Distributed frequent subgraph mining in the cloud

Sabeur Aridhi

N d'ordre : D.U : **2400** EDSPIC : **626**

UNIVERSITY OF BLAISE PASCAL - CLERMONT II

Engineering Doctoral School of Clermont Ferrand

LIMOS - CNRS UMR 6158

UNIVERSITY OF TUNIS EL MANAR

LIPAH - LR11ES14

P H D T H E S I S

To obtain the degree of

Doctor of Philosophy

in Computer Science

Defended by

Sabeur ARIDHI

Distributed Frequent Subgraph Mining in the Cloud

publicly defended on November $29^{th}, 2013$

Committee:

Reviewers:	
Pr. Anne LAURENT	University of Montpellier 2, France
Pr. Takeaki UNO	National Institute of Informatics, Japan
Examiners:	
Pr. Jérome DARMONT	University of Lyon 2, France
Pr. Mohamed Mohsen GAMMOUDI	University of Manouba, Tunisia
Advisors:	
Pr. Engelbert MEPHU NGUIFO	University of Blaise Pascal, France
Pr. Mondher MADDOURI	University of Manouba, Tunisia
Dr. Laurent D'ORAZIO	University of Blaise Pascal, France

Acknowledgments

This thesis would not have been possible without the help, support and patience of my advisors, Prof. Engelbert Mephu Nguifo, Prof. Mondher Maddouri and Dr. Laurent d'Orazio, not to mention their advice and unsurpassed knowledge in cloud computing, data mining and machine learning.

Special thanks are directed to Prof. Anne Laurent and Prof. Takeaki Uno for having accepted to review my thesis manuscript and for their kind efforts to comply with all the administrative constraints. My thanks go as well to Prof. Mohamed Gammoudi and Prof. Jérome Darmont for having accepted to act as examiners of my thesis.

Thanks to Prof. Alain Quillot for having hosted me at the LIMOS Lab and allowed me working in comfortable conditions, with financial, academic and technical support.

This work was partially supported by the French Region of Auvergne thru the projects LIMOS-AGEATIS-NUMTECH and PREFON-META, by the French-Tunisian PHC project EXQUI, the CNRS Mastodons project PETASKY and the French-Tunisian DGRST/CNRS project IRRB.

I would also like to thank my mother, brothers and all my family. They were always supporting me and encouraging me with their best wishes. Their faith in me allowed me to be as ambitious as I wanted and helped me a lot through the past years. It was under their watchful eye that I gained so much drive and ability to tackle challenges head on.

Special thanks go to my beloved father, wishing he was able to see this important step of my life, and feel his pride with his son.

At last but certainly not least, I would like to thank my friends, they were always there cheering me up and standing by me through both good and bad times.

Clermont-Ferrand, 2013
Sabeur Aridhi

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Contents

Goals

This chapter summarizes the contents and describes the plan of the thesis. First, we highlight the emergence of graph mining and cloud computing. Then, we state the addressed issues in this thesis.

1.1 Context and motivations

1.1.1 Graph mining emergence

Graphs show up in a diverse set of disciplines, ranging from computer networks, social networks [Faloutsos 2004, Bonchi 2011] to bioinformatics [Saidi 2012, Pizzuti 2012], chemoinformatics [Goto 2002, Borgelt 2002] and others. These fields exploit the representation power of graph format to describe their associated data. In social network analysis, the graph representing a social network is composed by a set of linked individuals where edges express the relationships between couples of individuals. In bioinformatics, the graph representing a protein interaction network is composed by a set of linked proteins where an edge links each couple of proteins that participate in a particular biological function. Moreover, the protein structure itself can be considered as a graph where nodes represent the amino acids and edges represent the links between them. In this scope, graph mining has become an important topic of research that allows knowledge discovery from graph data. The main goal of graph mining is to develop algorithms to mine and analyze graph data. A surge of interest in this area, fueled largely by pattern mining from graphs, graph clustering and predictive models building for graphs.

Graph mining algorithms are used in several applications such as structural motif discovery, social network analysis and protein fold recognition. Moreover, graph data are being generated at unprecedented size. Indeed, graphs are now measured in terabytes or even petabytes. Analyzing them has become increasingly challenging. Consequently, it is necessary to provide distributed infrastructures that allow storing and processing these large volumes of graph data. In this context, cloud computing constitutes a promising environments for large scale graph data processing.

1.1.2 Cloud computing emergence

Cloud computing [Armbrust 2010] is a technology that involves a large number of computers that are connected through a real-time communication network (typically the Internet network) to maintain data and applications. Cloud computing

refers to both the applications delivered as services over the Internet and the hardware and system software in the data centers that provide those services. A simple example of cloud computing is a web-based email services, e.g., Gmail and Hotmail. The consumer just need an internet connection and he can start sending emails. The server and email management software is all on the cloud and is totally managed by the cloud service providers. The consumer gets to use the software alone and gains the benefits.

A promising application of cloud computing is large scale graph mining. In this context, cloud computing provides propitious frameworks that help with the design of massively scalable graph-based algorithms.

1.1.3 Distributed frequent subgraph mining issue

Finding recurrent and frequent substructures may give important insights on the data under consideration. These substructures may correspond to important functional fragments in proteins such as active sites, feature positions or junction sites. In a social network, frequent substructures can help to identify the few most likely paths of transmission for rumors or jokes from one person to another [Faloutsos 2004]. Mining these substructures from data in a graph perspective falls in the field of graph mining and more specifically in frequent subgraph mining.

Frequent subgraph mining is a main task in the area of graph mining and it has attracted much interest. Consequently, several subgraph mining algorithms have been developed, such as FSG [Kuramochi 2001], gSpan [Yan 2002], CloseGraph [Yan 2003], Gaston [Nijssen 2004] and ORIGAMI [Chaoji 2008]. However, existing approaches are mainly used on centralized computing systems and evaluated on relatively small databases [Wörlein 2005]. Nowadays, there is an exponential growth in both size and number of graphs in databases. This makes the above cited approaches face the scalability issue in addition to the prior storage issue. To overcome this problem, a distributed frequent subgraph mining approach that scales with the available huge amounts of graph data is needed. In this context, several distributed solutions have been proposed [Luo 2011, Hill 2012, Luo 2011]. Nevertheless, the data distribution techniques adopted by these works does not

include data characteristics. Consequently, these techniques may face scalability problems such as load balancing problems. To overcome this obstacle, a data partitioning technique that considers data characteristics should be applied.

1.1.4 Cost models for distributed pattern mining issue

Recently, distributed pattern mining approaches have become very popular. In most cases, the distribution of the pattern mining process generates a loss of information in the output results [Riondato 2012, Aridhi 2013]. Reducing this loss may affect the performance of the distributed approach and thus, the monetary cost when using cloud environments. In this context, cost models are needed to help selecting the best parameters of the used approach in order to achieve a better performance especially in the cloud [Kashef 2012, Nguyen 2012]. Existing cost models deal with distributed data mining systems that use classic architectures such as client-server and software agents [Krishnaswamy 2004, Marbán 2008, Ogunde 2011]. However, the proposed cost models do not include monetary aspect of the cloud computing paradigm where users only pay for the resources they use. Due to this fact, there is an urgent need to define new cost models for cloud-based pattern mining applications.

1.2 Contributions

This thesis deals with distributed frequent subgraph mining from huge graph databases. Firstly, we present a scalable and distributed approach for large scale frequent subgraph mining based on MapReduce framework [Dean 2008]. The proposed approach provides a density-based data partitioning technique that enhances the default one provided by MapReduce. Secondly, we present new cost models for cloud-based pattern mining approaches and we apply them to subgraph patterns.

1.2.1 First axis: distributed subgraph mining in the cloud

In the first axis, we propose a MapReduce-based framework to approximate large scale frequent subgraph mining. The proposed approach offers the possibility to apply any of the known subgraph mining algorithms in a distributed way. In addition, it allows many partitioning techniques for the graph database. In this thesis, we consider two instances of data partitioning: (1) the default partitioning method proposed by MapReduce framework and (2) a density-based partitioning technique. The second partitioning technique allows a balanced computational loads over the distributed collection of machines. We experimentally show that the proposed solution is reliable and scalable in the case of huge graph datasets. This contribution has been published in [Aridhi 2013].

1.2.2 Second axis: cost models for distributed pattern mining in the cloud

The second axis is dedicated to the monetary aspect of cloud-based pattern mining applications. It addresses the multi-criteria optimization problem of tuning thresholds related to distributed frequent pattern mining in cloud computing environment while optimizing the global monetary cost of storing and querying data in the cloud. To achieve this goal, we design cost models for managing and mining graph data with large scale pattern mining framework over a cloud architecture. We focus on distributed subgraph mining approach in the cloud. We also define four objective functions, with respect to the needs of customers. These needs can be expressed by a financial budget limit, a response time limit or a mining quality limit.

1.3 Outline

This thesis is organized as follows. In Chapter 2, we provide the required material to understand the basic notions of the graph mining research field. We also give a survey of the main graph management and mining techniques especially Frequent

Subgraph Mining (FSM) techniques.

In Chapter 3, we introduce the cloud computing research field and highlight its interestingness for distributed data mining and graph mining applications. We give an overview of cloud-based data mining techniques and we focus on distributed graph mining techniques in the cloud.

In Chapter 4, we propose a novel approach for large scale subgraph mining by means of a density-based partitioning technique, using the MapReduce framework. The proposed partitioning technique aims to balance computational load on a collection of machines to replace the default arbitrary one of MapReduce. We carry out an experimental study and show that the proposed approach decreases significantly the execution time and scales the subgraph discovery process to large graph databases.

In Chapter 5, we define new cost models for managing and mining patterns in a cloud. We define also a set of objective functions with respect to the needs and the financial capacity of customers. We carry out an experimental validation of our cost models.

In Chapter 6, we conclude this thesis by summarizing our contributions and highlighting some prospects.

Part I

Background and related works

Chapter 2

Graph mining

Contents

Goals

This chapter introduces the data mining and the graph mining fields. It is mainly dedicated to present, in a simplified way, the basic notions related to graph mining. We mainly focus on presenting graph properties and describing graph mining techniques especially Frequent Subgraph Mining (FSM) techniques.

2.1 Data mining and graph mining: basic notions

In this section, we present basic notions of graphs. We first present some definitions and notations used throughout this report. Then, we describe some topological graph properties and attributes.

2.1.1 Data mining

Data mining is a particular step in the process of Knowledge Discovery in Data (KDD) [Fayyad 1997] that consists of methodologies for extracting useful knowledge from volumes of data. The KDD process is outlined in Figure 2.1.

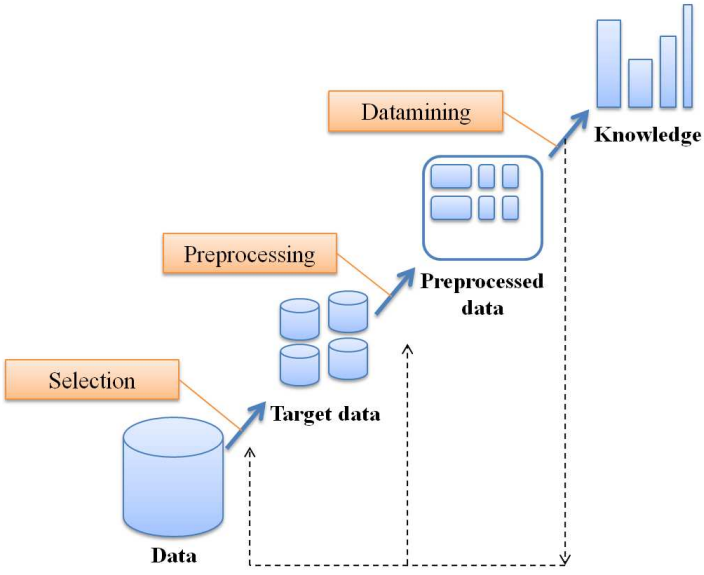


Figure 2.1: Steps of the KDD process.

The additional steps in the KDD process include the data selection, the preprocessing and the post-processing steps.

Definition 1 (Pattern) *In general, a pattern consists of a non-null finite feature that can characterize a given population P of objects. This pattern may be*

identified according to its high frequency in P, its rarity in other populations or based on other parameters.

Definition 2 (Data mining) *It consists of applying computational techniques that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (or models) over the data.*

Definition 3 (Knowledge discovery in data) *It is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.*

2.1.2 Graph mining

Graph mining consists of tools and techniques applied to graph data in order to discover interesting knowledge.

A graph is a collection of objects . Each object in a graph is called a node (or vertex). Corresponding to the connections in a network are edges (or links) in a graph. Each edge in a graph joins two distinct nodes.

Definition 4 (Graph) *A graph is denoted as $G = (V,E)$, where:*

- *V is a set of nodes (vertices).*
- *$E \subseteq V \times V$ is a set of edges (links).*

Figure 2.2 illustrates a graph containing five nodes (v_1,v_2,v_3,v_4 and v_5) and seven edges ((v_1,v_2),(v_1,v_4),(v_1,v_5),(v_2,v_4),(v_2,v_3),(v_3,v_5),(v_4,v_5)).

Definition 5 (Labeled Graph) *A labeled graph is a graph which is represented as a four tuple $G = (V,E,L,I)$ where:*

- *V is a set of nodes.*
- *$E \subseteq V \times V$ is a set of edges.*
- *L is a set of labels.*

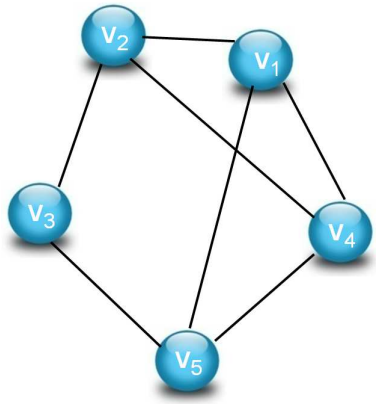


Figure 2.2: A graph with five nodes and seven edges.

- $I : V \cup E \longrightarrow L$ *is a labeling function.*

Figure 2.3 illustrates a labeled graph with $V = \{v_1, v_2, v_3, v_4, v_5\}$, $E = \{(v_1, v_2), (v_1, v_4), (v_1, v_5), (v_2, v_4), (v_2, v_3), (v_3, v_5), (v_4, v_5)\}$, $L = \{A, B, C, D, E, 1, 2, 3, 4, 5, 6, 7\}$, $I(v_1) = A$, $I(v_2) = B$,$I(v_3) = C$,$I(v_4) = D$,$I(v_5) = E$, $I((v_1, v_2)) = 5$, $I((v_1, v_4)) = 4$, $I((v_1, v_5)) = 7$, $I((v_2, v_4)) = 6$, $I((v_2, v_3)) = 1$, $I((v_3, v_5)) = 2$ *and* $I((v_4, v_5)) = 3$.

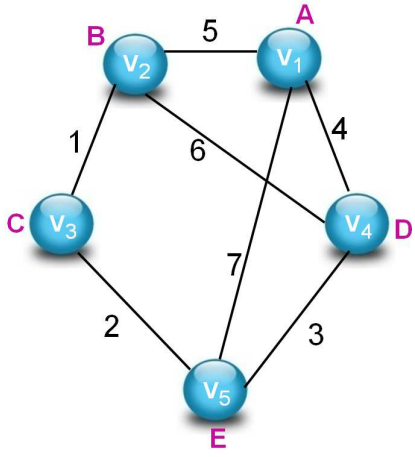


Figure 2.3: A labeled graph example.

The definitions of subgraph, graph isomorphism and subgraph isomorphism are

given as follows.

Definition 6 (Subgraph) *A graph $G' = (V', E')$ is a subgraph of another graph $G = (V, E)$ iff:*

- $V' \subseteq V$, *and*
- $E' \subseteq E \cap (V' \times V')$.

Figure 2.4 presents a subgraph of the graph in Figure 2.3. The presented subgraph contains three nodes (v_1,v_2 and v_4) and three edges ((v_1, v_4), (v_1, v_2) and (v_2, v_4)).

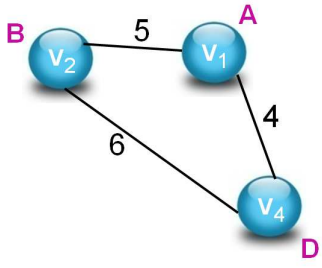


Figure 2.4: A subgraph of the graph shown in Figure 2.3.

Definition 7 (Graph isomorphism) *An isomorphism of graphs G and H is a bijection $f : V(G) \longrightarrow V(H)$ such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.*

We show in Figure 2.5, an isomorphic graph to the graph presented in Figure 2.3 with $f(v_1) = v'_1$, $f(v_2) = v'_2$, $f(v_3) = v'_3$, $f(v_4) = v'_4$ and $f(v_5) = v'_5$.

As shown in Figure 2.5, all adjacent vertices of the graph of Figure 2.5 are adjacent in the graph of Figure 2.4.

