP system using Graph Database without the need to define a new graph model.

Chapter 4

OLAP Analysis on Graph Database

In this chapter we will propose a system capable of executing OLAP analysis and that supports heterogeneous graphs, without the need to define a new graph data model. Initially, we will contextualise the proposed system and introduce a running example that will be used to better explain the system operation. Then, the system's architecture is presented and its main components are further detailed in the following sections.

4.1 Contextualisation

In Chapter 3, we investigated some of the main works in the area of Graph OLAP. Most of them only give support to homogeneous graphs [**?**][**?**][**?**], while real world graph-like data contains different types of vertices and edges.The frameworks HMGraph [**?**] and GRAD Graph Cube [**?**] support heterogeneous graphs, but they propose a new multidimensional model in order to do OLAP analysis.

The objective of the system proposed here is to support OLAP analysis on heterogeneous graph databases without the need to re-model operational data. This will be done by adding a layer of pre-processed aggregate graphs and an analytical query processor module on top of the operational graph database.

4.2 Running Example

Consider the Database System and Logic Programming (DBLP) dataset as the running example that will be used throughout this chapter to help explaining the main concepts of the system proposed. The DBLP[1] is an online computer science bibliography that, up until May 2016, indexes more than 3.3 million publications by more than 1.7 million authors. For this example, we will consider that the data was extracted and modelled according to the schema shown in Figure **??** and described as follows:

- Each publication becomes a vertex with label *Publication* and with the attributes *title*, *year* and *venue*.
- Each author becomes a vertex with label *Author* and with the attribute *name*
- Edges labeled *PUBLISHED* connect Author vertices to Publication vertices, representing the relationship between an author and their published work.
- Edges labeled *COAUTHOR* connect Author vertices to other Author vertices, representing the relationship between authors that have contributed to the same published work.

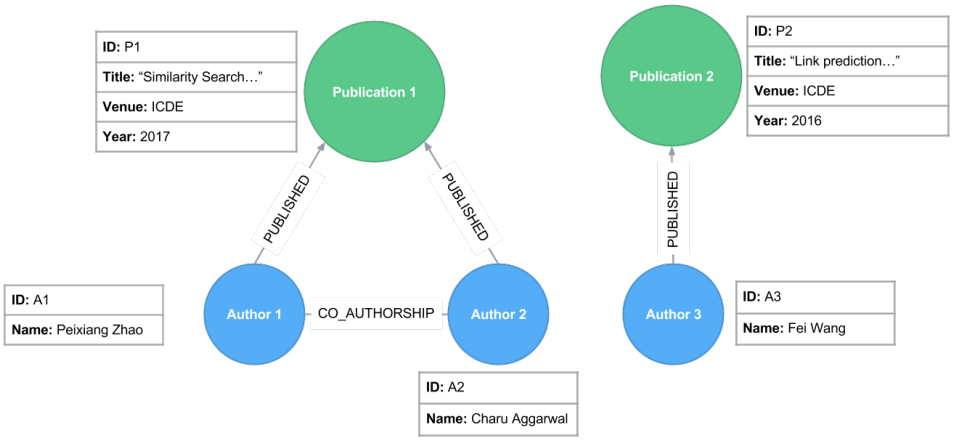
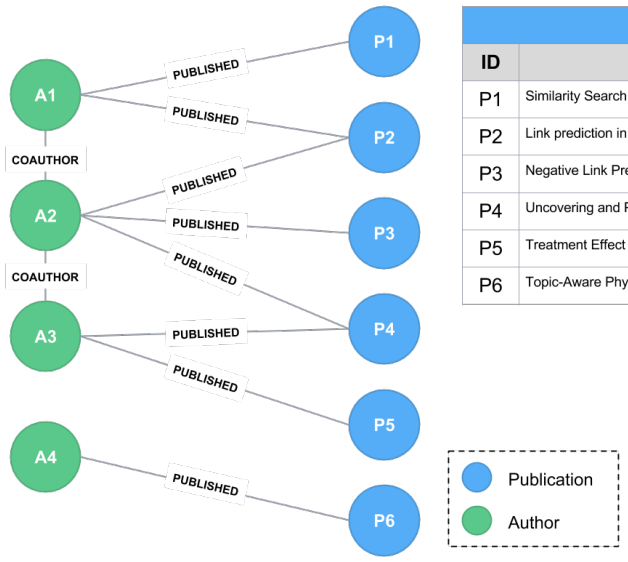


Figure 4.1: Schema representation of the DBLP data graph

[1] http://dblp.uni-trier.de/faq/What+is+dblp.html

Figure **??** shows a subset of the original DBLP dataset, modelled according to the schema representation depicted in Figure **??**. The following graph will be used as our running example throughout this chapter.



Publication			
ID	Title	Venue	Year
P1	Similarity Search in Graph Databases: A Multi-Layered Indexing...	ICDE	2017
P2	Link prediction in graph streams	ICDE	2016
P3	Negative Link Prediction in Social Media	WSDM	2015
P4	Uncovering and Predicting Human Behaviors	IEEE	2016
P5	Treatment Effect Estimation with Data-Driven Variable...	AAAI	2017
P6	Topic-Aware Physical Activity Propagation in a Health Social...	IEEE	2016

Author	
ID	Name
A1	Peixiang Zhao
A2	Charu Aggarwal
A3	Fei Wang
A4	NhatHai Phan

Figure 4.2: Running Example with subset of DBLP dataset

4.3 Dimensions and Measures

As discussed in Chapter 2, an OLAP system is a tool that facilitates multidimensional analysis of the data. In order to perform such kind of analysis, it is necessary to define the dimensions and measures that will be considered during the multidimensional analysis:

Dimension Given a labeled property graph $G = (V, E, L_V, L_E, A_V, A_E)$, where:

- V is a set of vertices
- $E \subseteq V \times V$ is a set of edges
- L_V is a set of vertices labels and L_E is a set of edge labels
- $A_V = \{a_1, \ldots, a_n\}$ is a set of vertex attributes, where $a_i = (k_i, m_i)$ is a key-value pair, k_i is the attribute key and m_i is the attribute value. Each vertex $v_i \in V$ is associated

with a set of attributes. $A_E = \{b_1, \ldots, b_n\}$ is the set of edge attributes defined in the same way as vertex attributes.

A dimension is given by $d = (a, l)$, where $a \in A_V$ and $l \in L_V$.