Definition 8 (Subgraph isomorphism) *A graph $G'(V', E')$ is subgraph-isomorphic to a graph $G(V, E)$ if there exists an injective function $f : V'(G') \longrightarrow V(G)$ such that $(f(u), f(v)) \in E$ holds for each $(u, v) \in E'$.*

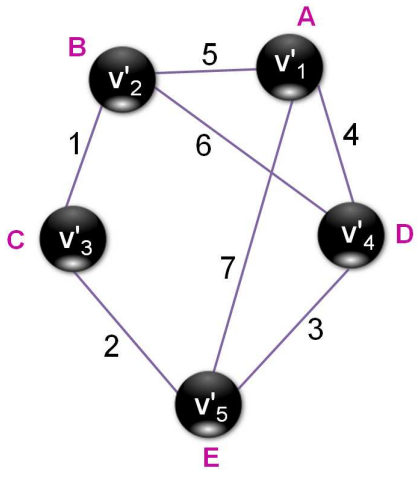


Figure 2.5: An isomorphic graph to the graph in Figure 2.3.

The graph presented in Figure 2.4 is subgraph-isomorphic to the graph presented in Figure 2.3. In fact, all adjacent vertices of the graph of Figure 2.4 are adjacent in the graph of Figure 2.3.

Definition 9 (Transitive closure of a graph) *The transitive closure of a graph $G = (V,E)$ is a graph $G^* = (V,E^*)$ such that E^* contains an edge (u,v) if and only if G contains a path (of at least one edge) from u to v.*

Definition 10 (Density) *The density of a graph $G = (V,E)$ is calculated by*

$$density(G) = 2 \cdot \frac{|E|}{(|V| \cdot (|V|-1))}. \tag{2.1}$$

The graph density measures the ratio of the number of edges compared to the maximal number of edges. A graph is said to be dense if the ratio is close to 1, and is considered as sparse if the ratio is close to 0. For example, the density of the graph in Figure 2.4 (G) can be calculated by

$$density(G) = 2 \cdot \frac{7}{(5 \cdot (5-1))} = 0.35.$$

Definition 11 (Degree) *The degree of a node u is the total number of adjacent*

nodes to u and is denoted by $degree(u)$.

For example, the degree of the node A of graph in Figure 2.3 is 3, since the node A has three adjacent nodes (B, C and D).

Definition 12 (Average degree) *Average degree of a graph is the average value of the degree of all nodes in the graph, i.e. the average degree of a graph $G = (V, E)$ is calculated by*

$$avgdegree(G) = \frac{\sum_i^{|V|} degree(u_i)}{|V|}. \tag{2.2}$$

For example, the average degree of the graph in Figure 2.3 can be calculated by

$$\begin{aligned} avgdegree(G) &= \frac{degree(A) + degree(B) + degree(C) + degree(D) + degree(E)}{5} \\ &= 2.8. \end{aligned}$$

Graphs are used to describe a variety of data including chemical data [Goto 2002, Borgelt 2002], biological data [Saidi 2012, Pizzuti 2012] and social network data [Faloutsos 2004, Bonchi 2011]. Nowaday, the high availability of graph data leads us to define graph data management techniques that allow usage and analysis of available data.

2.2 Graph data management

Graph data management [Aggarwal 2010] has become a dominant problem in many application areas, including social graphs [Faloutsos 2004], transportation networks [Speičys 2008] and biological interaction networks [Blin 2010, Pizzuti 2012]. These applications led to many new solutions for storing, partitioning, indexing and querying these graphs. In this section, we provide a review of recent graph data management algorithms and applications, namely, graph reachability queries, graph matching and keyword search.

2.2.1 Reachability queries

Graph reachability queries aim to test whether there is a path from one node to another in a large directed graph. Reachability queries are one of the most basic building blocks for many advanced graph operations. It is very useful in many applications, including applications in semantic web, biology networks and XML query processing. In general, two methods are used to answer reachability queries [Aggarwal 2010]. The first method consists of traversing the graph from the start node using breath-first search (BFS) or depth-first search (DFS) to see whether we can ever reach the end node. The second method consists of computing and storing the edge transitive closure of the graph. A reachability query can be answered by the second method by simply checking whether the edge linking the start node and the end node is in the transitive closure. We notice that for large graphs, neither of the two methods is feasible. In fact, the first method is too expensive at query time, and the second takes too much space. For these reasons, works like [Cheng 2008] and [Wang 2006] propose parallel and distributed algorithms for reachability.

2.2.2 Graph matching

The problem of graph matching [Aggarwal 2010, de Menibus 2011] consists of finding an approximate correspondence among the nodes of two graphs. This correspondence is based on one or more of the following structural characteristics of the graph:

- The labels on the nodes in the two graphs should be the same.
- The existence of edges between corresponding nodes in the two graphs should match each other.
- The labels on the edges in the two graphs should match each other.

These three characteristics may be used to define a matching between two graphs such that there is a one-to-one correspondence in the structures of the two

graphs. Such problems often arise in the context of several database applications such as schema matching, query matching, and vector space embedding. Generally, there are two kinds of graph matching: (1) exact graph matching and (2) approximate graph matching.

- The exact graph matching aims to determine a one-to-one correspondence between two graphs. Thus, if an edge exists between a pair of nodes in one graph, then that edge must also exist between the corresponding pair in the other graph. This may not be very practical in real applications in which approximate matches may exist, but an exact matching may not be feasible.

- In approximate graph matching, the main idea is to define an objective function which determines the similarity in the mapping between the two graphs. Fault tolerant mapping is a much more significant application in the graph domain, because common representations of graphs may have many missing nodes and edges.

2.2.3 Keyword search

The problem of keyword search [Aggarwal 2010] aims to determine small groups of link-connected nodes which are related to a particular keyword. For example, a web graph or a social network may be considered a massive graph, in which each node may contain a large amount of text data.

Because the underlying data assumes a graph structure, keyword search becomes much more complex than traditional keyword search over documents. The challenges lie in three aspects:

- Query semantics: the input data is often a single graph, so the algorithms must return subgraphs as answers. The keyword search algorithm must decide what subgraphs are qualified as answers.

- Ranking strategy: Based on the query semantics, answers of a keyword query are likely to be many subgraphs. However, each subgraph has its own underlying graph structure, with subtle semantics that makes it different

from other subgraphs that satisfy the query. Thus, the keyword search algorithm must take the graph structure into consideration and design ranking strategies that find most meaningful and relevant answers.

- Query efficiency: Many real life graphs are extremely large. A major challenge for keyword search over graph data is query efficiency, which, to a large extent, hinges on the semantics of the query and the ranking strategy.

2.3 Graph pattern mining

In this section, we first present graph patterns and their associated tasks and applications. Then, we present the frequent subgraph mining (FSM) task and we survey some recent FSM approaches.

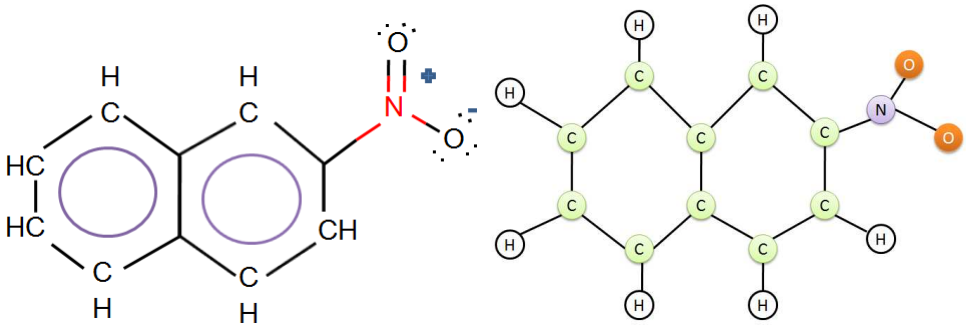
2.3.1 Applications of graph patterns

Graph patterns aim to characterise complex graphs. They help finding properties that distinguish real-world graphs from random graphs and detect anomalies in a given graph. Graph patterns are important for many applications such as chemoinformatics, bioinformatics and machine learning.

Chemical patterns: Chemical data is often represented as graphs in which the nodes correspond to atoms, and the links correspond to bonds between the atoms [Miyashita 1989, Ranu 2012, Wegner 2012]. In some cases, chemical patterns may be used as individual nodes. In this case, the individual graphs are quite small, though there are significant repetitions among the different nodes. This leads to isomorphism challenges in applications such as graph matching. The isomorphism challenge is that the nodes in a given pair of graphs may match in a variety of ways. The number of possible matches may be exponential in terms of the number of the nodes.

Figure 2.6 illustrates the graph representation of a real chemical compound.

Biological patterns: From a computer science point of view, the protein structure can be viewed as a set of elements. Each element can be an atom,



(a) A chemical compound

(b) A graph representation of a chemical compound

Figure 2.6: Graph representation of a chemical compound.

an amino acid residue or a secondary structure fragment. Hence, several graph representations have been developed to preprocess protein structure, ranging from coarse representations in which each vertex is a secondary structure fragment [Auron 1982, Zuker 1984] to fine representations in which each vertex is an atom [Saidi 2010, Saidi 2012]. Indeed, a protein interaction network can be represented by a graph where an edge links a couple of proteins when they participate in a particular biological function. Biological patterns may correspond to important functional fragments in proteins such as active sites, feature positions and junction sites.

Computer networked and Web data patterns: In the case of computer networks and the web, the number of nodes in the underlying graph may be massive [Erciyes 2013]. Since the number of nodes is massive, this can lead to a very large number of distinct edges. This is also referred to as the massive domain issue in networked data. In such cases, the number of distinct edges may be so large, that it may be hard to hold in the available storage space. Thus, techniques need to be designed to summarize and work with condensed representations of the graph data sets. In some of these applications, the edges in the underlying graph may arrive in the form of a data stream. In such cases, a second challenge arises from the fact that it may not be possible to store the incoming edges for future analysis. Therefore, the summarization techniques are especially essential

for this case. The stream summaries may be leveraged for future processing of the underlying graphs.

Figure 2.7 depicts the pipeline of graph applications built on frequent patterns.

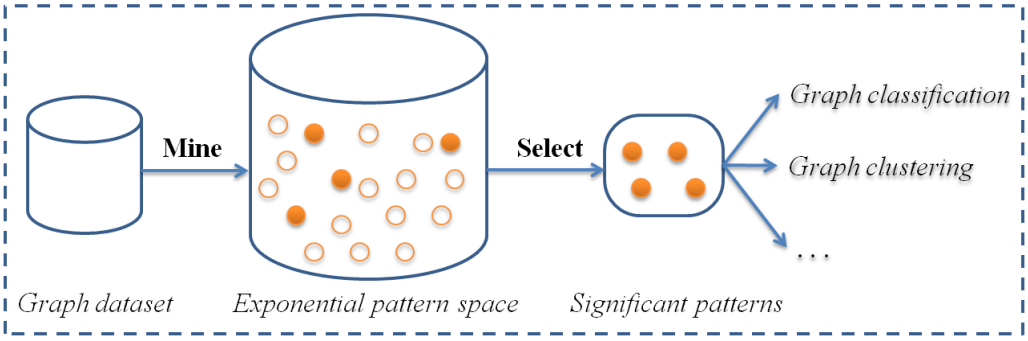


Figure 2.7: Graph patterns application pipeline.

In this pipeline, frequent patterns are mined first; then significant patterns are selected based on user-defined objective functions for different applications.

2.3.2 Frequent Subgraph Mining (FSM)

This section describes in details the frequent subgraph mining task [Cook 1994, Cook 2000]. First, it presents the motivations behind FSM. Then, it defines the problem of FSM. Finally, it describes some FSM algorithms.

2.3.2.1 Notations and FSM problem definition

There are two separate problem formulations for FSM: (1) graph transaction based FSM and (2) single graph based FSM. In graph transaction based FSM, the input data comprises a collection of medium-size graphs called transactions. In single graph based FSM the input data, as the name implies, comprise one very large graph.

In this work, we are interested in large scale graph transaction based FSM. The definitions of subgraph support and the graph transaction based FSM are given as follows.

Definition 13 (Subgraph relative support) *Given a graph database* $DB = \{G_1, \ldots, G_K\}$*, the* relative support *of a subgraph* G' *is defined by*

$$Support(G', DB) = \frac{\sum_{i=1}^{k} \sigma(G', G_i)}{|DB|}, \tag{2.3}$$

where

$$\sigma(G', G_i) = \begin{cases} 1, & \text{if } G' \text{ has a subgraph isomorphism with } G_i, \\ 0, & \text{otherwise.} \end{cases}$$

In the following, support refers to relative support.

Definition 14 (Graph transaction based FSM) *Given a minimum support threshold* $\theta \in [0,1]$*, the frequent subgraph mining task with respect to* θ *is finding all subgraphs with a support greater than* θ*, i.e., the set* $SG(DB, \theta) = \{(A, Support(A, DB)) : A \text{ is a subgraph of } DB \text{ and } Support(A, DB) \geq \theta\}$.

2.3.2.2 Properties of FSM techniques

Frequent subgraph mining approaches perform differently in order to mine frequent subgraphs from a graph dataset. Such differences are related to the search strategy, the generation of candidate patterns strategy and the support computing method.

Search strategy There are two basic search strategies employed for mining frequent subgraphs [Jiang 2013, Suryawanshi 2013]: the depth first search (DFS) strategy and the breadth first search (BFS) strategy.

The DFS strategy is a method for traversing or searching tree or graph data structures. It starts at the root (selecting a node as the root in the graph case) and explores as far as possible along each branch before backtracking. For the graph presented in Figure 2.3, a depth-first search starting at vertex B, which is adjacent to vertices A, C and D, we push D onto the stack, then C, then A. Next, we pop vertex A and iteratively process 2. Since A is connected to D and also E, we push E and then D onto the stack. Note that D is in the stack twice. Now

the important part is that since we added vertex E and D after adding vertex C, those will be processed before vertex C. That is, the neighbors of more recently visited vertices (vertex A) have a preference in being processed over the remaining neighbors of earlier ones (vertex B). This is precisely the idea of recursion: we do not finish the recursive call until all of the neighbors are processed, and that in turn requires the processing of all of the neighbors' neighbors, and so on.

The BFS strategy is limited to essentially two operations:

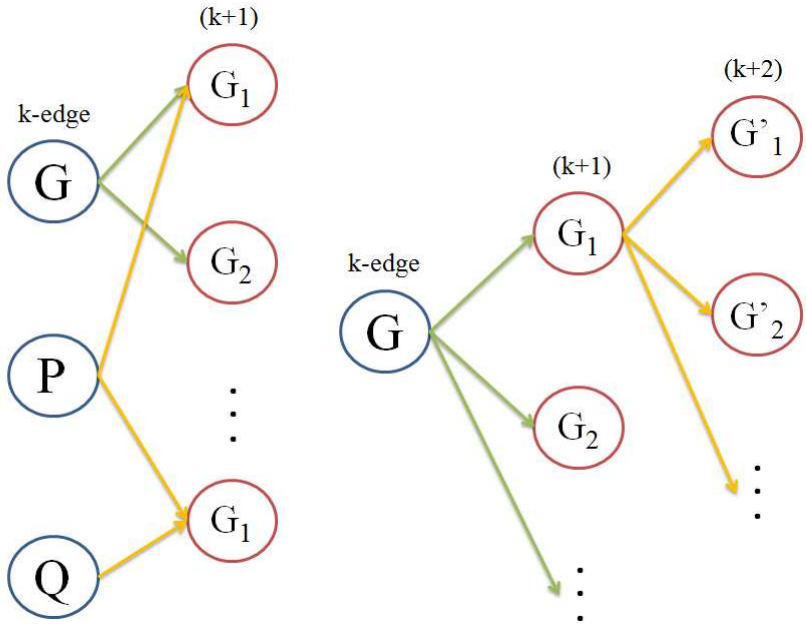
1. Visit and inspect a node of a graph,

2. Gain access to visit the nodes that neighbor the currently visited node.

The BFS begins at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on. Let us reexamine the graph of Figure 2.3. Starting again with vertex B, we add D, C, and A (in that order) to our data structure, but now we prefer the first thing added to our data structure instead of the last. That is, in the next step we visit vertex D instead of vertex A. Since vertex D is adjacent to nobody, the recursion ends and we continue with vertex C. Now vertex C is adjacent to E, so we add E to the data structure. That is, and this is the important bit, we process vertex A before we process vertex E. Notice the pattern here: after processing vertex B, we processed all of the neighbors of vertex B before processing any vertices not immediately adjacent to vertex one.

Generation of candidate patterns The generation of candidate patterns is the core element of the frequent subgraph discovery process. In this work, we consider two types of approaches: Apriori-based and pattern growth-based. Figure 2.8 shows the difference between the two approaches.

Apriori-based approaches share similar characteristics with Apriori-based frequent itemset mining algorithms [Agrawal 1994]. The search for frequent graphs starts with graphs of small size, and proceeds in a bottom-up manner. At each iteration, the size of the newly discovered frequent substructures is increased by one (see Figure 2.8b). These new substructures are first generated by joining two similar but slightly different frequent subgraphs that were discovered already.



(a) Apriori-based approach (b) Pattern growth-based approach

Figure 2.8: Apriori-based vs pattern growth-based approach.

The frequency of the newly formed graphs is then checked. The Apriori-based algorithms have considerable overhead when two size-k frequent substructures are joined to generate size-(k+1) graph candidates. Typical Apriori-based frequent substructure mining algorithms are discussed in the following paragraphs.