In our running example, we can define $d_1 = (journal, Publication)$ and $d_2 = (year, Publication)$, i.e. the *Publication* attributes *journal* and *year* are dimensions d_1 and d_2 in our OLAP system, respectively.

Measure Given a labeled property graph $G = (V, E, L_V, L_E, A_V, A_E)$, a measure m is a calculation computed over the graph G using a function F ($m = F(G)$), that can return the type of measures defined by [**?**]:

Content-based measure For this kind of measure is calculated based on the vertices and edges attributes and the function $F \in \{SUM, COUNT, AVG, orotheraggregatefunctions\}$ used in the calculation are similar to the ones used in an OLAP system.

In our running example, the total number of authors that published a work in 2007 is a content-based measure that is calculated using the function $COUNT$, which will count the number of *Author* vertices that have a relationship *PUBLISHED* to *Publication* vertices that have attribute *year* equals to 2007.

Graph-based measure This type of measure is calculated by applying a network analysis algorithm over the graph, i.e. a network analysis algorithm is used as the function F.

In our running example, the list of authors that most contributed with other authors is a graph-based measure that is computed by applying the degree centrality measure to the *Author* vertices of the graph.

Graph as measure This kind of measure is given by different aggregation levels of a graph and the function F that calculates this measure is the aggregate function that will generate the aggregate graph.

In our running example, the network of authors and publications aggregated according to the venue in which the work was published in is a graph that represents a measure.

4.4 Architecture

The Graph OLAP system proposed in this work attempts to provide an efficient way to answer analytical queries without having to propose a new graph data model, which would imply changing the original data source model. The Figure **??** depicts the architecture of the system, illustrating its main components: Graph Aggregators, Aggregated Graphs and Analytical Query Processor.

The Graph Aggregators are modules that are responsible for processing the original data and generate Aggregate Graphs, which are stored in Aggregate Graph Databases. The Analytical Query Processor is in charge of processing the incoming query and the user will determine whether it should be answered by processing the original or the aggregate data, based on the type of measure being analysed.

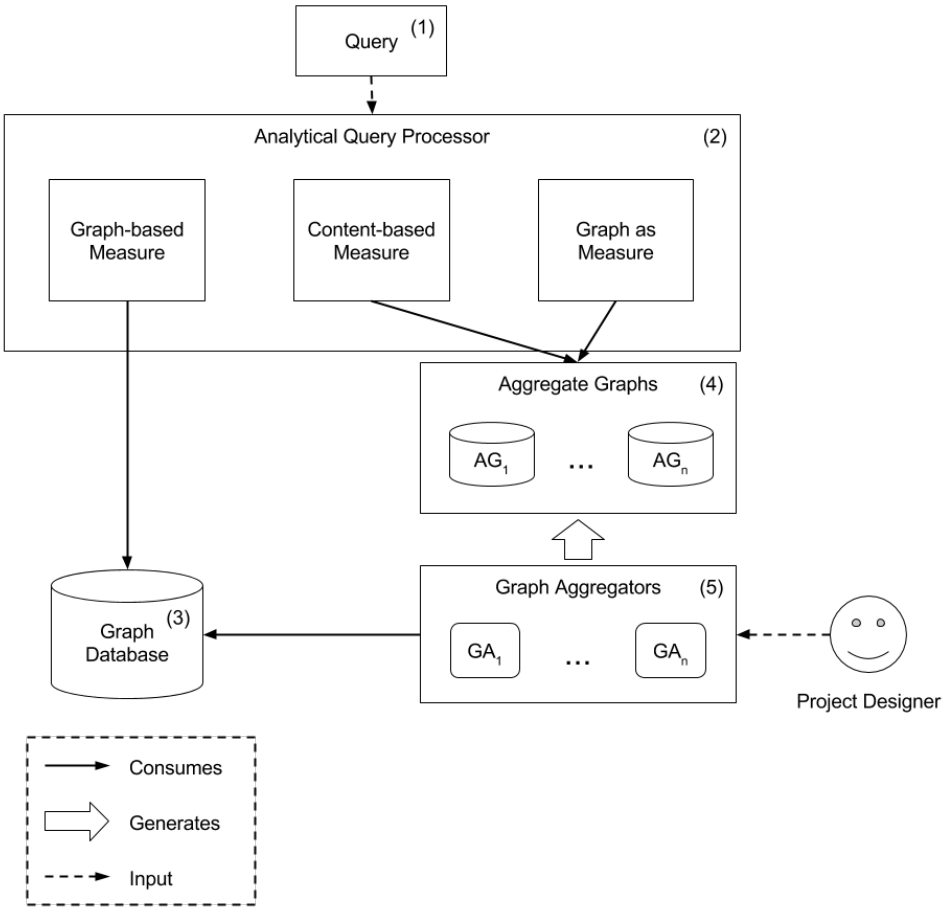


Figure 4.3: OLAP Analysis over Graph Databases Architecture

The system's input is an analytical query submitted by the user (1), that will be processed by the Analytical Query Processor (AQP) (2). According to the type of measure required by the user, the AQP will determine which specific processor will handle the query: graph-based, content-based or graph measures processors. If the user asks for a graph-based measure, the AQP will consume the original data stored in the graph database (3) to calculate the measure. On the other hand, if the user requires a content-based or a graph as measure, the AQP will calculate the measure based on the data from Aggregate Graphs (4), which are also stored in graph databases.

The Aggregate Graphs are generated by the Graph Aggregators (GAs) (5), which are defined during the design process of the system by the project designer. The project designer is responsible to define what are the dimensions considered in the OLAP system and, therefore, create the GAs that will generate all possible aggregate graphs for the dimensions. More details on Aggregate Graphs and Graph Aggregators are given in the following sections.

The data source considered for this system is a Graph Database that follows the labeled property graph model and supports heterogeneous graphs. This means that vertices can have one or more labels indicating different types of entities. Vertices and edges can have properties. We assume that the data stored in the GDB is integrated, i.e. the data in the repository is consistent, well-formatted and normalised.

4.5 Aggregate Graph

Given a graph $G = (V, E, L_V, L_E, A_V, A_E)$ and a set of dimensions $D = \{d_1, \ldots, d_n\}$, where $d_i \in A_V \cup A_E$, an aggregate graph is generated by applying an aggregate function F to one or more dimensions. The result is a new graph $G_A = (V_A, E_A, L_{VA}, L_{EA}, A_{VA}, A_{EA})$, where:

- $VA = \{v_1^A, \ldots, v_n^A\}$ is a set of aggregate vertices, where each vertex v_i^A either (i) corresponds to the result of applying the function F to a set of vertices $V' \subseteq V$ that is associated with a set of attributes $\{a_1, \ldots, a_k\}$ containing one or more dimensions in D or (ii) corresponds to a vertex in V.