The pattern-growth mining algorithm extends a frequent graph by adding a new edge, in every possible position as shown in Figure 2.8b. A potential problem with the edge extension is that the same graph can be discovered many times. The gSpan [Yan 2002] algorithm solves this problem by introducing a right-most extension technique, where the only extensions take place on the rightmost path. A right-most path is the straight path from the starting vertex to the last vertex, according to a depth-first search on the graph.

Support computing Several methods are used for graph counting. Some frequent subgraph mining algorithms use transaction identifier (TID) lists for frequency counting. Each frequent subgraph has a list of transaction identifiers which support it. For computing frequency of a k subgraph, the intersection of

the TID lists of $(k-1)$ subgraphs is computed. Also, DFS lexicographic ordering can be used for frequency evaluation. Here, each graph is mapped into a DFS sequence followed by construction of a lexicographic order among them based on these sequences, and thus a search tree is developed. The minimum DFS code obtained from this tree for a particular graph is the canonical label of that graph which helps in evaluating the frequency. Embedding lists are used for support computing. For all graphs, a list is stored of embedding tuples that consist of (1) an index of an embedding tuple in the embedding list of the predecessor graph and (2) the identifier of a graph in the database and a node in that graph. The frequency of a structure is determined from the number of different graphs in its embedding list. Embedding lists are quick, but they do not scale very well to large databases. The other approach is based on maintaining a set of active graphs in which occurrences are repeatedly recomputed.

Types of patterns Several kinds of subgraph patterns can be mined with existing frequent subgraph mining algorithms. Algorithms including gSpan [Yan 2002] and FFSM [Huan 2003a] aim to mine frequent subgraphs. A frequent subgraph is a subgraph whose support is no less than a minimum support threshold.

Definition 15 (Frequent subgraph) *For a given graph database* $DB = \{G_1, \ldots, G_K\}$ *with* K *graphs and a minimum support threshold* $\theta \in [0,1]$, G' *is a* frequent subgraph *if* $Support(G', DB) \geq \theta$.

Another type of subgraph patterns that can be mined is the closed frequent patterns.

Definition 16 (Closed frequent subgraph) *A frequent subgraph* G' *is closed if and only if there is no supergraph* G *that has the same support as* G.

CloseGraph method [Yan 2003] was developed for mining closed frequent subgraphs by extension of the gSpan algorithm. A set of closed subgraph patterns has the same expressive power as the full set of subgraph patterns under the same minimum support threshold, because the latter can be generated by the derived set of closed graph patterns.

Algorithms like SPIN [Huan 2004] and MARGIN [Thomas 2010] aim to mine maximal frequent subgraphs.

Definition 17 (Maximal frequent subgraph) *A frequent subgraph G is maximal if and only if there is no frequent subgraph that contains G.*

The maximal pattern set is a subset of the closed pattern set. It is usually more compact than the closed pattern set. However, we cannot use it to reconstruct the entire set of frequent patterns.

Although the set of closed or maximal subgraphs is much smaller than the set of frequent ones, real-world graphs contain an exponential number of subgraphs.

A subgraph miner called ORIGAMI has been proposed in [Chaoji 2008] which discover representative subgraph patterns called α-orthogonal, β-representative graph patterns.

Definition 18 (α-orthogonal subgraph) *Two subgraph patterns are α-orthogonal if their similarity is bounded by a threshold α.*

Definition 19 (β-representative subgraph) *A subgraph pattern is a β-representative of another pattern if their similarity is at least β.*

The orthogonality constraint ensures that the resulting pattern set has controlled redundancy. For a given α, more than one set of graph patterns qualify as an α-orthogonal set.

2.3.2.3 FSM algorithms

In this section, we detail three existing subgraph miners, namely FSG [Kuramochi 2001], gSpan [Yan 2002] and Gaston [Nijssen 2004].

FSG algorithm. FSG [Kuramochi 2001] uses the Apriori level-wise approach [Agrawal 1994] to find frequent subgraphs. In this algorithm, edges in a graph are considered as frequent items in traditional itemset mining. Hence, the size of a graph can be increased only by adding a single edge to the subgraph. Each time, candidate subgraphs are generated by adding edges to previous subgraph. Hence,

candidate subgraph generated in the current step must be greater in size than the subgraph generated before. FSG uses sparse graph representation to store input graph transactions, intermediate candidate graphs and frequent subgraphs. Adjacency list representation is used to store each transaction graph. To check unique subgraph, FSG uses canonical labeling technique in which each graph has unique canonical label. Canonical labels of two graphs are the same only when both graphs have the same topological structure, edges and vertices labels. This technique is used to check isomorphic graphs. The subgraph mining process adopted by FSG incorporates various optimizations for candidate generation and frequency counting which enables it to scale to large graph datasets [Kuramochi 2001].

gSpan algorithm. gSpan [Yan 2002] is an algorithm for frequent graph-based pattern mining in graph datasets based on DFS lexicographic order and its properties [Yan 2002]. gSpan discovers frequent substructures without candidate generation, which aims to avoid the candidate generation phase and the subgraph isomorphism test [Kijima 2012]. Based on DFS, a hierarchical search tree is constructed. By pre-order traversal of the tree, gSpan discovers all frequent subgraphs with a support greater than a support threshold. Since the design of the algorithm combines the subgraph isomorphism test and frequent subgraph growth into one procedure, gSpan accelerates the mining process [Yan 2002].

Gaston algorithm. Gaston [Nijssen 2004] is a substructure/subgraph-finding algorithm that uses steps of increasing complexity to generate frequent substructures. Gaston searches first for frequent paths, then frequent free trees and finally cyclic subgraphs [Nijssen 2004]. It stores all embeddings, to generate only refinements that actually appear and to achieve fast isomorphism testing. The main insight is that there are efficient ways to enumerate paths and trees. By considering fragments that are paths or trees first, and by only considering general graphs with cycles at the end, a large fraction of the work can be done efficiently. Gaston defines a global order on cycle-closing edges and only generates those cycles that are larger than the last one. Duplicate detection is done in two phases: hashing to pre-sort and a graph isomorphism test for final duplicate detection. We notice also that the frequency counting process for Gaston is carried out with the help of embedding lists, where all the occurrences of a particular label are stored in the

embedding lists.

In order to give a summary, we present in Table 2.1 the details of recent algorithms for frequent subgraph discovery with respect to properties of frequent subgraph mining techniques presented in Section 2.3.2.2. We notice that the algorithms presented in Table 2.1 output a set of frequent subgraphs.

2.4 Conclusion

In this chapter, we presented preliminary notions of graphs and we described graph data mining and management techniques. We focused on frequent subgraph mining, one of the main goal of this thesis. A notable issue in this context is that existing approaches are mainly used on centralized computing systems and evaluated on relatively small databases. In addition, the exponential growth in both the graph size and the number of graphs in databases makes the existing approaches face the scalability issue. To overcome this issue, a distributed subgraph mining approach that scales with the available huge amount of graph data is needed especially with the advent of cloud computing. In the next chapter, we will present the cloud computing domain and we will discuss some cloud-based subgraph mining techniques.

Table 2.1: Algorithmic aspects of three popular algorithms.

Algorithm	Graph representation	Search strategy	Generation of candidate patterns	Support computing
gSpan	Adjacency list	Depth first search (DFS)	Pattern growth approach	DFS lexicographic ordering
FSG	Adjacency list	Depth first search (DFS)	A priori-based approach	Transaction identifier (TID) lists
Gaston	Hash table	Depth first search (DFS)	Pattern growth approach	Embedding lists

Key points

- We presented basic notions of graphs that help understanding graph related aspects. These notions will be used throughout this manuscript.
- We described the graph pattern mining field and its related notations and applications.
- We focused on the frequent subgraph mining task by presenting its problem definition and by presenting a review of frequent subgraph mining techniques.
- We highlighted the raised issue with existing subgraph mining techniques in the case of large scale data. At the end of this chapter, we mentioned the need of distributing the task of frequent subgraph mining in order to handle scalability issues.

CHAPTER 3

Cloud-based graph mining

Contents

Goals

This chapter introduces the cloud computing research field and highlights its interestingness for distributed data mining and graph mining applications. We describe cloud computing models such as the deployment model, the business model and the programming models. We give an overview of cloud-based data mining techniques and we focus on distributed graph mining techniques in the cloud.

3.1 What is cloud computing?

Cloud computing [Armbrust 2010] is a technology that involves a large number of computers that are connected through a real-time communication network (typically the Internet network) to maintain data and applications. Cloud computing refers to both the applications delivered as services over Internet and the hardware and systems software in the data centers that provide those services.

This promising technology allows much more efficient computing by centralizing data storage, processing and bandwidth. A simple example of cloud computing is a Web-based email services, e.g., Gmail and Hotmail. They deliver a cloud computing service: users can access their email "in the cloud" from any computer with a browser and Internet connection, regardless of what kind of hardware is on that particular computer. The emails are hosted on provider's servers, rather than being stored locally on the client computer.

Nowadays, businesses are moving to the cloud because it helps improving cash flow and offers much more benefits.

3.2 Cloud benefits

The following are some of the possible benefits for those who offer cloud computing-based services and applications:

- **On demand self services** Computer services such as email, applications, network or server service can be provided without human interaction with each service provider.
- **Cost savings** Initial expense is really cost effective. Cloud computing uses the "pay as you go" billing model which has a per usage basis. This pay-as-you-go model means that usage is metered and you pay only for what you consume. The maintaining and service costs are much lower compared to traditional computing methods.
- **Rapid elasticity** Cloud services can be rapidly and elastically provisioned, in some cases automatically, to quickly scale-out and rapidly released to

quickly scale-in. If we consider industrial level, companies can start with a small deployment and grow to a large deployment fairly rapidly, and then scale back if necessary.

- **Reliability** We can use multiple redundant clouds to do our computation purposes. This supports business continuity and disaster recovery.

- **Maintenance** Cloud service providers (CSPs) do the system maintenance, and access is through application programming interfaces (APIs) that do not require application installations onto PCs, thus further reducing maintenance requirements.

The success of cloud computing comes not only from the above mentioned benefits, but also from its related models such as the deployment model, the business model and the programming models.

3.3 Cloud computing deployment models

Cloud services can be deployed in different manners, depending on the organizational structure and the provisioning location [Armbrust 2009]. Four deployment models are usually distinguished, namely public, private, community and hybrid cloud service usage.

3.3.1 Public cloud

In a public cloud, services are rendered over a network that is open for public use. Public cloud allows the users to access various important resources on cloud, such as software, applications or stored data. Generally, public cloud service providers (CSPs) like Amazon AWS [Inc 2013a], Microsoft [Inc 2013c] and Google [Inc 2013b] own and operate the infrastructure and offer access only via Internet. Figure 3.1 illustrates a public cloud where users from different organisation (users with three different colors) can access to cloud services.

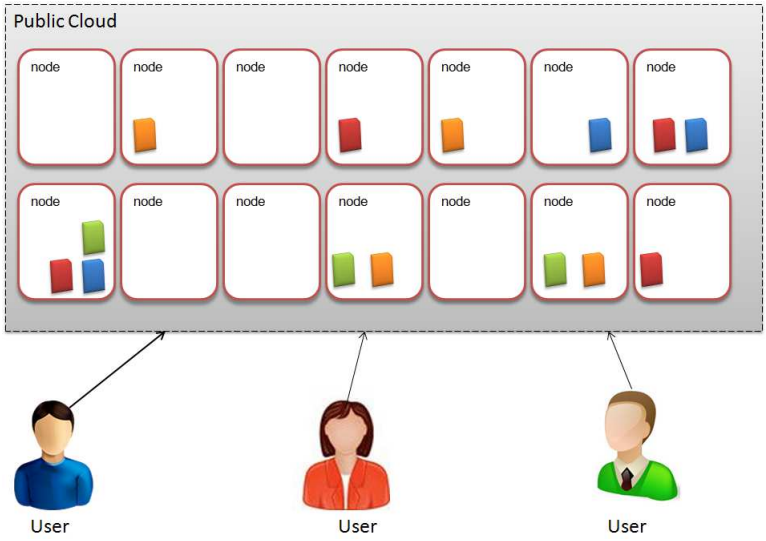


Figure 3.1: Public cloud.

3.3.2 Private cloud

Private cloud computing systems emulate public cloud service offering within an organization's boundaries to make services accessible for one specific organization. Private cloud computing systems make use of virtualization solutions and focus on consolidating distributed information technology (IT) services often within data centers belonging to the company. Figure 3.2 presents a private cloud where users from a specific company (users with green color) can access only to the private cloud of their company.

3.3.3 Community cloud

A community cloud is a generalization of a private cloud in which the cloud infrastructure is shared by organizations with similar requirements. A community cloud differs from private cloud by the fact that the cloud infrastructure is accessible by more than one organization. Figure 3.3 illustrates community cloud where the cloud infrastructure is shared between three organizations and user from each organization accesses to services provided by the three organizations.

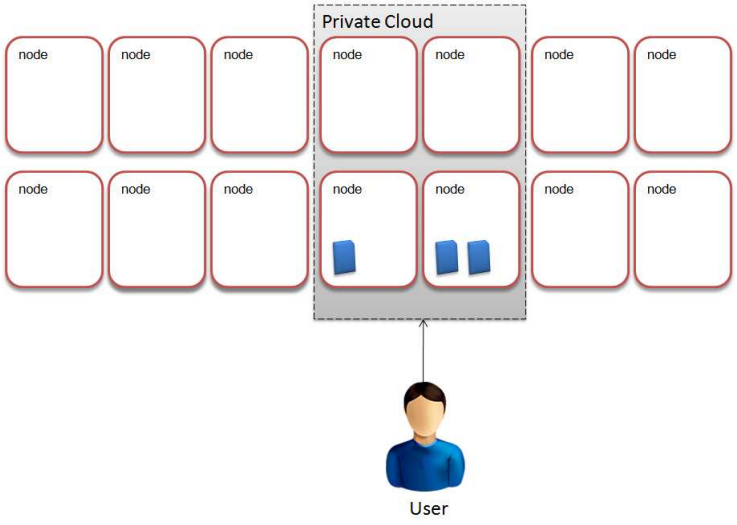


Figure 3.2: Private cloud.

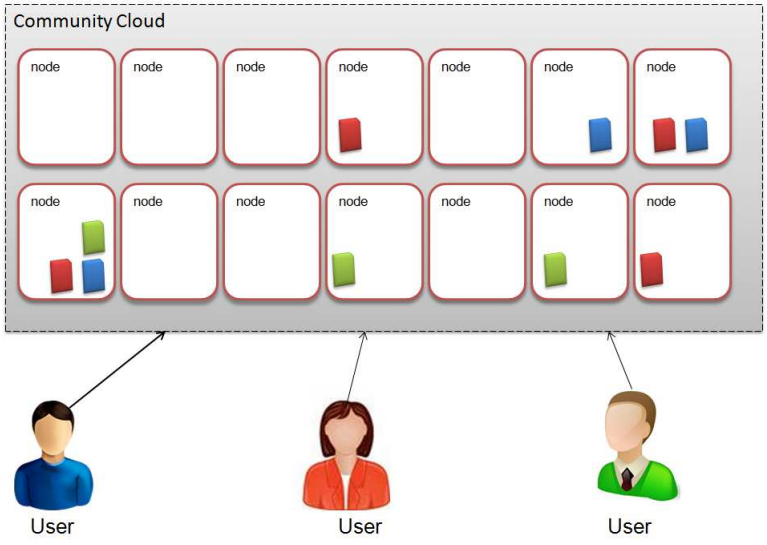


Figure 3.3: Community cloud.

3.3.4 Hybrid cloud

Hybrid cloud is a composition of cloud services of different cloud computing systems, e.g., private and public cloud services. A hybrid cloud service deployment model offers the benefits of multiple deployment models. Such composition ex-

pands deployment options for cloud services, allowing organizations to use public cloud computing resources to meet temporary needs. This capability enables hybrid clouds to employ cloud bursting for scaling across clouds. Figure 3.4 presents a hybrid cloud where users can access not only to their related private cloud but also to public or community cloud.

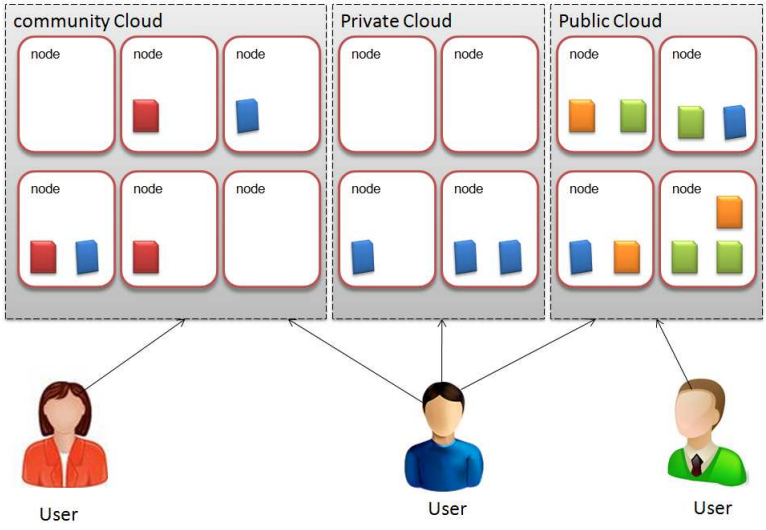


Figure 3.4: Hybrid cloud.