- $E_A \subseteq V_A \times V_A$ is a set of aggregate edges, where each edge e_i^A either (i) corresponds to

the result of combining a set of edges $E' \subseteq E$ that connects one or more vertices in V that were aggregated in V_A or (ii) corresponds to an edge in E.

- L_{VA} is a set of aggregate vertex labels and L_{EA} is a set of aggregate edge labels
- $A_{VA} = \{a_1, \ldots, a_n\}$ is the set of attributes for the aggregate vertices, where $a_i = (k_i, m_i)$ is a key-value pair, k_i is the attribute key and m_i is the attribute value. Each aggregate vertex $v_i \in V_A$ is associated with an attribute vector. $A_{EA} = \{b_1, \ldots, b_n\}$ is the set of aggregate edge attributes defined in the same way as aggregate vertices attributes.

Consider the graph depicted in Figure **??** of our running example. Figure **??** shows the aggregate graph G_A obtained by applying the aggregate function COUNT to the dimension set $D = \{d\}$, where $d = (year, Publication)$. Notice that the resulting aggregate graph ends up with the same *Author* vertices as the original graph, since these vertices do not have attributes contained in the dimension set D. The *Publication* vertices were aggregated according to their *year* attribute, resulting in three vertices representing the works published in 2017, 2016 and 2015. The aggregate vertices also store the measure calculated using the function COUNT. The edges with the label PUBLISHED were combined, representing the connection between each author and the group of works published in a specific year. The combined edges also store the measure obtained by the use of COUNT function.

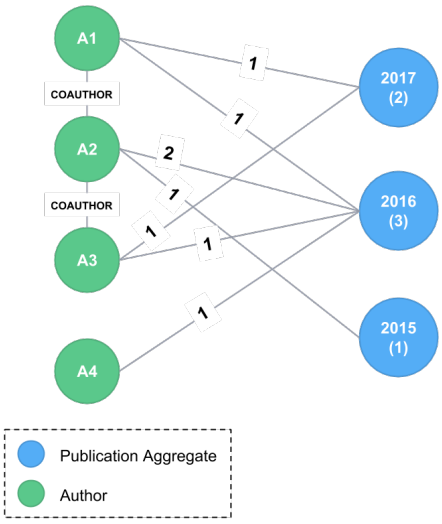


Figure 4.4: Aggregate Graph obtained from running example graph

4.6 Graph Aggregators

The Graph Aggregators (GAs) are modules responsible for generating the aggregate graph that will be used to answer the analytical query submitted by the user. During the design process of the system, the Project Designer is responsible for building the GAs based on the dimensions and measures the system should be able to analyse. Each GA will receive as input the original graph G stored in the Graph Database, the set of dimensions to be aggregated D, the aggregate function F and should provide as output an aggregate graph as defined in the previous section. Algorithm **??** describes the process performed by a GA in order to generate an aggregate graph.

Algorithm 1 Graph Aggregator Process

```
1:  function generateAggGraph(G, F, D)
2:      dimValue                                   ▷set of values for dimensions being aggregated
3:      aggVertices        ▷map from dimensions values to set of vertices with corresponding values
4:      aggEdges           ▷map from dimensions values to set of edges with corresponding values
5:      nonAggVertices                             ▷set of vertices that were not aggregated
6:      originalVertices ← G.vertices              ▷set of vertices from the original graph
7:      for each vertex in originalVertices do
8:          if vertex in D then
9:              dimValue ← getDimValue(vertex, D)
10:             if dimValue in aggVertices then
11:                 aggregateVertices(dimValue, vertex, F)
12:                 aggregateEdges(dimValue, vertex, F)
13:             else
14:                 aggVertices.add(dimValue, vertex)
15:                 aggregateEdges(dimValue, vertex, F)
16:             end if
17:         else
18:             nonAggVertices(vertex)    ▷vertex does not contain the dimensions being aggregated
19:         end if
20:     end for
21:     aggGraph ← (aggVertices ∪ nonAggVertices, aggEdges)
22:     return aggGraph
23: end function
```

The GA algorithm starts by creating an empty set of aggregate vertices and edges (lines **??** and **??**) that will compose the final aggregate graph that will be returned. Then, the GA will iterate over all the vertices (nodes) of the original graph G (line **??**), checking for each node if it has the same label as the dimension being aggregated (line **??**). If the node has the same label, the algorithm checks if already exists an aggregate node for the dimension value

of the node (line **??**). If the aggregate node exists, we collapse the value of the node to the value of the aggregate node using the function F and updating the aggregate node value (line **??**). We also aggregate the edges that are connected to the node accordingly (line **??**). If the aggregate node does not exist, we add a new aggregate node with the initial value equals to node's dimension value and aggregate its edges accordingly (lines **??** and **??**). If the node does not have the same label as the dimension being aggregated, we only add the node as it is to the non aggregate vertices set (line **??**). Finally, we setup the aggregate graph and return it (lines **??** and **??**).

Once the aggregate graph is generated, it will be stored in a graph database and it will be accessed by the Analytical Query Processor.

4.7 Analytical Query Processor

The Analytical Query Processor (AQP) is responsible for processing the query submitted by the user. The user is responsible for submitting the correct query based on the information he/she is trying to retrieve, i.e. the query should be written according to the type of measure being requested:

Content-based measure To calculate this measure, the AQP consumes the data from an aggregate graph and uses the aggregate function to give the resulting measure, which can be a single value, a list or a table of values. This module's response is similar to the response given by traditional OLAP systems.

From our running example, we can ask for the number of publications by year. In this case AQP consumes the data from the aggregate graph illustrated in Figure **??** and it would list the nodes with *Publication Aggregate* label, which already contains the count measure as attribute.

Graph-based measure To calculate this measure, the AQP consumes the data from the original graph database and executes network analysis algorithms on the data.

In our running example, we could submit a query asking for the *Author* node with highest

centrality degree. The AQP calculates the centrality degree for each node in the original graph using an external library. Then, it should return the node with id **A2**, since it is the Author node with the highest number of connections with other nodes.