As shown in Figure 3.4, the user colored in blue can access to the private cloud related to his company and also to the public and the community cloud.

3.4 Cloud computing business model

Cloud computing is not a stand-alone technology. It is a business and delivery model enabled by existing technologies modified for remote, on-demand, and fractional consumption. In this section we describe basic cloud computing service models and we detail some pricing policies of cloud service providers.

3.4.1 Basic cloud computing service models

Cloud computing providers offer their services according to three fundamental models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) where IaaS is the most basic and each higher model abstracts from the details of the lower models (see Figure 3.5).

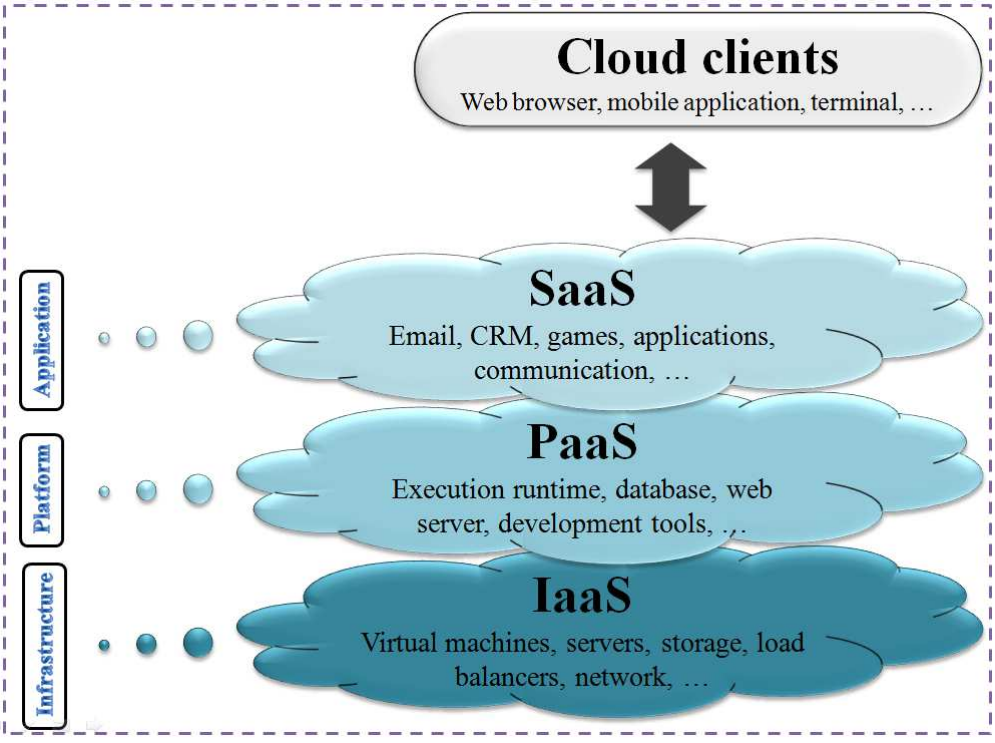


Figure 3.5: Cloud computing layers.

Infrastructure as a Service (IaaS) Different from conventional hosting services, IaaS comprises the sharing of infrastructure resources for running software in the cloud that would ordinarily be deployed and operated on-premise. IaaS provides consumers with the processing, storage, networks, and other fundamental computing resources required for running applications. The channel is both a provider and broker of IaaS by building and delivering cloud-based infrastructures, reselling infrastructure services, and supporting organizations in their use and operations of the services.

Platform as a Service (PaaS) A platform upon which users can develop and deploy services for consumption. PaaS providers include Microsoft Azure, Salesforce.com's [Woollen 2010], Google's App Engine [Ciurana 2009] and Heroku [Middleton 2013]. The channel can either use PaaS to develop its own unique offerings or resell capacity and support to organizations that require PaaS services. For the channel, PaaS is about exercising expertise to both leverage platforms and support cloud-based platforms.

Software as a Service (SaaS) Applications running on a cloud infrastructure via a thin client or browser. SaaS includes such services as managed email, customer relationship management (CRM), and office productivity applications. Vendors providing such services are reselling their offerings through solution providers who can add deployment, migration, training, and support services on top of the core offering.

The economics of its underlying sales model are what separates cloud computing from the conventional, on-premise technology model. Since cloud computing is essentially a subscription-based service, it is sold and billed as a recurring operational expense (as opposed to a one-time or limited capital expense).

3.4.2 Pricing policies

Cloud service providers supply a variety of resources, such as hardware (CPU, storage, networks), development platforms with different services and pricing. In order to have an overview of such a pricing policy, the following examples present a simplified version of both Windows Azure offer [Inc 2013c] and Amazon Web Service (AWS) offer [Inc 2013a]. The objective of this description is indeed not to compare the different providers, but to provide an idea about CSPs pricing offers[1].

3.4.2.1 Windows Azure offer

Windows Azure provides a variety of computing resources that can be rented (very small, small, medium, large and very large) at various prices, as illustrated in Table 3.1.

[1]The presented prices were taken on September 2013

Table 3.1: Windows Azure computing prices.

Type	Virtual cores	RAM	Price per hour
Very small	shared	768 MB	$0.02
Small	1	1.75 GB	$0.08
Medium	2	3.5 GB	$0.16
Large	4	7 GB	$0.32
Very large	8	14 GB	$0.64

Bandwidth consumption is billed with respect to data volume (see Table 3.2). In this model, input data transfers are free, whereas output data transfer cost varies with respect to data volume, with an earned rate when volume increases.

Table 3.2: Windows Azure bandwidth prices (output data).

Data volume	Price per month
First 5 GB per month	free
5 GB-10 TB per month	$0.12 per GB
40 TB per month	$0.09 per GB
100 TB per month	$0.07 per GB
350 TB per month	$0.05 per GB

Finally, Windows Azure Storage provides storage of non-relational data, including storage blob[2], table and disk. It introduces two options for storage: locally and geographically redundant. The locally redundant storage option allows multiple copies of data within a single sub-region to provide the highest level of durability. The geographically redundant storage option offers an extra level of durability by replicating data between two remote sub-regions.

In this model, the price varies with respect to data volume, with an earned rate when volume increases (see Table 3.3).

[2]A blob is a collection of binary data stored as a single entity in a database management system.

Table 3.3: Windows Azure storage prices.

Data volume	Price per month	
	Geographically redundant storage	Locally redundant storage
First 1TB per month	$0,095 per GB	$0,07 per GB
Next 49TB per month	$0,08 per GB	$0,065 per GB
Next 450TB per month	$0,07 per GB	$0,06 per GB
Next 500TB per month	$0,065 per GB	$0,055 per GB
Next 4PB per month	$0,06 per GB	$0,045 per GB
Next 4PB per month	$0,055 per GB	$0,037 per GB

3.4.2.2 Amazon Web Service offer

Amazon Web Service (AWS) provides a variety of computing resources that can be rent (extra small, small, large and extra large) at various prices, as illustrated in Table 3.4.

Table 3.4: Amazon Elastic Compute Cloud (EC2) computing prices.

Type	Virtual cores	RAM	Price per hour
Small	1	1.7GB	$0.042
Medium	1	3.75GB	$0.085
Large	2	7.5GB	$0.17
Extra large	4	15GB	$0.339

For example, the costs for a small instance (1.7GB RAM, 1 virtual core) is $0.042 per hour.

Table 3.5 presents EC2 bandwidth prices with respect to data volume.

Table 3.5: EC2 bandwidth prices (output data).

Data volume	Price per month
First 1GB per month	free
2GB-10TB per month	$0.12 per GB
40TB per month	$0.09 per GB
100TB per month	$0.07 per GB
350TB per month	$0.05 per GB

In this model, input data transfers are free, whereas output data transfer cost varies with respect to data volume, with an earned rate when volume increases.

Finally, AWS Storage provides storage capabilities. Amazon EBS proposes a fixed price ($0.10 per GB), whereas Amazon S3 (Table 3.6) enables an earned rate when volume increases.

Table 3.6: Amazon S3 storage prices.

Data volume	Price per month
Frist 1 TB	$0,14 per GB
Next 49 TB	$0,125 per GB
Next 450 TB	$0,11 per GB

The choice of the cloud service providers depends not only on the pricing offer but also on the studied problem and the used programming models.

3.5 Cloud computing programming models

Cloud computing offers multiple programming models in order to facilitate the efficient design of cloud-based applications. Cloud programming models often take existing programming models as their base and adapt them a bit for cloud usage. In this section, we present the three main cloud programming models, namely Web Services programming model, composite applications programming model and MapReduce programming model.

3.5.1 Web services programming model

Service oriented architecture (SOA) [Erl 2004] is the underlying structure supporting communications between services, which preceded cloud computing and has definitely influenced it [Barry 2013]. SOA allows easy cooperation of a large number of connected computers. Thus, every computer can run an arbitrary number of services. SOA is a very recommended programming model in the cloud thanks to its related standards [Erl 2004]. However, it presents major limitation related

to fault tolerance. In fact, SOA do not provide fault management component that helps development of fault-tolerant applications in the cloud.

3.5.2 Composite applications programming model

Composite applications are techniques for composing applications together from multiple components. As it is the case, they are closely related to SOA and benefit from cloud-driven standardization in the same way [Tejedor 2011, Eidson 2001]. We identify a couple of composite types:

- A composite that tries to orchestrate components to achieve some batch jobs.
- An interactive composite (called also mash-ups). This type involves a user interface and interacts with a user. It must deal with the fact that contracts are loose and failure of some components is to be expected. This is especially significant when you cannot wait since the user wants a response.
- A combination of batch and interactive components.

The composite applications programming model is a simple way to design cloud-based applications. Nevertheless, it is not used in the context of very large scale cloud-based applications since it does not provide a robust framework for procesing huge amount of data.

3.5.3 MapReduce programming model

MapReduce [Dean 2008] is a framework for processing highly distributable problems across huge datasets using a large number of computers (nodes). It was developed within Google as a mechanism for processing large amounts of raw data, for example, crawled documents or web request logs. This data is so large, it must be distributed across thousands of machines in order to be processed in a reasonable amount of time. This distribution implies parallel computing since the same computations are performed on each CPU, but with a different dataset. MapReduce is an abstraction that allows to perform simple computations while

hiding the details of parallelization, data distribution, load balancing and fault tolerance [Dean 2008]. The central features of the MapReduce framework are two functions, written by a user: *Map* and *Reduce*. The *Map function* takes as input a pair and produces a set of intermediate key-value pairs $\langle key, value \rangle$. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function. The Reduce function accepts an intermediate key I and a set of values for that key. It merges these values together to form a possible smaller set of values.

Figure 3.6 shows an execution workflow of a MapReduce program.

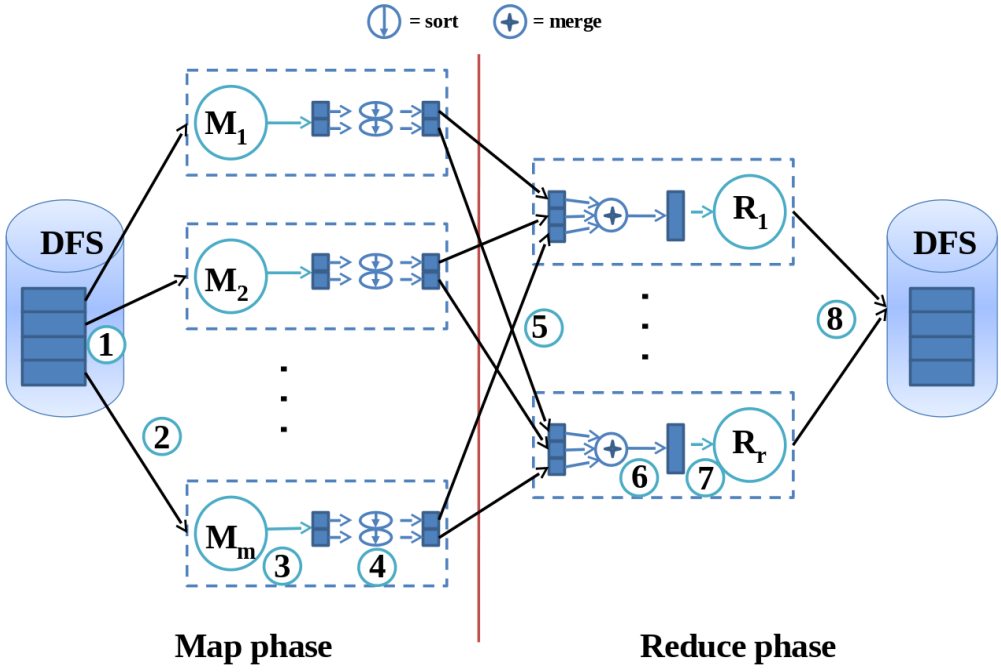


Figure 3.6: MapReduce execution overview.

The different tasks in Figure 3.6 are numbered as means of identifying the tasks in the following description. The execution workflow is made-up of two main phases:

- Map phase, which contains the following steps:

  1. The input file is splitted into several pieces of data. Each piece is called

a split or chunk.

2. Each node hosting a map task, called a mapper, reads the content of the corresponding input split from the distributed file system.
3. Each mapper converts the content of its input split into a sequence of key-value pairs $\langle k, v \rangle$ and calls the user-defined Map function for each key-value pair. The produced intermediate pairs $\langle I, v' \rangle$ are buffered in memory.
4. Periodically, the buffered intermediate key-value pairs are written to r local intermediate files, called segment files, where r is the number of reducer nodes. The partitioning of data into r regions is achieved by a partitioning function which ensures that pairs with the same key are always allocated to the same segment file. In each partition, the data items are sorted by values. The sorted chunks are written to local storage.

- The reduce phase, made of the following steps:

5. On the completion of a map task, the reducers (i.e., nodes executing the reduction function), will pull over their corresponding segments.
6. When a reducer read all intermediate data, it sorts the data by the intermediate keys so that all occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort is used. The reducer then merges the data to produce for each intermediate key I a single pair $\langle I, list(v') \rangle$.
7. Each reducer iterates over the sorted intermediate data and passes each pair $\langle I, list(v') \rangle$ to the user's reduce function.
8. Each reducer writes its final results to the distributed file system.

All these programming models have been used for building reliable, efficient and scalable cloud-based applications. In the context of this thesis, we are interested in cloud-based data mining applications especially the distributed subgraph mining

techniques in the cloud. The next section provides a survey of recent cloud-based data mining applications.

3.6 Cloud-based techniques

As cloud computing is penetrating more and more in all ranges of business and scientific computing, it becomes a great area to be focused by data mining and especially by graph mining. The distribution of data mining techniques through cloud computing will allow the users to retrieve meaningful information from virtually integrated data that reduces the costs of infrastructure and storage. In this section, we survey distributed data mining and graph mining techniques in cloud. We focus on cloud-based subgraph mining algorithms which constitutes the addressed issue in this thesis.

3.6.1 Cloud-based data mining techniques

In this section, we present an overview of distributed data mining and machine learning techniques. Then, we give a summary of the described approaches with respect to the format of the input data and the output patterns.

3.6.1.1 Distributed data mining techniques

Data mining and machine learning hold a vast scope of using the various aspects of cloud computing for scaling existing algorithms and solving some of the related challenges.

NIMBLE. NIMBLE [Ghoting 2011a] is a portable infrastructure that has been specifically designed to enable the implementation of parallel machine learning and data mining algorithms. The NIMBLE approach allows to compose parallel data mining algorithms using reusable (serial and parallel) building blocks that can be efficiently executed using MapReduce and other parallel programming models. The programming abstractions of NIMBLE have been designed with the intention of parallelizing data mining computations and allow users to specify data parallel, iterative, task parallel, and even pipelined computations. The NIMBLE

approach has been used to implement some popular data mining algorithms such as k-Means clustering [MacQueen 1967] and pattern growth-based frequent itemset mining [Han 2000], k-Nearest Neighbors [Coomans 1982] and random decision trees [Breiman 2001].