Graph as measure For this type of measure, the AQP also consumes data from an aggregate graph, but in this case, the measure is the aggregated graph itself. Therefore, this module does not need to perform other calculations.

For instance, the aggregate graph shown in Figure **??** can be considered a measure if the user requests a topological view of the original data grouping the publications by year.

4.8 Final Considerations

In this chapter, we showed the main components of the proposed system and what is the general operation to answer an analytical query. We also specified in details how each one of the components works and what are their roles in the data analysis process. In the next chapter, we will report how the proposed system was implemented and show some experiments and the results obtained.

Chapter 5

Implementation and Experiments

In this chapter, we will show how the system specified in the Chapter 4 was implemented and what were the technologies used. We will also describe the experiments made and analyse the results obtained in comparison to existing solutions. Finally, we will discuss the difficulties found in the implementation and experimentation process.

5.1 Used Technologies

The Graph Aggregator (GA) algorithm was implemented using Python programming language, in version 2.7. The original data and the aggregate graphs were stored in a Neo4J database. In order to connect the GA to the Neo4J database, it was necessary to use the Python library Py2Neo, in version 3.1.2. The Analytical Query Processor uses the compiler built in Neo4J and the query accepted as input to the system is written using Cypher query language.

5.2 Dataset

The dataset used for the experiments was the Database System and Logic Programming (DBLP) computer science bibliography, available in (http://dblp.uni-trier.de/db/), which contains more than 3.8 million publications published by more than 1.7 million authors. The dataset can be downloaded as a XML file accompanied by a DTD file that describes the schema of the data.

The Listing in **??** is an excerpt of the DBLP XML file showing how a publication is structured in the document.

```
<article mdate="2017-01-03" key="journals/jacm/GanorKR16">
  <author>Anat Ganor</author>
  <author>Gillat Kol</author>
  <author>Ran Raz</author>
  <title>Exponential Separation of Information and
    Communication for Boolean Functions.</title>
  <pages>46:1-46:31</pages>
  <year>2016</year>
  <volume>63</volume>
  <journal>J. ACM</journal>
  <number>5</number>
  <ee>http://dl.acm.org/citation.cfm?id=2907939</ee>
  <url>db/journals/jacm/jacm63.html#GanorKR16</url>
</article>
```

Listing 5.1: DBLP XML File Excerpt

The excerpt shown in Listing **??** refers to an article published in J. ACM journal and it contains information about the article's authors, title, pages in journal, year of publication and other informations about the journal's volume, number and electronic edition location. Each publication also has a unique key and the date of the last modification as attributes and an element with the url of the publication in the DBLP website. Besides journal articles, the DBLP dataset also contains books, thesis, conferences and workshop papers, among others.

Once the XML file was downloaded, all the data was parsed and stored into a SQLite database. In order to have a more controlled environment for the experiments and allow an eventual manual check of the results obtained from hte experiments, we selected a subset of the original data, considering only papers and articles published in the following venues since 2014:

- SIGMOD International Conference on Management of Data (SIGMOD)
- Brazilian Symposium on Databases (SBBD)
- International Conference on Very Large Databases (VLDB)
- IEEE International Conference on Data Engineering (ICDE)
- World Wide Web: Internet and Web Information Systems (WWW)

The selected subset of publications was imported to a Neo4J instance, following the schema depicted in Figure **??**. Each publication became a node in the graph database, with three attributes: (i) the title; (ii) the year of publication and (iii) the acronym of the venue. The authors of each publication also became a node, uniquely identified by the name of the author, and they are connected with the publications node by a relationship of type PUBLISHED. Authors that have contributed in the same publication are also connected by a relationship of type CO_AUTHORSHIP.

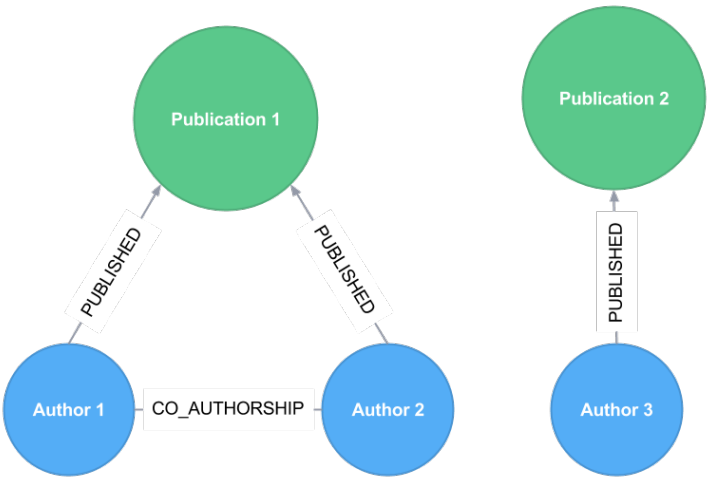


Figure 5.1: DBLP dataset schema in graph database

By the end of the DBLP subset loading process to Neo4J, we had 887 Publication nodes, 2.398 Author nodes, 6.754 PUBLISHED relationships and 25.572 CO_AUTHORSHIP relationships.

5.3 Experiments and Results

With the Neo4J database loaded with DBLP data, we executed the Graph Aggregator (GA) algorithm passing as parameters the dimensions year and venue of a Publication node and the COUNT aggregate function. Figure **??** shows a subgraph of the aggregate graph generated by the GA, with one aggregate node representing all the publications on ICDE 2016 and some of the authors that published on that conference, that year. From the measure attribute of the aggregate node, we know that ICDE had 60 publications in 2016.

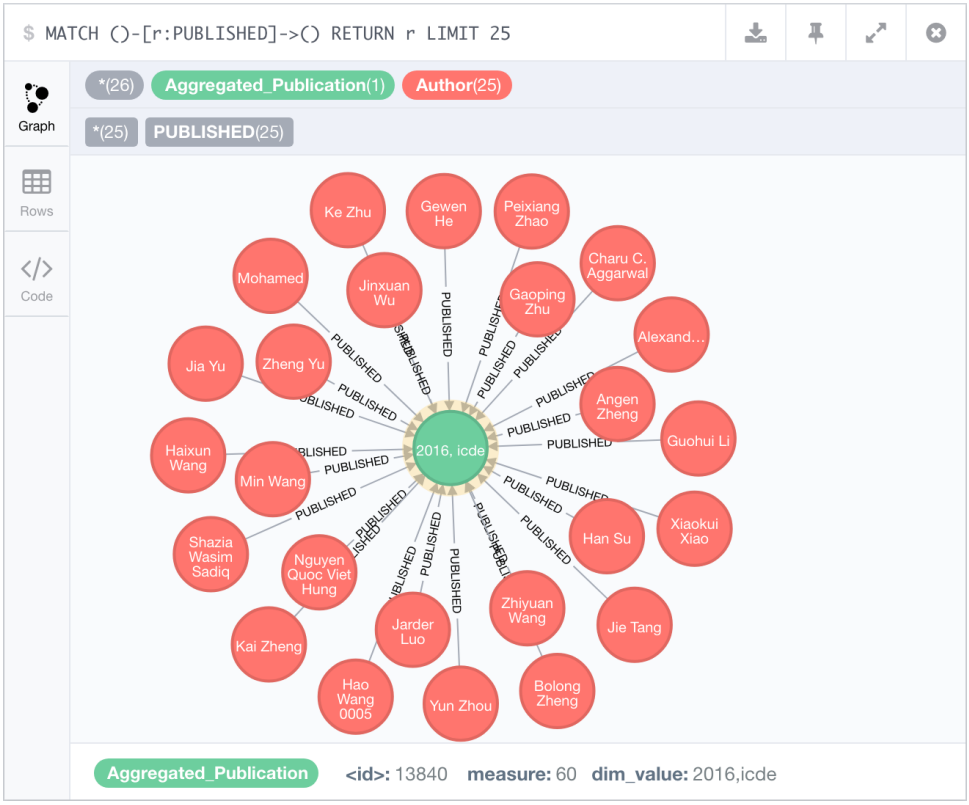


Figure 5.2: Subgraph from the Aggregate Graph generated by the GA

Once we generated the aggregate graph, we were able to do an experiment to evaluate the effectiveness of the proposed system. For that, we submitted queries to calculate the three types of supported measures: content-based, graph-specific and graph as measure.

5.3.1 Content-based Measure

For this type of measure, we submitted a query asking for the amount of publications by venue in the year of 2016. This query was submitted to the aggregate graph generated by the GA and it returns all the nodes with the label Aggregate_Publication where the dimension year has the value 2016. The result is then ordered by the number of publications measure.

Figure **??** depicts the result of the query, listing the venues ordered by the amount of publications in the year of 2016. The conference SIGMOD appears at the top of the list with 61 publications for that year. In comparison to a traditional relational OLAP system, this query is corresponding to a slice operation, in which we slice a part of the venue dimension based on the value of the dimension year.

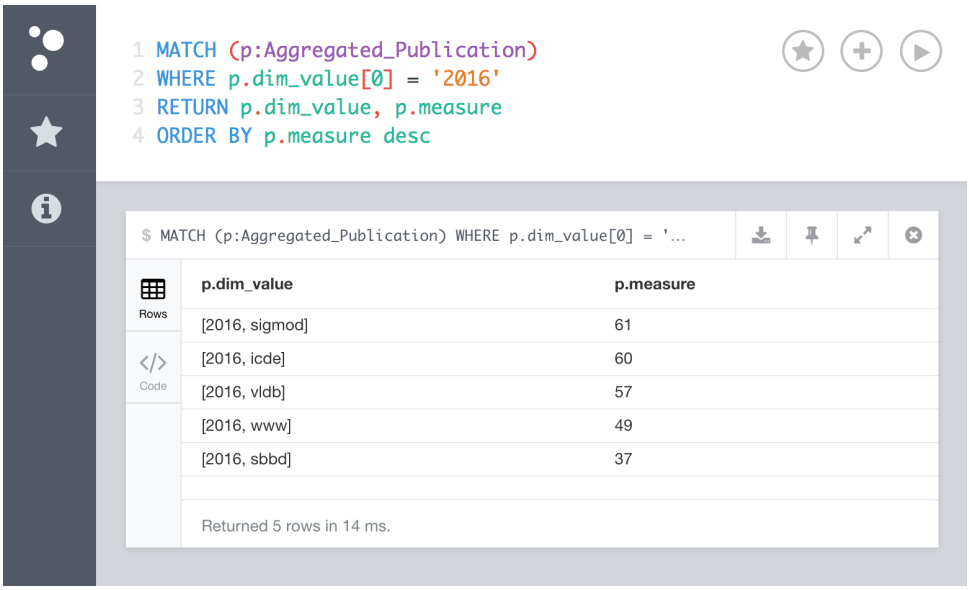


Figure 5.3: Result of experiment with a content-based measure query

The other OLAP operation tested for this type of measure was a roll-up operation. For this experiment, we had to include a random month of release for each publication in order to establish a hierarchy between the attributes of a publication, i.e. month and year. Once the original data was chaged and the fictional month of release of each publication was added, we generated an aggregate graph for the dimensions year and month using the COUNT aggregate function. The result of the aggregate graph is show in Figure **??**, where we can see the amount of publications per month per year.

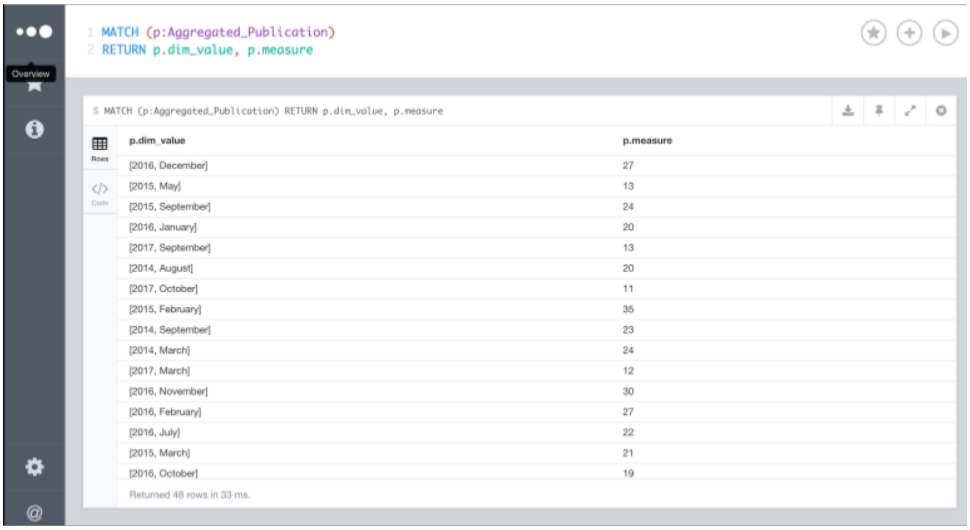


Figure 5.4: Result of experiment with a query for the dimensions year and month of publication

In order to execute a roll-up operation on the data and have an overview of the data only considering the dimension year, we generated another aggregate graph for that dimension using the COUNT aggregate function. The result of the second aggregate graph is show in Figure **??**, where we can see the amount of publications per year.