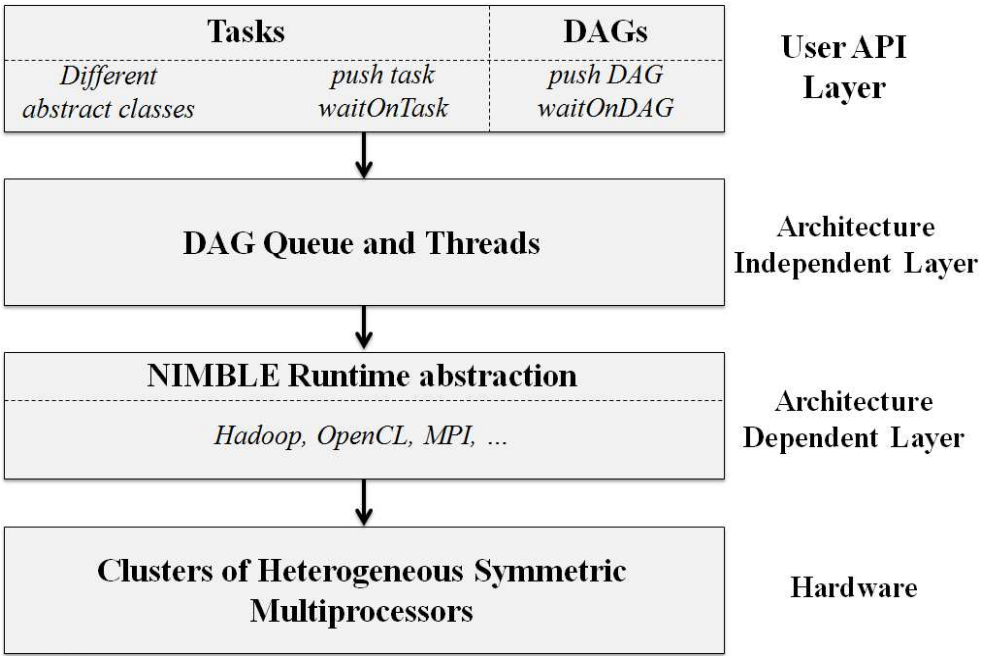


Figure 3.7: An overview of the software architecture of NIMBLE [Ghoting 2011a].

As shown in Figure 3.7, NIMBLE is organized into four distinct layers:

1. The user API layer, which provides the programming interface to the users. Within this layer, users are able to design tasks and directed acyclic graphs (DAGs) of tasks to express dependencies between tasks. A task processes one or more datasets in parallel and produces one or more datasets as output.

2. The architecture independent layer, which acts as the middleware between the user specified tasks/DAGs, and the underlying architecture dependent layer. This layer is primarily responsible for the smart scheduling of tasks, and delivering the completion notifications of these tasks to the users.

3. The architecture dependent layer, which consists of harnesses that allow

NIMBLE to run portably on various platforms. Currently, NIMBLE only supports execution on the Hadoop platform.

4. The hardware layer, which consists of the used cluster.

Mahout. The Apache's Mahout project [Foundation 2010] provides a library of machine learning implementations. The primary goal of Mahout is to create scalable machine learning algorithms that are free to use under the Apache license. It contains implementations for clustering [Esteves 2011, Esteves 2011], categorization [Ericson 2013], and dimension reduction. In addition, Mahout uses the Apache Hadoop library to scale effectively in the cloud. Mahout's primary features are:

- Scalable machine learning libraries. The core libraries of Mahout are highly optimized for good performance and for non-distributed algorithms.
- Several MapReduce enabled clustering implementations, including k-Means [MacQueen 1967], fuzzy k-Means [Ahmed 2002] and Mean-Shift [Cheng 1995].
- Distributed Naive Bayes [Zhang 2004] implementation.
- Distributed Principal Components Analysis (PCA) [Pearson 1901] techniques for dimensionality reduction [Ye 2005].

SystemML. SystemML [Ghoting 2011b] is a system that enables the development of large scale machine learning algorithms. It first expresses a machine learning algorithm in a higher-level language called **D**eclarative **M**achine learning **L**anguage (DML). Then, it executes the algorithm in a MapReduce environment. This DML language exposes arithmetical and linear algebra primitives on matrices that are natural to express a large class of machine learning algorithms.

As shown in Figure 3.8, SystemML is organized into distinct layers:

- The Language component: It consists of user-defined algorithms written in DML.

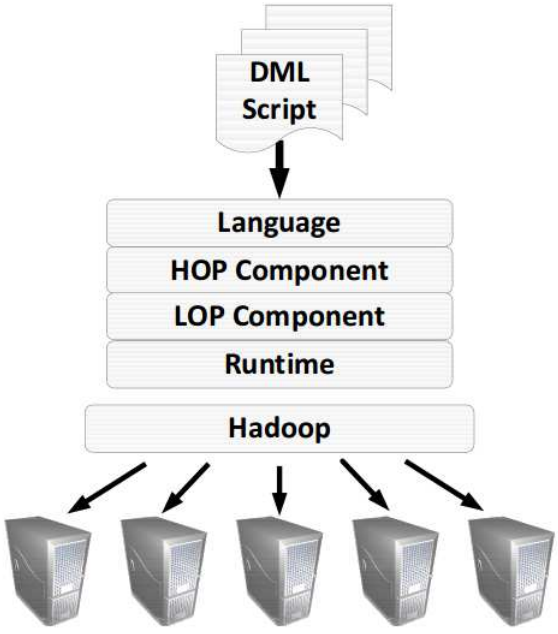


Figure 3.8: An overview of the software architecture of SystemML [Ghoting 2011b].

- The High-Level Operator Component (HOP): It analyzes all the operations within a statement block and chooses from multiple high-level execution plans. A plan is represented in a directed acyclic graph of basic operations (called hops) over matrices and scalars.

- The Low-Level Operator Component (LOP): It translates the high-level execution plans provided by the HOP component into low-level physical plans on MapReduce.

- The runtime component: It executes the low-level plans obtained from the LOP component on Hadoop.

Another attention was carried to the distribution of data mining techniques on multicore architectures [Negrevergne 2013, Laurent 2012, Qiu 2008, Négrevergne 2010]. For example, in [Laurent 2012], the authors propose an enhanced version of GRITE [Di Jorio 2009], an existing algorithm of gradual pattern

mining, on multicore processors. In [Sicard 2010], the authors propose a method for parallelizing fuzzy tree mining on multicore architectures. Gradual patterns consists of set of patterns in which an order is defined between them [Di Jorio 2010].

3.6.1.2 Summary of cloud-based data mining approaches

Table 3.7 presents the above mentioned cloud-based data mining techniques. It describes the implemented techniques, the programming languages and the used programming model of each approach.

Table 3.7: Summary of recent cloud-based data mining techniques.

Approach	Implemented techniques	Langage	Programming model
NIMBLE [Ghoting 2011a]	Clustering and itemset mining techniques	JAVA	MapReduce, OpenCL, MPI
Mahout [Foundation 2010]	Classification, clustering and itemset mining techniques	JAVA	MapReduce
SystemML [Ghoting 2011b]	Regression and ranking techniques	JAVA and DML	MapReduce

We notice that the input and the output of the above presented approaches are user-defined. The works presented in Table 3.7 consists of programming interfaces that enable the implementation of distributed data mining algorithms. Nevertheless, they do not provide implementations of distributed subgraph mining algorithms which is one of the addressed issue in this thesis.

3.6.2 Cloud-based graph mining techniques

In this section, we first present an overview of distributed graph mining algorithms. Then, we present an overview of cloud-based subgraph mining algorithms. Finally, we summarize recent cloud-based graph mining approaches with respect to the format of the input data and the output patterns.

3.6.2.1 Distributed graph mining techniques

PEGASUS. In [Kang 2009, Kang 2013], a description of PEGASUS is proposed, an open source peta graph mining library which performs typical graph mining tasks such as computing the diameter of a graph, computing the radius of each node and finding the connected components. The main idea of PEGASUS is the GIM-V primitive, standing for Generalized Iterative Matrix-Vector multiplication, which consists of a generalization of normal matrix-vector multiplication. PEGASUS customizes the GIM-V primitive and uses MapReduce in order to handle with important large scale graph mining operations.

PREGEL. In [Malewicz 2010], the authors present PREGEL, a computational model suitable for large scale graph processing. Programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. This vertex centric approach is flexible enough to express a broad set of algorithms. Implementing a large scale graph processing algorithm consists of creating a PREGEL program.

Cohen etal.'s approach. In the work of [Cohen 2009], the authors give an investigation into the feasibility of decomposing useful graph operations into a series of MapReduce processes. Such a decomposition could enable implementing graph mining algorithms on a cloud, in a streaming environment, or on a single computer.

HADI. In [Kang 2008], the authors propose HADI algorithm, a solution for mining diameter in massive graphs on the top of MapReduce. HADI has been used to analyze the largest public web graph, with billions of nodes and edges.

Another attention was carried to the computing of eigenvalue in massive graphs [Zhao 2007] and to the counting triangles in massive graphs such as the DOULION method [Tsourakakis 2009].

3.6.2.2 Distributed frequent subgraph mining techniques

With the exponential growth in both the graph size and the number of graphs in databases, several distributed solutions have been proposed for frequent subgraph

mining on a single large graphs and on massive graph databases. Here we present an overview of research works in the area of large scale subgraph mining.

MRPF. In [Liu 2009], the authors propose the MRPF algorithm for finding patterns from a complex and large network. As illustrated in Figure 3.9, the algorithm is divided into four steps: distributed storage of the graph, neighbor vertices finding and pattern initialization, pattern extension, and frequency computing.

Each step is implemented by a MapReduce pass. In each MapReduce pass, the task is divided into a number of sub-tasks of the same size and each sub-task is distributed to a node of the cluster. MRPF uses an extended mode to find the target size pattern. That is trying to add one more vertex to the matches of i-size patterns to create patterns of size $i+1$. The extension does not stop until the patterns reach the target size. The proposed algorithm is applied to prescription network to find some commonly used prescription network motifs that provide the possibility to discover the law of prescription compatibility.

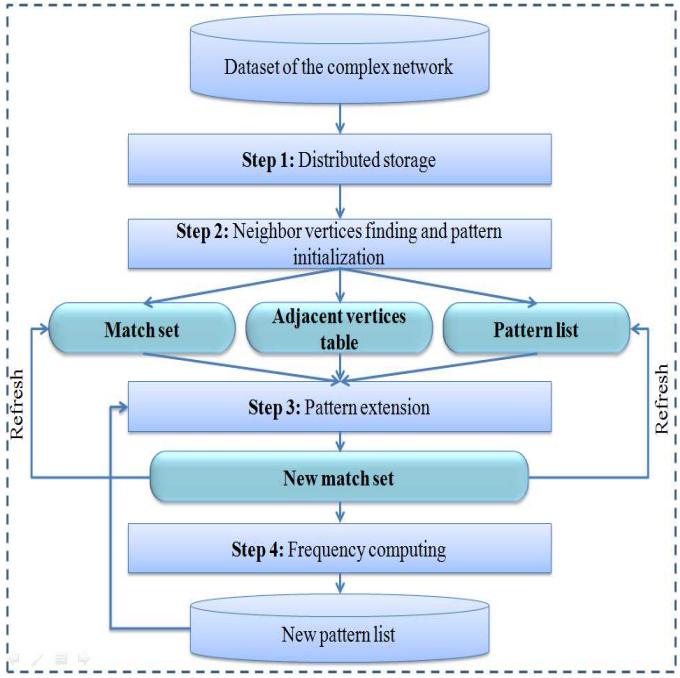


Figure 3.9: An overview of the software architecture of MRPF [Liu 2009]. Adjacent vertices table contains adjacent neighbor of each vertex of the input network. Match set and Pattern set contain the intermediate results.

Hill etal.'s approach. The work presented in [Hill 2012] presents an iterative MapReduce-based approach for frequent subgraph mining. The authors propose two heterogeneous MapReduce jobs per iteration: (1) gathering subgraphs for the construction of the next generation of subgraphs, and (2) counting these structures to remove irrelevant data.

1. **First MapReduce job** The first MapReduce job aims to construct the next generation of subgraphs. Its associated Map function sends the subgraph to the correct reducer using the graph identifier as a key. All the subgraphs of size $k-1$ with the same graph identifier are gathered for the Reduce function. Single edges in these subgraphs are used to generate the next generation of possible subgraphs of size k. The subgraph is encoded as a string. All labels alphabetized are kept and the special markers are used to designate different nodes with the same labels. The results of this step are subgraphs of size k and graph identifiers.

2. **Second MapReduce job** The second MapReduce job aims to output the frequent subgraphs. The map function of this job has the responsibility of taking in the encoded strings representing subgraphs of size-k and corresponding graph identifiers as well as outputting the subgraph as a key and the node identification numbers and graph identifiers as values. The reduce function gathers (per iteration) on label only subgraph structures. The main task of the reduce function is to compute the support of these subgraphs. The label markers are removed at this point. The outputs of iteration k are all subgraphs of size k that meet the given user defined support.

Wu etal.'s approach. In [Wu 2010], the authors propose a MapReduce-based algorithm for frequent subgraph mining. The algorithm takes a large graph as input and finds all subgraphs that match a given motif. The input large graph is represented as Personal Centre Network of every vertex in the graph [Wu 2010]. For each vertex in the graph, the algorithm calculates the candidate subgraph according to graph isomorphism algorithms. It outputs the candidate subgraphs if they are isomorphic with the motif.

Luo etal.'s approach. In [Luo 2011], the authors propose an approach to subgraph search over a graph database under the MapReduce framework. The main idea of this approach is first to build inverted edge indexes for graphs in the database, and then to retrieve data only related to the query subgraph by using the built indexes to answer the query.

Another attention was carried to the discovery and the study of dense subgraphs from massive graphs. For example, in [Bahmani 2012], an algorithm for finding the densest subgraph in a massive graph is proposed. The algorithm is based on the streaming model of MapReduce.

3.6.2.3 Summary of cloud-based graph mining approaches

Table 3.8 presents the above mentioned cloud-based graph mining techniques. It describes the input, the output of each approach and the used programming model.

As mentioned before, we focus our study on distributed FSM techniques. As one can see in Table 3.8, most distributed FSM approaches use a single large graph as input [Malewicz 2010, Tsourakakis 2009, Wu 2010, Liu 2009, Bahmani 2012]. They search frequent subgraphs in a large single graph or for subgraphs that match a given motif. Only a few works include the FSM from large graph databases. These attempts suffer from three crucial problems:

First, they try to construct the final set of frequent subgraphs iteratively using MapReduce, possibly resulting a big number of MapReduce passes and an exponential growth of intermediate data especially with large scale datasets.

Second, none of the presented distributed FSM solutions have discussed the distribution of the input data according to data characteristics. They simply use the data partitioning schema proposed by the used programming model such as MapReduce. Such partitioning may be the origin of imbalanced computational load among map tasks. This problem is known by $map - skew$ [YongChul K. 2011, Kwon 2012] and it refers to highly variable task runtimes in MapReduce applications. In most cases, skew is originating from the characteristics of the algorithm and dataset. Accordingly, it is recommended to define

Table 3.8: Summary of recent cloud-based graph mining approaches.

Approach	Input	Output	Programming model
PEGASUS [Kang 2009]	A graph database	Diameter and radius of each graph	MapReduce
Pregel [Malewicz 2010]	A single graph	Depends on the user-defined pregel program	MapReduce and Pregel
HADI [Kang 2008]	A graph database	Diameter of each graph	MapReduce
Zhao etal.'s approach [Zhao 2007]	A graph database	Eigenvalue of each graph	MPI and OpenMP
DOULION [Tsourakakis 2009]	A single graph	Triangles	MapReduce
MRPF [Liu 2009]	A single graph and a subgraph model	Frequent subgraphs	MapReduce
Hill etal.'s approach [Hill 2012]	A graph database and a subgraph model	Frequent subgraphs	MapReduce
Wu etal.'s approach [Wu 2010]	A single graph and a subgraph model	Frequent subgraphs	MapReduce
Luo etal.'s approach [Luo 2011]	A graph database	Frequent subgraphs	MapReduce
Bahmani etal.'s approach [Bahmani 2012]	A single graph	Densest subgraphs	MapReduce

a preprocessing step that distributes the data according to data characteristics which constitutes one of the best practices to alleviate skew problems in the map phase [YongChul K. 2011]. This preprocessing step consists of extracting properties of the input data and appropriately partitioning the data before starting the computations.

Third, the above presented solutions do not include the monetary aspect. It is necessary to design cost models for managing and mining subgraph patterns in a cloud architecture. Such cost models consider the needs of customers such as budget limit, response time limit and quality of the result limit.

3.7 Conclusion

In this chapter, we have introduced the concept of cloud computing. We have presented its associated models, namely deployment models, business models and programming models. We have also presented some popular cloud-based data mining and graph mining techniques. We tried to categorize distributed data mining and graph mining approaches according to the input data, the output patterns and the used programming model. The raised issue is that few works have addressed the problem of distributed frequent subgraph mining in the cloud which constitutes the addressed problem of this thesis. In addition, the data distribution technique adopted by these works does not include data characteristics. To overcome this obstacle, a data partitioning technique that considers data characteristics should be applied.

In the next part of the thesis, we will present our proposed approach (see Chapter 4), a scalable and distributed approach for large scale frequent subgraph mining based on MapReduce. The proposed approach provides a density-based partitioning technique of the input data. Such a partitioning technique allows a balanced computational loads over the distributed collection of machines. We will also present in Chapter 5 new cost models for managing and mining subgraph patterns in a cloud setting. We notice that our contributions are located in the Platform as a Service (PaaS) layer of the cloud computing service model layers.

Key points

- We presented the field of cloud computing and its benefits. We give an overview of its associated models, namely business model, deployment models, and programming models.
- We studied existing works on the field of distributed data mining approaches in the cloud.
- We focused on cloud-based subgraph mining approaches and we presented a summary of most popular ones.
- We highlighted the raised issues with existing distributed subgraph mining techniques.
- We concluded this chapter by mentioning the need not only to propose a large scale cloud-based subgraph mining technique, but also to include data characteristics in the partitioning step. Also, we mentioned the need to define cost models that consider the needs of customers such as budget limit, response time limit and quality of the result limit.