Figure 5.5: Result of performing a roll-up operation on the content-based measure

5.3.2 Graph-specific Measure

In order to test this type of measure, we submitted two centrality measures queries to the original dataset in the graph database. The first measure calculated was the degree centrality of each Author node considering the CO_AUTHORSHIP relationship. As mentioned in Chapter 2, the degree centrality is given by the number of adjacent node, i.e. the number of

CO_AUTHORSHIP relationship an Author node has.

Figure **??** shows the result of the centrality measure query, listing the 10 authors with the highest degree centrality in the original network. For our dataset, Michael J. Franklin is the author with the highest number of adjacent Author nodes, which means that he is one of the focal points in the network.

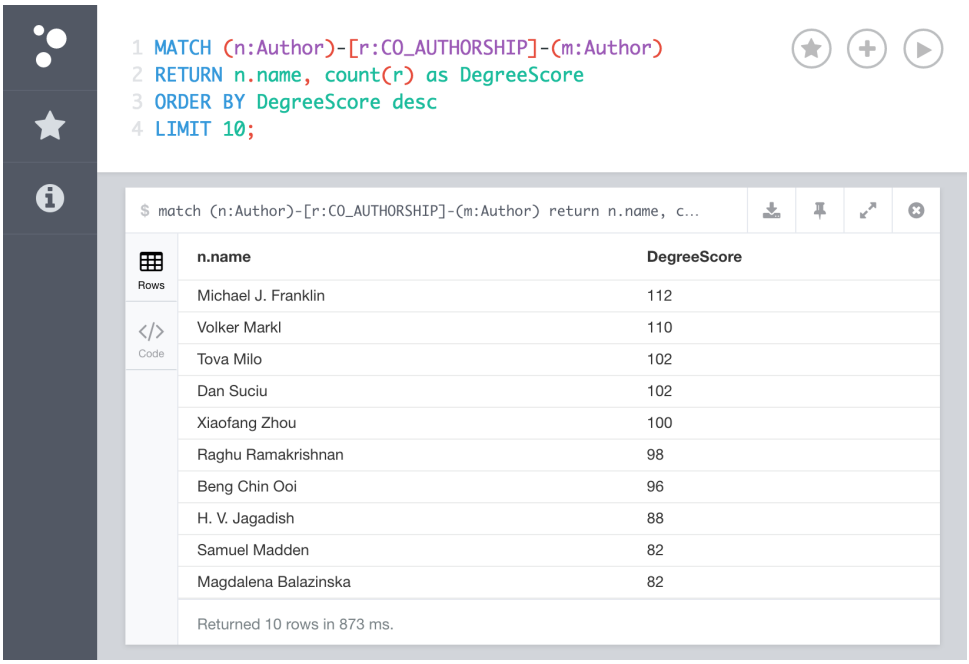


Figure 5.6: Result of experiment with the degree centrality measure for Authors

The second measure calculated was the betweenness centrality of the authors. This centrality measure is given by the frequency in which an author node appears in the shortest path of any other two author nodes in the graph. In Cypher, there is a built-in function to retrieve all shortest path between two nodes called *allShortestPaths*. Since getting all the shortest path between the combination of any 2 nodes in the graph is such a complex computation, we considered the paths with maximum 3 degrees of separation and we avoided inverse relationships by comparing the id of the nodes. Finally, we returned the 10 authors that most appeared in the list of the shortest paths.

Figure **??** shows the result of author's betweenness centrality calculation. Reaffirming his importance in the co-authorship network, Michael J. Franklin appears again in the first position of the list, meaning that he is an important point of collaboration between authors. One

interesting result is related to the second place for both experiments: the second author with the highest degree centrality is not the same as the second author with the highest betweenness centrality. This means that, even so Volker Marki co-authored publications with more authors than Beng Chin Ooi, the latter is part of more collaboration chains between two other authors in the network.

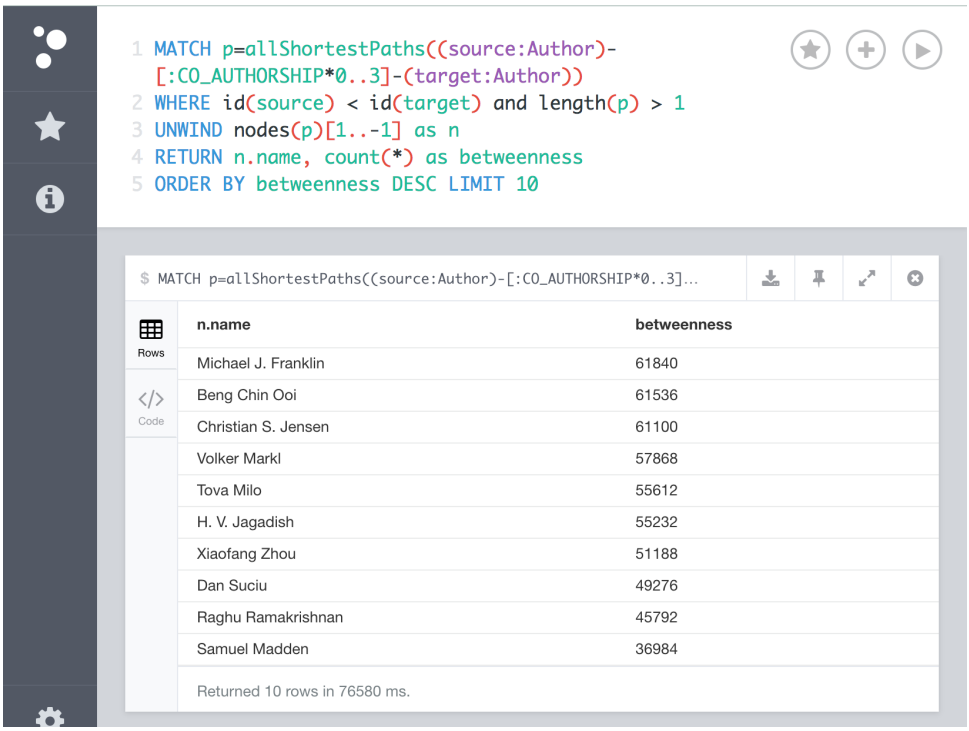


Figure 5.7: Result of experiment with the betweenness centrality measure for Authors

5.3.3 Graph as Measure

This type of measure is given by the aggregate graph itself, since it is a representation of data aggregated by dimensions. For our running example, this measure represents the topological disposition of the relationship between publications and authors when the data is aggregated according to the dimensions year and venue.