Part II

Contributions

Chapter 4

Density-based partitioning for frequent subgraph mining in the cloud

Contents

Goals

Recently, graph mining approaches have become very popular, especially in certain domains such as bioinformatics, chemoinformatics and social networks. As mentioned in Chapter 1 and 2, frequent subgraph discovery is one of the most challenging tasks. This task has been highly motivated by the tremendously increasing size of existing graph databases. Due to this fact, there is urgent need for efficient and scaling approaches for frequent subgraph discovery. In this chapter, we propose a novel approach for distributed large scale subgraph mining in the cloud. The proposed approach allows the use of different subgraph mining algorithms and different partitioning methods. We present an experimental study and we show that our approach decreases significantly the execution time and scales the subgraph discovery process to large graph databases.

4.1 Formulation of the distributed subgraph mining problem

In this work, we are interested in frequent subgraph mining in large scale graph databases.

Let $DB = \{G_1, \ldots, G_K\}$ be a large scale graph database with K graphs, $SM = \{M_1, \ldots, M_N\}$ a set of distributed machines, $\theta \in [0,1]$ is a minimum support threshold. For $1 \leq j \leq N$, let $Part_j(DB) \subseteq DB$ be a non-empty subset of DB. We define a partitioning of the database over SM by the following: $Part(DB) = \{Part_1(DB), \ldots, Part_N(DB)\}$ such that

- $\bigcup_{i=1}^{N} \{Part_i(DB)\} = DB, and$
- $\forall i \neq j, Part_i(DB) \cap Part_j(DB) = \emptyset$.

In the context of distributed frequent subgraph mining, we propose the following definitions.

Definition 20 (Globally frequent subgraph) *For a given minimum support threshold $\theta \in [0,1]$, G' is* globally frequent subgraph *if $Support(G', DB) \geq \theta$. Here, θ is called global minimum support threshold (GS).*

Definition 21 (Locally frequent subgraph) *For a given minimum support threshold $\theta \in [0,1]$ and a tolerance rate $\tau \in [0,1]$, G' is* locally frequent subgraph *at site i if $Support(G', Part_i(DB)) \geq ((1-\tau) \cdot \theta)$. Here, $((1-\tau) \cdot \theta)$ is called local minimum support threshold (LS).*

Definition 22 (Loss Rate) *Given S_1 and S_2 two sets of subgraphs with $S_2 \subseteq S_1$ and $S_1 \neq \emptyset$, we define the loss rate in S_2 compared to S_1 by*

$$LossRate(S_1, S_2) = \frac{\mid S_1 - S_2 \mid}{\mid S_1 \mid}, \tag{4.1}$$

We define the problem of distributed subgraph mining by finding a good partitioning of the database over the set of distributed machines (SM) and by minimizing a defined approximation of the complete set of frequent subgraphs extracted from the DB database ($SG(DB, \theta)$).

Definition 23 (an ε-approximation of a set of subgraphs) *Given a parameter $\varepsilon \in [0,1]$ and $SG(DB,\theta)$. An ε-approximation of $SG(DB,\theta)$ is a subset $S \subseteq SG(DB,\theta)$ such that*

$$LossRate(SG,S) \leq \varepsilon. \tag{4.2}$$

We measure the cost of computing an ε-approximation of $SG(DB,\theta)$ with a given partitioning method $PM(DB)$ by the standard deviation of the set of runtime values in mapper machines.

Definition 24 (Cost of a partitioning method) *Let $R = \{Runtime_1(PM), \ldots, Runtime_N(PM)\}$ be a set of runtime values. $Runtime_j(PM)$ represents the runtime of computing frequent subgraphs in the partition j ($Part_j$) of the database. The operator E denotes the average or expected value of R. Let μ be the mean value of R:*

$$\mu = E[R]. \tag{4.3}$$

The cost measure of a partitioning technique is:

$$Cost(PM) = \sqrt{E[(R-\mu)^2]}. \tag{4.4}$$

A large cost value indicates that the runtime values are far from the mean value and thus imbalanced computational load among the distributed machines. A small cost value indicates that the runtime values are near the mean value and thus balanced computational load. The smaller the value of the cost is, the more efficient the partitioning is.

4.2 Density-based partitioning for large scale subgraph mining with MapReduce

In this section, we present our approach for large scale subgraph mining with MapReduce. It first describes the proposed framework to approximate large scale

frequent subgraph mining [Aridhi 2013]. Then, it presents our density-based partitioning technique.

4.2.1 A MapReduce-based framework to approximate large scale frequent subgraph mining

In this section, we present the proposed framework for large scale subgraph mining with MapReduce (see Figure 4.1).

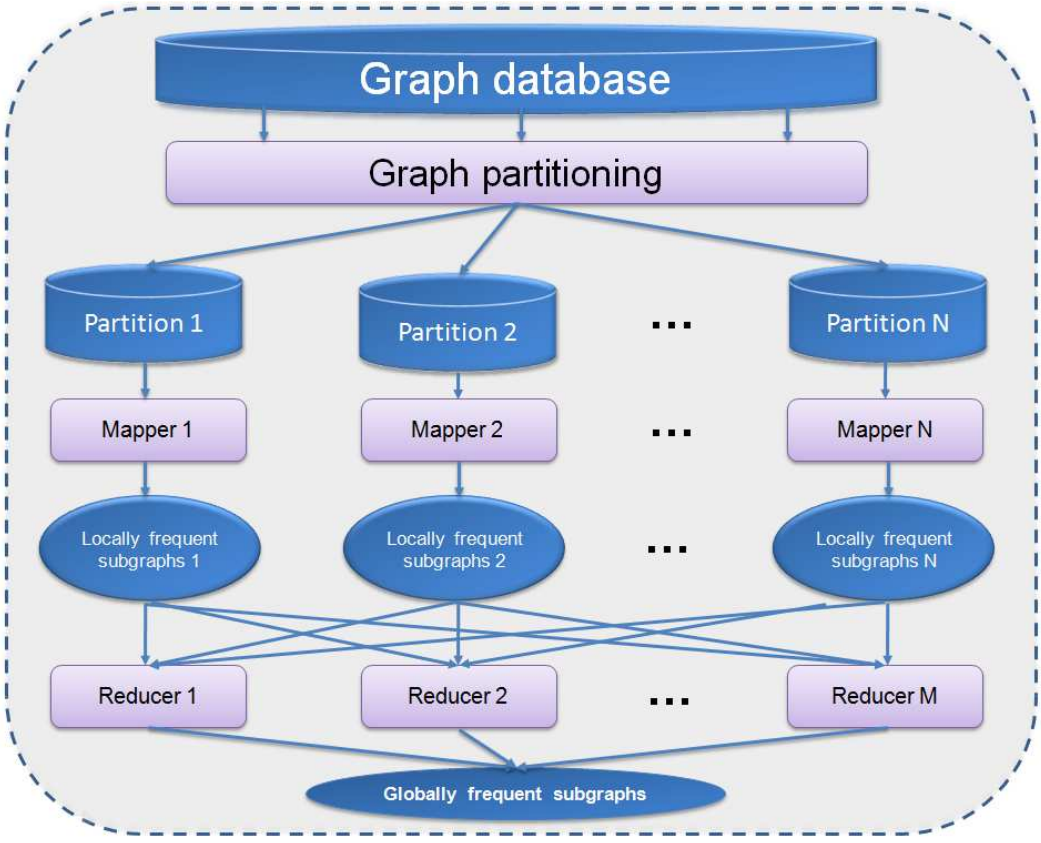


Figure 4.1: A system overview of our approach. In this figure, ellipses and cylinders represent data, squares represent computations on the data and arrows show the flow of data.

As shown in Figure 4.1, our method works as follows:

1. Input graph database is partitioned into N partitions. Each partition will be processed by a mapper machine.

2. Mapper i reads the assigned data partition and generates the corresponding locally frequent subgraphs according to a local support. Mapper i outputs $\langle key, value \rangle$ pairs of locally frequent subgraphs $\langle subgraph, Support(subgraph, Part_i(DB)) \rangle$.

3. For each unique intermediate key, the reducer passes the key and the corresponding set of intermediate values to the defined $Reduce$ function. According to these $\langle key, value \rangle$ pairs, the reducer outputs the final list of $\langle key, value \rangle$ pairs after filtering according to the global support $\langle subgraph, Support(subgraph, DB) \rangle$.

In the following sections, we give a detailed description of our approach.

4.2.1.1 Data partitioning

In the data partitioning step, the input database is partitioned into a user-specified number of partitions N. The input of this step is a graph database $DB = \{G_1, \ldots, G_K\}$ and the output is a set of partitions $Part(DB) = \{Part_1(DB), \ldots, Part_N(DB)\}$. Our framework allows many partitioning techniques for the graph database. In our work, we consider two instances of data partitioning. The first partitioning method is the default one proposed by MapReduce framework that we called MRGP (which stands for **M**apReduce **G**raph **P**artitioning). It arbitrarly constructs the final set of partitions according to chunk size. Though, MRGP does not consider the characteristics of the input data during partitioning. Besides the standard MRGP partitioning, we propose a partitioning technique that takes into account the characteristics of the input data during the creation of partitions. We termed it **D**ensity-based **G**raph **P**artitioning (shortly called DGP). More precisely, DGP tends to balance graph density distribution in each partition (for more details, see Section 4.2.2).

4.2.1.2 Distributed subgraph mining

The distributed subgraph mining step consists of mining the set of globally frequent subgraphs. The input of this step is a set of partitions $Part(DB) = \{Part_1(DB), \ldots, Part_N(DB)\}$ and the output is the set of globally frequent subgraphs. The execution of the distributed subgraph mining step is resumed by a MapReduce pass. In the Map step, we use a frequent subgraph mining technique that we run on each partition in parallel. In the *Reduce* step, we compute the final set of globally frequent subgraphs. The Algorithms 1 and 2 present our Map and *Reduce* functions:

Algorithm 1 Map function.

Require: A partitioned graph database $DB = \{Part_1(DB), \ldots, Part_N(DB)\}$, minimum support threshold θ, tolerance rate τ, key $= i$, value= graph partition $Part_i(DB)$
Ensure: Locally frequent subgraphs in $Part_i(DB)$
1: $S_i \leftarrow FSMLocal(Part_i(DB), \theta, \tau)$
2: **for all** s in S_i **do**
3: $\quad EmitIntermediate(s, Support(s, Part_i(DB)))$
4: **end for**

Algorithm 2 Reduce function.

Require: Minimum support threshold θ, key=a subgraph s, values=local supports of s
Ensure: Globally frequent subgraphs in DB
1: $GlobalSupportCount \leftarrow 0$
2: **for all** v in $values$ **do**
3: $\quad GlobalSupportCount \leftarrow GlobalSupportCount + v$
4: **end for**
5: $GlobalSupport \leftarrow \frac{GlobalSupportCount}{N}$
6: **if** $GlobalSupport >= \theta$ **then**
7: $\quad Emit(s, GlobalSupport)$
8: **end if**

In the Map function, the input pair would be like $\langle key, Part_i(DB) \rangle$ where $Part_i(DB)$ is the graph partition number i. The $FSMLocal$ function applies the

subgraph mining algorithm to $Part_i(DB)$ with a tolerance rate value and produces a set S_i of locally frequent subgraphs. Each mapper outputs pairs like $\langle s, Support(s, Part_i(DB))\rangle$ where s is a subgraph of S_i and $Support(s, Part_i(DB))$ is the local support of s in $Part_i$.

The $Reduce$ function receives a set of pairs $\langle s, Support(s, Part_i(DB))\rangle$ and computes for each key (a subgraph), the global support $GlobalSupport$. Only globally frequent subgraphs will be kept.

4.2.1.3 Analysis

The output of our approach is an ε-approximation of the exact solution $SG(DB, \theta)$. Algorithms 1 and 2 do not offer a complete result since there are frequent subgraphs that cannot be extracted. The decrease in the number of ignored frequent subgraphs can be addressed by a good choice of tolerance rate for the extraction of locally frequent subgraphs. Theoretically, we can achieve the exact solution with our approach (which refers to $LossRate(S, SG) = 0$) by adjusting the tolerance rate parameter to $\tau = 1$ which mean a zero value of ε ($\varepsilon = 0$). This means that the set of locally frequent subgraphs contains all possible subgraphs (Local support equal to zero $LS = 0$) and the set of globally frequent subgraphs contains the same set as $SG(DB, \theta)$. In this case, the value of the loss rate is zero. However, the generation of the exact solution can cause an increase of the running time.

In the distributed subgraph mining process of our approach, we perceive the following lemma:

Lemma 1 *If a subgraph G' is globally frequent then G' is locally frequent in at least one partition of the database.*

Proof 1 *Let $DB = \{G_1, \cdots, G_K\}$ be a graph database with K graphs, let G' be a globally frequent subgraph of DB, let $\theta \in [0, 1]$ be a minimum support threshold, let $Part(DB) = \{Part_1(DB), \cdots, Part_N(DB)\}$ be a partitioning of the database and let $\tau \in [0, 1]$ be a tolerance rate, let LS be the local support in the different partitions, let s_i be the support of G' in $Part_i(DB)$ and let s be the support of G' in DB.*

Assume that G' is not locally frequent in all the partitions $Part_i(DB)$ then we have $s_i \leq LS$, for all $i \in [1,N]$. Thus, $\frac{\sum_{i=1}^{N} s_i}{N} \leq \frac{\sum_{i=1}^{N} LS}{N}$ and therefore, $s \leq LS$. We have $LS = (1-\tau) \cdot \theta$ and therefore $LS \leq \theta$, for all $\tau \in [0,1]$. Thus, we have $s \leq \theta$ and so G' is not globally frequent, contradicting our assumption.

4.2.2 The density-based graph partitioning method

4.2.2.1 Motivation and principle

The motivation behind dividing the input data into partitions is to effectively reduce the computation space by dealing with smaller graph databases that need to be processed in parallel. However, we need to combine intermediate results to get the overall one. Using this approach, we can decrease the subgraph mining complexity, knowing that the time complexity of the subgraph mining process is proportional to the size of the input data. However, this data decomposition is the origin of a loss of the global vision in terms of support computing. In addition, the arbitrary partitioning method of MapReduce that we called MRGP (which stands for **M**apReduce **G**raph **P**artitioning) may be the origin of *map-skew* which refers to imbalanced computational load among map tasks [Kwon 2012].

Considering the fact that the task of frequent subgraph mining depends on the density of graphs [Wörlein 2005, Huan 2003b], we propose a density-based partitioning method that we called DGP (which stands for **D**ensity-based **G**raph **P**artitioning) which consists of constructing partitions (chunks) according to the density of graphs in the database. The goal behind this partitioning is to ensure load balancing and to limit the impact of parallelism and the bias of the tolerance rate. Figure 4.2 gives an overview of the proposed partitioning method.

Our partitioning technique consists of two levels: (1) dividing the graph database into B buckets and (2) constructing the final list of partitions.

The first level of our partitioning method consists of two steps: graph densities computing and density-based decomposition. The graph densities computing step is performed by a MapReduce pass that we called $DensitiesComputing$. This MapReduce pass computes the densities of all instances in the database. Algorithm 3 presents the Map function of this step.

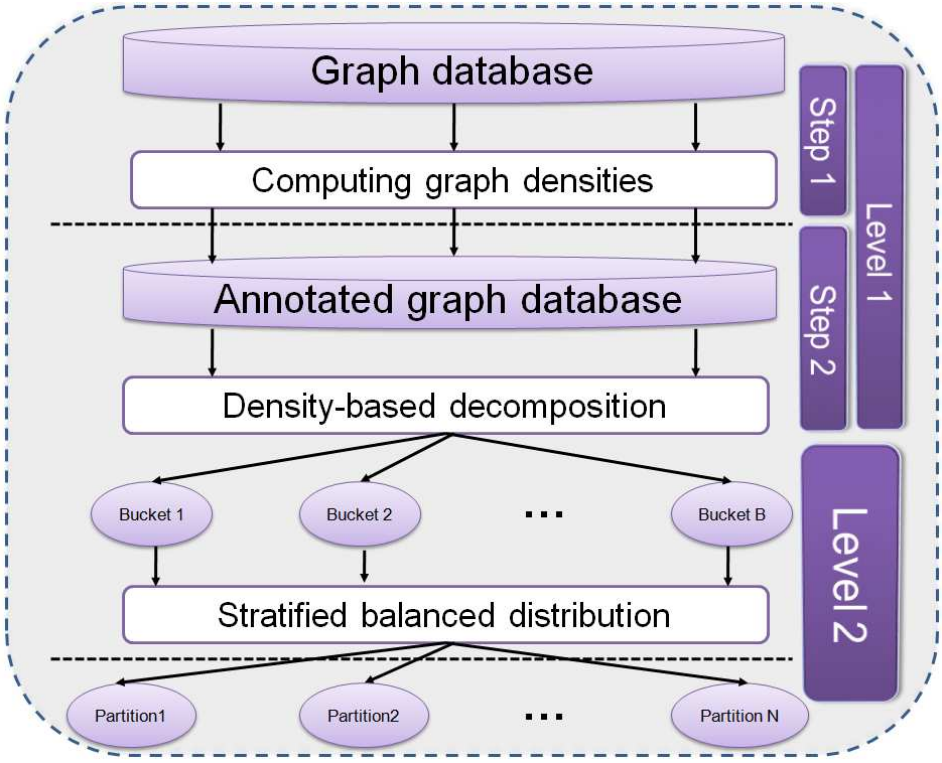


Figure 4.2: The DGP method. In this figure, ellipses and cylinders represent data, squares represent computations on the data and arrows show the flow of data.