Figure **??** shows a screenshot of a subgraph retrieved from the aggregate graph. The subgraph was limited to 400 relationships in order to facilitate the visualisation. From the screenshot we can notice the topology distribution of author nodes that published works on

ICDE in 2016 and 2015, specially the authors that published in both editions of the conference.

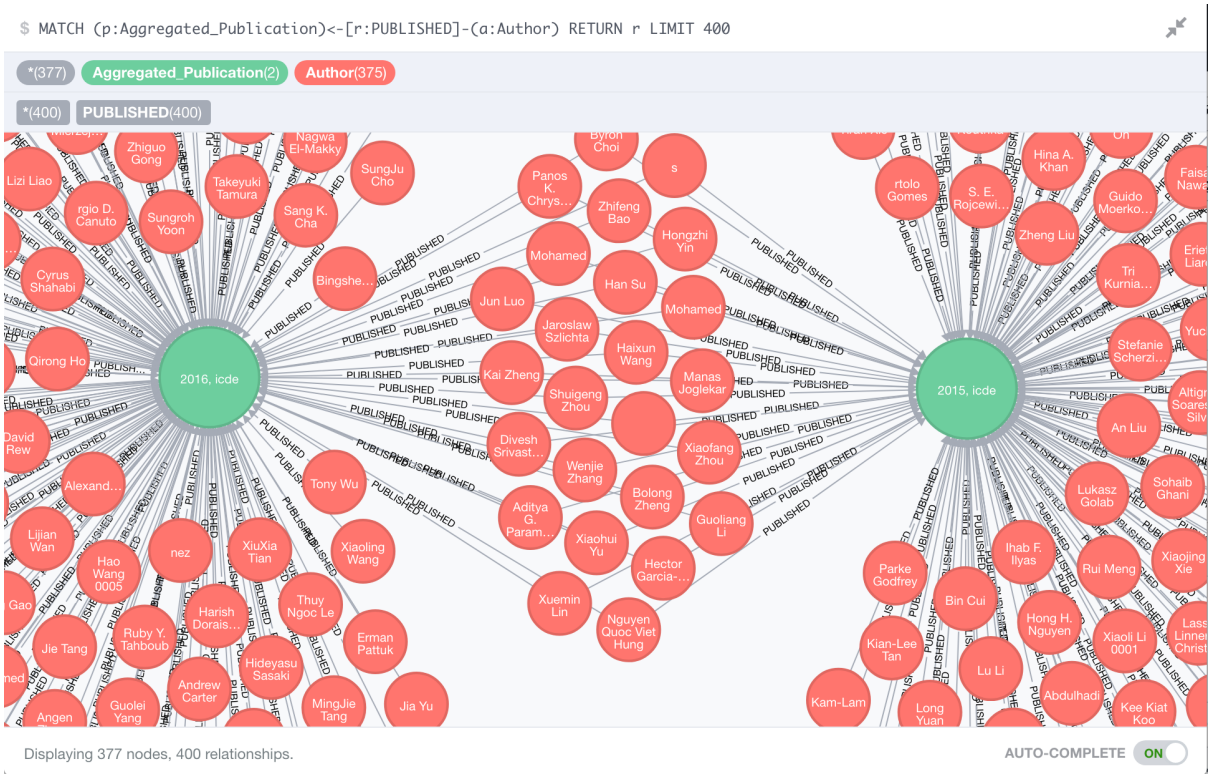


Figure 5.8: Result of experiment with the graph as a measure

5.3.4 Aggregate Graph versus Original Graph

For this experiment, we tried to obtain the same measure shown in Figure **??** from the original graph, i.e. without using the aggregate graph generated by the GA. The query was submitted to the original graph of publications and authors and it returns the count of all the publications from 2016 grouped by its venue.

Figure **??** shows the result of the query, listing the venues ordered by the amount of publications on the year of 2016. Notice that the result is the same as the one obtained in the experiment depicted in Figure **??**, but it is important to highlight that the query submitted to the original query took longer to be answered (625 ms), while the query submitted to the aggregate graph only took 14 ms to be processed. This means that, by using aggregate graph to obtain the measure, we reduce the time to process the query by 97.76%.

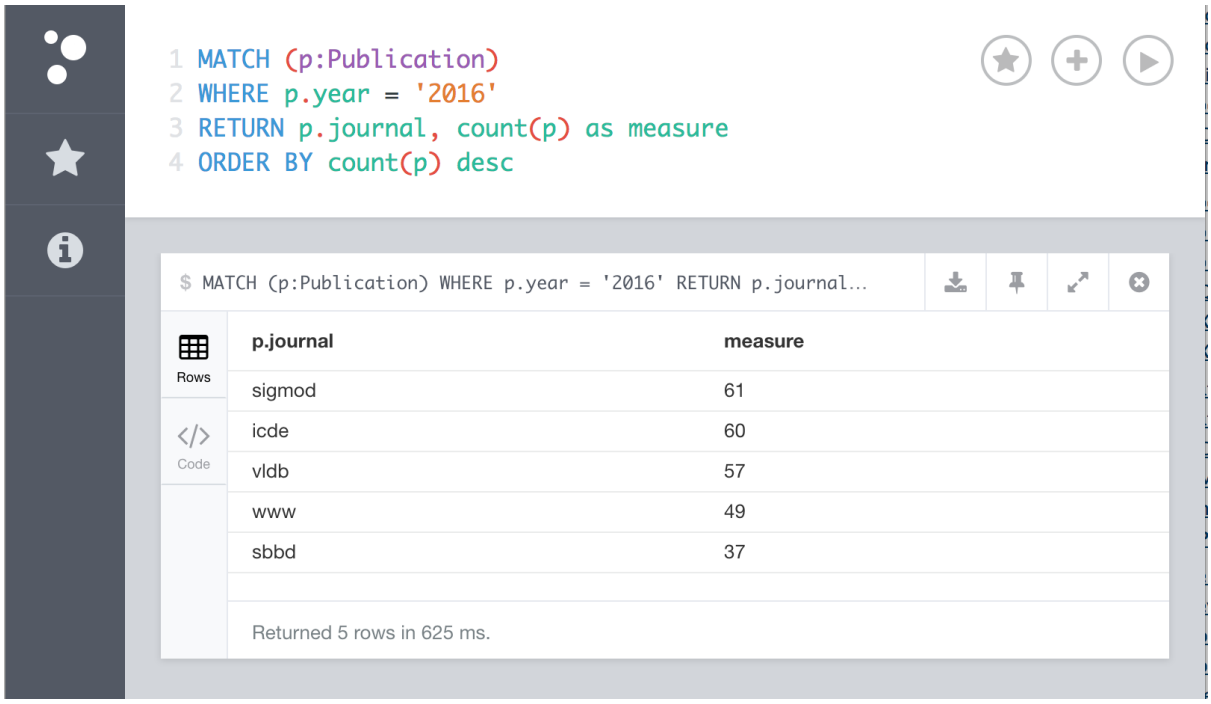


Figure 5.9: Result of experiment with content-based measure query submitted to the original graph

5.4 Result Analysis and Qualitative Comparison

Given the results obtained from the experiments described in the previous section, we are able to confirm the effectiveness of the proposed system in supporting multidimensional analysis on graph database. The presented solution also supports the execution of graph-based analysis, such as centrality measures. In this way we are able to extract three different types of measures from the same origin data aggregated according to user defined dimensions.

In comparison to other works in this area, the proposed system gives full support for heterogeneous graphs without the need to change the original data schema. Focusing the comparison with the framework presented in [**?**], our system is able to answer the same types of query without the extra step to change the data model of our original dataset. Table **??** summarises the comparison between existing frameworks and our proposed system.

Table 5.1: Comparison between existing frameworks and our proposed system

Framework	Graph	Dimensions	Operations
Graph Cube	Homogeneous	Vertex Attributes	Cuboid and Crossboid Query
Graph OLAP	Homogeneous	Informational and Topological	I-OLAP and T-OLAP Operations
HMGraph	Heterogeneous	Informational, Topological and Entity	I-OLAP, T-OLAP, Rotate and Stretch Operations
Pagrol	Homogeneous	Vertex and Edge Attributes	3 Query Category and Roll-up/Drill-down Operations
GRAD Graph Cubes	Heterogeneous	Inter-class and Intra-class	-
Using OLAP Queries for Data Analysis on Graph Databases	**Heterogeneous**	**Vertex and Edge Attributes**	**Roll-up, Drill-down, Slice, Dice and Centrality Measures Operations**

5.5 Difficulties Found

Unfortunately, we are unable to provide a more precise comparison between the framework in [**?**] and our proposal due to the lack of experiment description. A similar issue also applies to other works presented in Chapter 3, i.e. the state of the art for Graph Cubes and OLAP analysis on graph databases.

The published papers in this area fail in providing enough description on how the proposed solution was implemented and where the dataset used in the experiments is available. Furthermore, when experiments are presented in the paper, they only compare different version of the same framework. Until now, it hasn't been proposed an experiment that can be replicated amongst different frameworks.

The absence of a benchmark for experiments in this area makes the evaluation of the proposed system difficult. In order to compare ourselves to others, we can only rely on qualitative measures, based on the features presented by existing frameworks.

Another difficulty found during the implementation of the system was the size of the graph supported by Neo4J. The DBLP has more than 3.8 million publications, but we limited our dataset according to some conferences and journals, as mentioned before. At first, we wanted to conduct the experiments using at least 5.000 publications, but we were not able to load all the nodes and relationships to Neo4J. Our script to load the data kept getting error related to the communication with Neo4J REST API and we couldn't figure out how to fix it. In order to be able to finish all the experiments on time, we had to reduce the number of publications loaded to the database.

5.6 Final Considerations

In this chapter, we detailed how the system proposed by this work was implemented, specifying what were the technologies used in the process. The dataset used in the experiments was presented as well as the results obtained. Finally, we presented a qualitative comparison with existing solutions and listed the main difficulties found during the system's implementation.

Chapter 6

Conclusion

In this dissertation, we addressed the issue of executing multidimensional and network analysis on a Graph Database. At first, the main concepts related to this issue were presented. Then, we had an overview of academic works in this area, giving a brief summary of each paper and comparing the frameworks they presented according to the following criteria: type of graph supported, OLAP dimensions and operations implemented. After this initial study, it was possible to notice that only two frameworks supported heterogeneous graphs, but they required the generation of an intermediate data model in order to execute OLAP analysis.

Once the state of the art for the area was covered, we specified the architecture and the main components of a data analysis system over Graph Databases, which supports heterogeneous graphs and the execution of OLAP queries and network analysis algorithms without the need of an intermediate data model. Next, we proceed to describe in details how the proposed system was implemented and what were the technologies used. Finally, some experiments were presented, as well as an analysis of the results obtained and a qualitative comparison between the system built and the existing frameworks.

6.1 Contributions

As an outcome of the work presented in this document, we have the following contributions:

- Documented the state of the art for OLAP systems with Graph Databases and established

an analytical comparison between existing frameworks, defining the main characteristics to be taken into consideration for the comparison.

- Specification and implementation of a prototype for a Data Analysis System for Graph Databases, describing how its main components operate and how to build such system using open-source tools (e.g. Python and Neo4J).

- Implementation of OLAP operators and network analysis algorithms, providing a comprehensive analysis of the graph data. The execution of OLAP queries was possible due to a set of aggregate graphs that provided a multidimensional view of the graph data.

- Definition of experiments and qualitative analysis in comparison with existing frameworks. All the difficulties found in this area regarding execution of experiments and quantitative comparison between solutions were described.

6.2 Future Work

The user interface of the proposed system relies on the interface provided by Neo4J. An interesting work that could be done is building a specific interface, where the user could execute OLAP queries and network analysis algorithm in a more friendly way. This specific interface could also provide different types of data visualisation depending on the type of measure the user requested.

Regarding the difficulties found during the implementation of the proposed system, an interesting issue that still needs to be tackled is the standardisation of the experiments. Amongst academic papers in this area, there is no consensus on how a comparative experiment should be done. Define general experiments and a benchmark dataset would be a great contribution for this area, since it would allow a more precise quantitative comparison between existing solutions.