Algorithm 3 Map function of $DensitiesComputing$.

Require: A graph database $DB = \{G_1, \ldots, G_K\}$
Ensure: Annotated and sorted graph database $ADB = \{\{d_1, G_1\}, \ldots, \{d_K, G_K\}\}$

1: **for all** G_i in DB **do**
2: $d_i \leftarrow density(G_i)$
3: $EmitIntermediate(d_i, G_i)$
4: **end for**

The *Reduce* function is the identity function which output a sorted list of graphs according to the densities values. In fact, the sorting of graphs is done automatically by the *Sort* function of MapReduce since we used the density value as a key (see Algorithm 3). In the second step, a density-based decomposition is applied which divides the sorted graph database into B buckets. The output

buckets contain the same number of graphs.

The second level of our partitioning method is to construct the output partitions. To do this, we first divide each bucket into N sub-partitions $B_i = \{P_{i1}, \cdots, P_{iN}\}$. We then construct each output partition by appending one sub-partition from each bucket. Each output partition contains one sub-partition from each bucket. Thus, each partition has almost the portion of dense graphs and sparse graphs.

4.2.2.2 Illustrative example

In this section, we give an illustrative example to explain the principle of the two partitioning techniques. Given a graph database of 12 graphs $DB = \{G_1, \cdots, G_{12}\}$. Table 4.1 presents the size on disk and the density of each graph in the database.

Table 4.1: Graph database example.

Graph	Size (KB)	Density
G_1	1	0.25
G_2	2	0.5
G_3	2	0.6
G_4	1	0.25
G_5	2	0.5
G_6	2	0.5
G_7	2	0.5
G_8	2	0.6
G_9	2	0.6
G_{10}	2	0.7
G_{11}	3	0.7
G_{12}	3	0.8

Considering that we are running our example in a four nodes cloud environment. Using the MRGP method, the graph database will be divided into four partitions of six KB each:

$$Part(DB) = \{\{G_1, G_2, G_3, G_4\}, \{G_5, G_6, G_7\}, \{G_8, G_9, G_{10}\}, \{G_{11}, G_{12}\}\}.$$

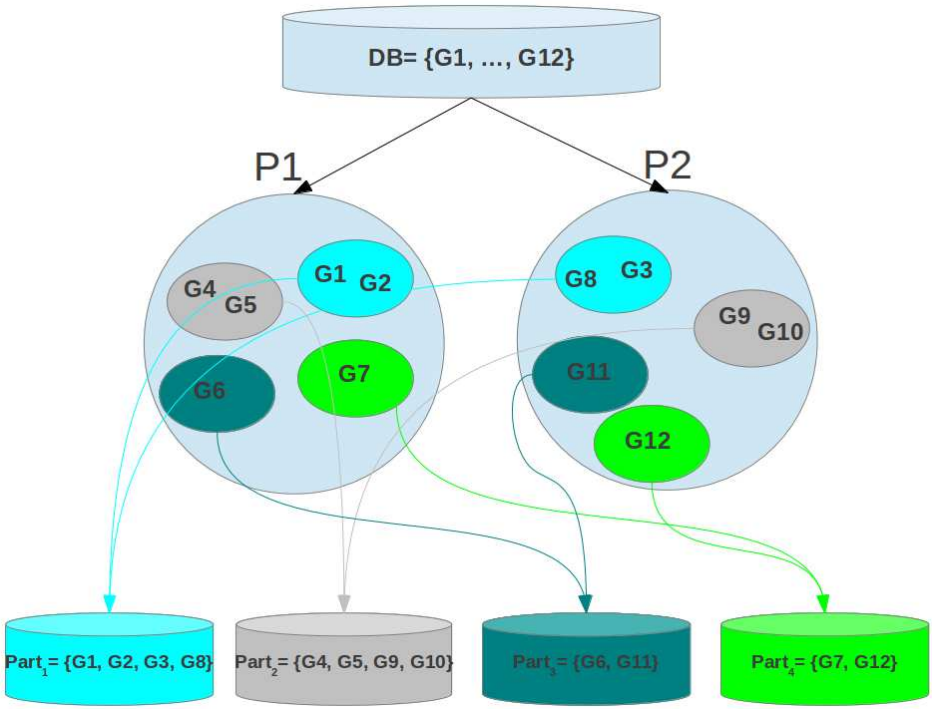


Figure 4.3: Example of DGP method.

Using the DGP method with two buckets, we first compute graph densities and we sort the database according to graph densities. Then, we divide the graph database into two buckets $B_1 = \{G_1, G_2, G_4, G_5, G_6, G_7\}$ and $B_2 = \{G_3, G_8, G_9, G_{10}, G_{11}, G_{12}\}$. Bucket B_1 contains the first six graphs and bucket B_2 contains the last six graphs in the database. Finally, we construct the four graph partitions from B_1 and B_2 (see Figure 4.3).

As shown in Figure 4.3, each partition contains a balanced set of graphs in terms of density. Each partition contains the same portion of dense graphs and sparse graphs from B_1 and B_2. The final set of partitions will be:

$$Part(DB) = \{\{G_1, G_2, G_3, G_8\}, \{G_4, G_5, G_9, G_{10}\}, \{G_6, G_{11}\}, \{G_7, G_{12}\}\}.$$

Our method is sound and correct, and will always produce a balanced set of partitions in terms of density.

4.3 Experiments

This section presents an experimental study of our approach on synthetic and real datasets. It first describes the used datasets and the implementation details. Then, it presents a discussion of the obtained results.

4.3.1 Experimental setup

4.3.1.1 Datasets

The datasets used in our experimental study are described in Table 4.2.

Table 4.2: Experimental data.

Dataset	Type	Number of graphs	Size on disk	Average size
DS1	Synthetic	20,000	18 MB	[50-100]
DS2	Synthetic	100,000	81 MB	[50-70]
DS3	Real	274,860	97 MB	[40-50]
DS4	Synthetic	500,000	402 MB	[60-70]
DS5	Synthetic	1,500,000	1.2 GB	[60-70]
DS6	Synthetic	100,000,000	69 GB	[20-100]

The datasets described in Table 4.2 are composed of synthetic and real ones. The synthetic datasets are generated by the synthetic data generator *GraphGen* provided by Kuramochi and Karypis [Kuramochi 2001]. We generate various synthetic datasets (DS1, DS2, DS4, DS5 and DS6) according to different parameters such as: the number of graphs in the dataset, the average size of graphs in terms of edges and the size on disk. Varying datasets allows us to avoid specific outcomes to data and to have better interpretations.

The real dataset (DS3) we tested is a chemical compound dataset which is available from the Developmental Therapeutics Program (DTP) at National Cancer Institute (NCI)[1].

[1] http://dtp.nci.nih.gov/branches/npb/repository.html

4.3.1.2 Implementation platform

We implemented our approach in Perl language and we used Hadoop, (Hadoop 0.20.1 release), an open source version of MapReduce. The databases files are stored in the Hadoop Distributed File System (HDFS), an open source implementation of GFS [Ghemawat 2003].

All the experiments of our approach were carried out using a virtual cluster of five virtual machines. The processing nodes used in our tests are equipped with a Quad-Core AMD Opteron(TM) processor 6234 2.40 GHz CPU and 4 GB of memory for each node.

4.3.1.3 Experimental protocol

The experimental evaluation of our approach focuses on three main aspects.

First, we tested the result quality of our approach in terms of number of generated subgraphs. To do this, we first compared the proposed partitioning method DGP with MRGP, the default partitioning method of MapReduce. Then, we compared our method with the random sampling method in terms of loss rate and number of false positives. The results of the sampling method are obtained after performing a sampling of the input data. For each dataset, we conducted a sampling and we generated several subsamples. We calculated the loss rate for each subsample and their average value. The loss rate is calculated for each subsample using all k subgraphs generated from the entire sample and the top-k subgraphs generated from the subsample.

Second, we conducted a performance evaluation of our approach in terms of execution time and scalability. We tested the ability of the DGP method to balance the computational load over the used distributed machines. To do this, we compared DGP with MRGP in terms of execution time. In addition, we studied the impact of the number of machines used in the process of distributed frequent subgraph mining with both methods, DGP and MRGP.

Third, we studied the impact of some MapReduce parameters on the performance of our approach. In this context, we tested the impact of the block size and the number of copies of data (replication factor) on the execution time of our

approach.

4.3.2 Experimental results

4.3.2.1 Result quality

Table 4.3 shows the obtained results using the sequential version of the used subgraph extractors.

Table 4.3: Experimental results of classic subgraph extractors.

Dataset	Support θ (%)	gSpan		FSG		Gaston	
		Number of subgraphs	Runtime (s)	Number of subgraphs	Runtime (s)	Number of subgraphs	Runtime (s)
DS1	30	372	31	352	9	352	5
	50	41	6	23	5	23	7
DS2	30	156	171	136	235	136	17
	50	26	9	9	165	9	4
DS3	30	98	138	93	111	93	17
	50	38	106	35	61	35	9

For each dataset and support value, we note the results of the classic subgraph mining algorithm and those of the proposed method. We indicate that the sets of frequent subgraphs generated by the sequential implementations of the three algorithms are not the same. In fact, gSpan generates all frequent subgraphs including the ones formed by a single node, while the used FSG and Gaston implementations generate all frequent subgraphs without considering the ones formed by a single node. Thus, the set of frequent subgraphs generated by FSG or Gaston is a subset of the set of frquent subgraphs generated by gSpan.

Table 4.4 shows the obtained results using our proposed approach with the default MapReduce partitioning technique and those obtained with the density-based partitioning technique with two buckets.

We mention that we could not conduct our experiment with the sequential algorithms in the case of $DS4$, $DS5$ and $DS6$ due to the lack of memory. However, with the distributed algorithm we were able to handle those datasets. We notice that the number of subgraphs generated by the distributed solution is, in general, smaller than the number generated by the sequential version of the algorithm. This is related to the application of subgraph mining process on each partition separately

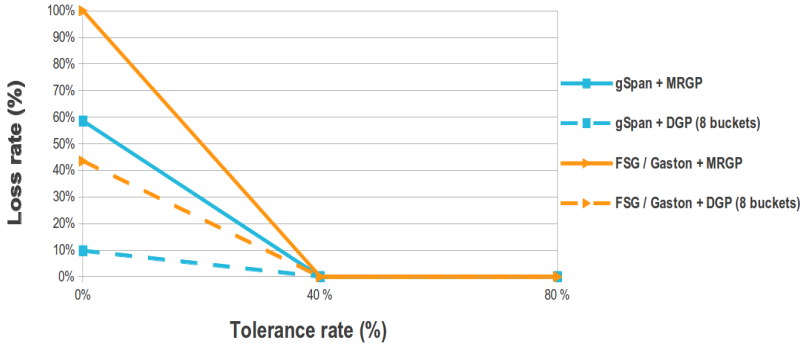
Table 4.4: Experimental results of the proposed approach.

Dataset	Support θ (%)	Tolerance rate τ	Number of subgraphs					
			MRGP			DGP		
			gSpan	FSG	Gaston	gSpan	FSG	Gaston
DS1	30	0	82	61	61	198	179	179
		0.3	227	207	207	364	344	344
		0.6	312	352	352	371	351	351
	50	0	17	0	0	23	6	6
		0.3	41	23	23	41	23	23
		0.6	41	23	23	41	23	23
DS2	30	0	145	124	124	146	125	125
		0.3	156	136	136	156	136	136
		0.6	156	136	136	156	136	136
	50	0	25	7	7	25	7	7
		0.3	26	9	9	26	9	9
		0.6	26	9	9	26	9	9
DS3	30	0	77	70	70	80	77	77
		0.3	97	92	92	88	93	93
		0.6	97	93	93	97	93	93
	50	0	36	31	31	37	32	32
		0.4	38	35	35	38	35	35
		0.6	38	35	35	38	35	35
DS4	30	0	137	116	116	78	117	117
		0.3	155	135	135	78	135	135
		0.6	155	135	135	155	135	135
	50	0	24	6	6	24	6	6
		0.3	26	9	9	26	9	9
		0.6	26	9	9	26	9	9
DS5	30	0	131	121	121	104	118	118
		0.3	155	135	135	104	135	135
		0.6	155	135	135	155	135	135
	50	0	24	7	7	18	6	6
		0.3	26	9	9	18	9	9
		0.6	26	9	9	18	9	9
DS6	30	0	66	0	0	104	3	3
		0.3	66	0	0	104	3	3
		0.6	66	0	0	104	3	3
	50	0	4	0	0	17	0	0
		0.3	4	0	0	17	0	0
		0.6	4	0	0	17	0	0

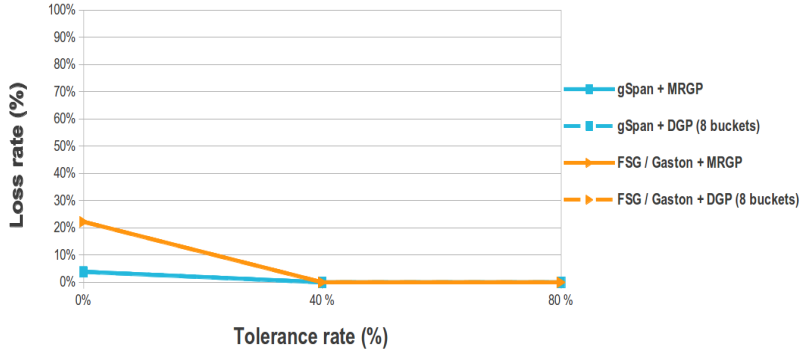
with a local support. Similarly, in the reduce phase, we ignore subgraphs which are frequent in the whole dataset but infrequent in the partitions. This loss can be decreased by the use of a maximal value of tolerance rate, i.e., which means a minimal value of local support (see Table 4.4). For example, in Table 4.4, for $DS1$ and with $\theta = 0.3$, we generate 372 subgraphs with the sequential algorithm gSpan, but we just generate 198 subgraphs with the distributed solution (with the density-based graph partitioning and a tolerance rate $\tau = 0$). By increasing the tolerance rate to $\tau = 0.6$, we restore 173 of previously ignored subgraphs and we practically reach the number of subgraphs generated by the sequential algorithm.

As shown in Table 4.4, the density-based partitioning method allows a decreasing number of lost subgraphs compared to the default MapReduce partitioning

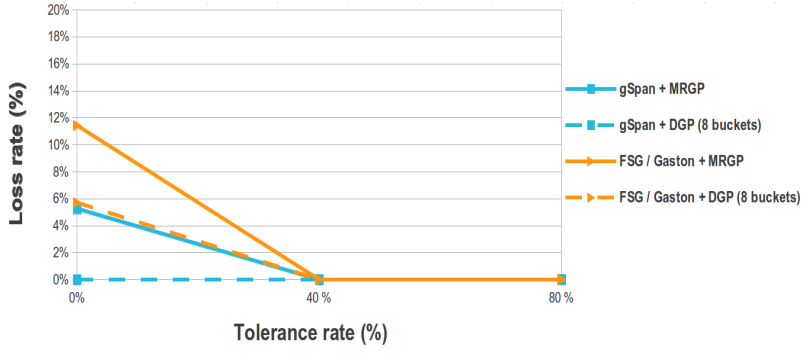
method, in almost all cases. We illustrate in Figure 4.4 the effect of the proposed partitioning methods on the rate of lost subgraphs.



(a) DS1



(b) DS2



(c) DS3

Figure 4.4: Effect of the partitioning method on the rate of lost subgraphs.

We can easily see in Figure 4.4 that the density-based graph partitioning allows low values of loss rate especially with low values of tolerance rate. We also notice

that FSG and Gaston present a higher loss rate than gSpan in almost all cases.

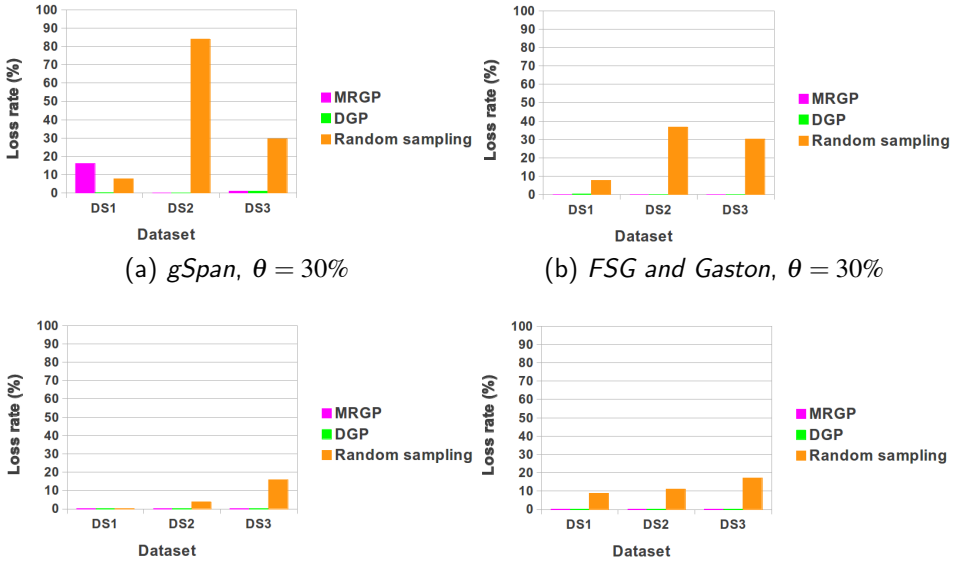
We recall that our distributed frequent subgraph mining method is an approximation method. In order to compare our method with other approximation methods, we present in the Table 4.5, the number of subgraphs obtained with the sampling method and the number of false positives that correspond to frequent subgraphs in the subsample but infrequent in the overall sample. The numbers of subgraphs presented in the table are the average values of the number of subgraphs generated from different subsamples constructed for each data sample.

Table 4.5: Number of false positives of the sampling method.

Dataset	Support θ (%)	gSpan		FSG		Gaston	
		Number of subgraphs	Number of false positives	Number of subgraphs	Number of false positives	Number of subgraphs	Number of false positives
DS1	30	4421	4078	4401	4078	4401	4078
	50	194	155	174	153	174	153
DS2	30	164	139	144	58	144	58
	50	29	4	12	4	12	4
DS3	30	264	195	258	193	258	193
	50	62	30	59	30	59	30

As shown in Table 4.5, the sampling method has a major problem. In fact, it leads to a high number of false positives in contrast to our method which generates a set of globally frequent subgraphs over the overall sample data (see Section 4.2.1.3). We present in Figure 4.5, a comparison of our method with random sampling method in terms of loss rate. The presented loss rate values of the sampling method correspond to the average values of the loss rate of the various subsamples of each sample.

As shown in Figure 4.5, the random sampling technique leads, in almost all cases, to a loss in the quality of the obtained results compared to MRGP and DGP. This loss is expressed in Figure 4.5 by high values of loss rate compared to those obtained with the MRGP and DGP methods. This can be explained by the non-representative subsamples of data generated by the random sampling method. We note that the random sampling method allows, in some cases, low loss rates values compared to the MRGP method (see Figure 4.5a). However, these loss rates values are always accompanied by high values of false positives (see Table 4.5).



(a) *gSpan*, $\theta = 30\%$ (b) *FSG and Gaston*, $\theta = 30\%$

(c) *gSpan*, $\theta = 50\%$ (d) *FSG and Gaston*, $\theta = 50\%$

Figure 4.5: Comparison of our method with the random sampling method.

4.3.2.2 Speedup

Figure 4.6 shows the effect of the density-based partitioning method on the distribution of workload across the used worker nodes in comparison with the default MapReduce partitioning method. Figure 4.7 shows the effect of the number of buckets in the density-based partitioning method on the distribution of workload across the used worker nodes.

As illustrated in figures 4.6 and 4.7, the density-based partitioning method allows a balanced distribution of workload across the distributed worker nodes especially with high number of buckets. We note also that the use of the proposed density-based partitioning method significantly improves the performance of our approach. This improvement is expressed by the decrease in the runtime in comparison with results given by the default MapReduce partitioning method. This result can be explained by the fact that each partition of the database contains a balanced set of graphs in term of density. Consequently, this balanced distribution of the data provides an effective load balancing scheme for distributed computations over worker nodes.

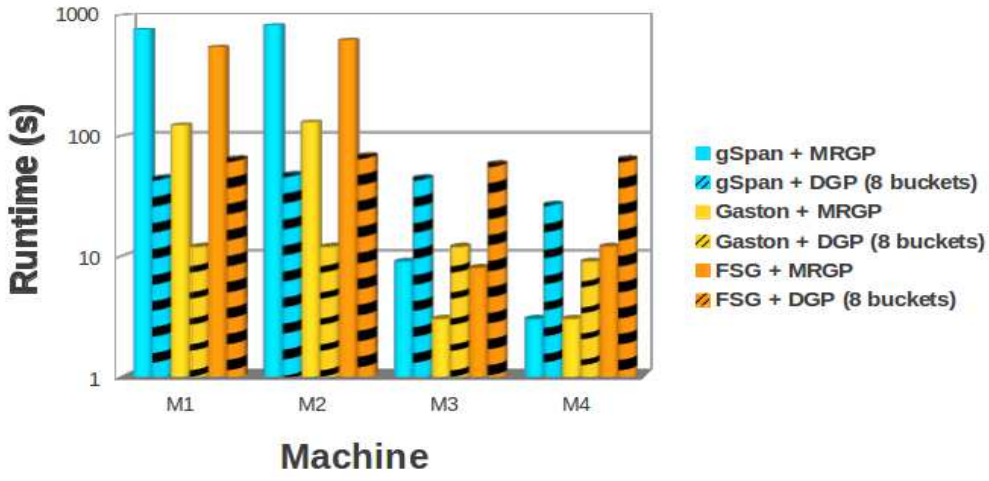


Figure 4.6: Effect of the partitioning method on the distribution of computations. We used $\theta = 30\%$ and $\tau = 0.3$.

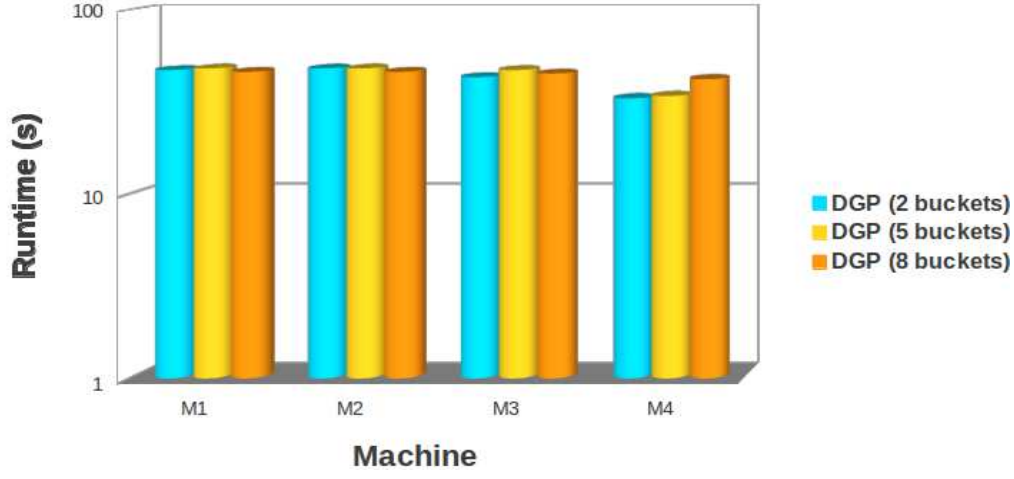


Figure 4.7: Effect of the number of buckets of the density-based partitioning method on the distribution of computations. We used gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

In order to evaluate the capability of the density-based partitioning method to balance the computations over the used nodes, we show in Figure 4.8 the cost of this partitioning method in comparison with the MapReduce-based partitioning method. In addition, we show in Figure 4.9 the effect of the number of buckets in the density-based partitioning method on the cost of the partitioning method.

For each partitioning method and for each dataset, we present the mean value of the set of runtime values in the used set of machines and the cost bar which corresponds to the error bar. This cost bar gives a general idea of how accurate the partitioning method is.

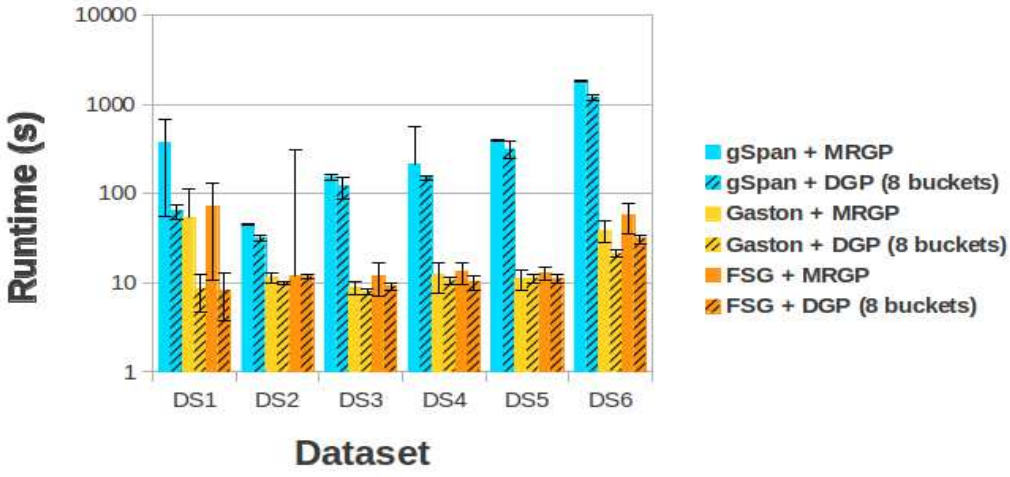


Figure 4.8: Cost of partitioning methods. We used $\theta = 30\%$ and $\tau = 0.3$.

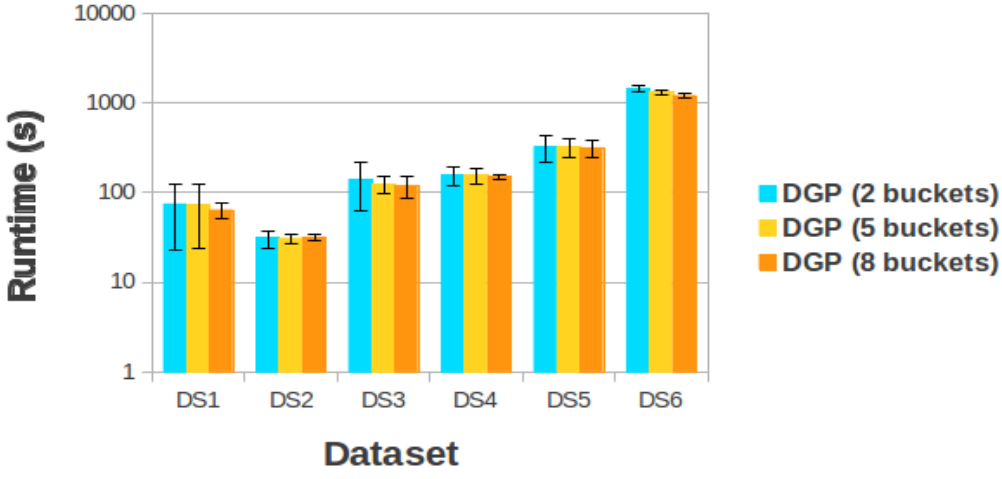


Figure 4.9: Effect of the number of buckets on the cost of the density-based partitioning method. We used gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

As shown in figures 4.8 and 4.9, the density-based partitioning method allows

minimal cost values in almost all datasets and all thresholds setting especially with high numbers of buckets. This can be explained by the balanced distribution of graphs in the partitions and thus by the balanced workload insured by a high number of buckets. It is also clear that FSG and Gaston present a shorter runtime than gSpan (see Figure 4.8).

In order to study the scalability of our approach and to show the impact of the number of used machines on the large scale subgraph mining runtime, we present in Figure 4.10 the runtime of our approach for each number of mapper machines.

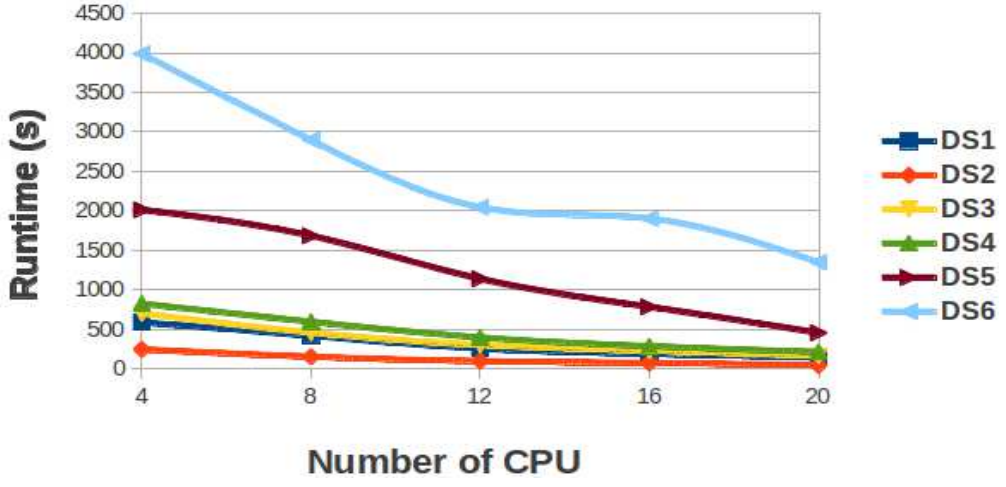


Figure 4.10: Effect of the number of workers on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

As illustrated in Figure 4.10, our approach scales with the number of machines. In fact, the execution time of our approach is proportional to the number of nodes or machines.

4.3.2.3 Chunk size and replication factor

In order to evaluate the influence of some MapReduce parameters on the performance of our implementation, we conducted two types of experiments. Firstly, we varied the block size and calculated the runtime of the distributed subgraph mining process of our system. In this experiment, we used five datasets and varied

the chunk size from 10MB to 100MB. Secondly, we varied the number of copies of data and calculated the runtime of the distributed subgraph mining process.

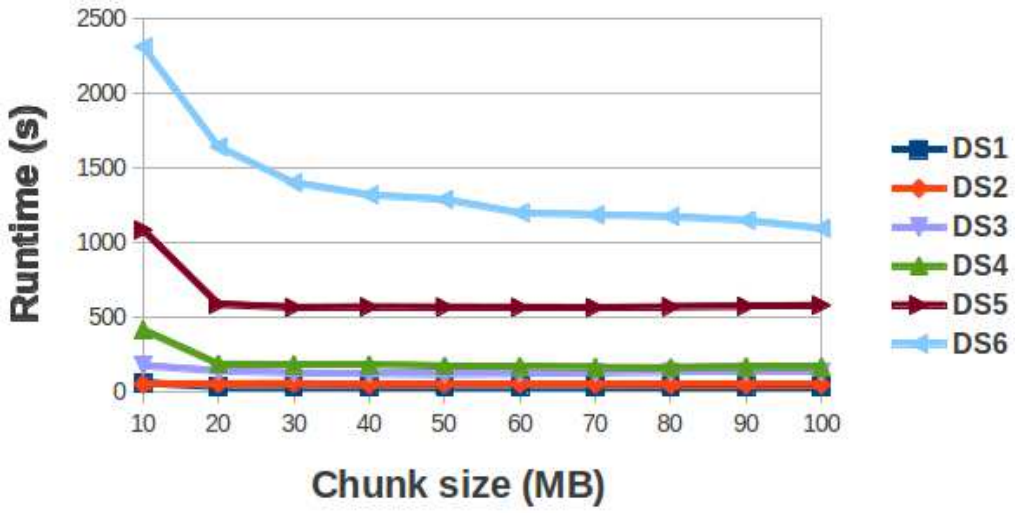


Figure 4.11: Effect of chunk size on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

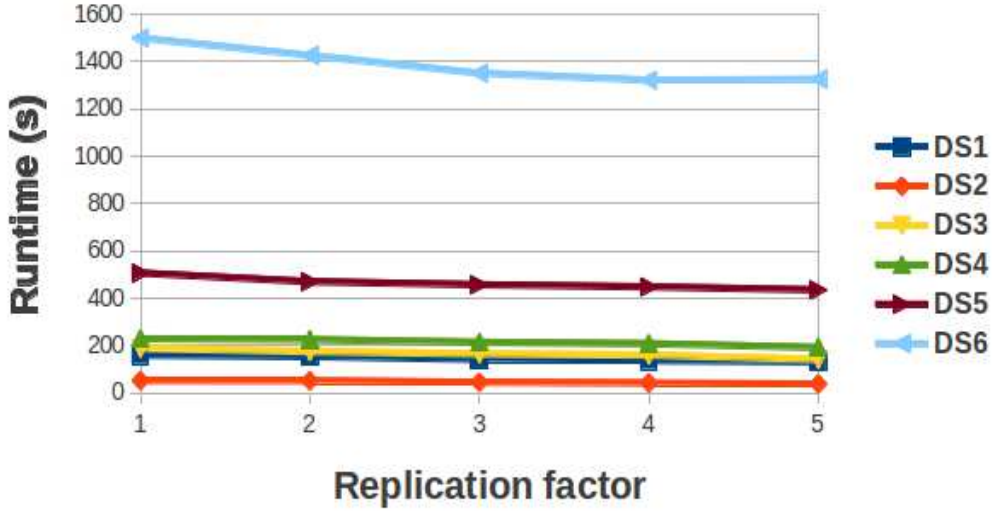


Figure 4.12: Effect of replication factor on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

The experimentations presented in Figure 4.11 show that with small values

of chunk size and with big datasets, our approach present high runtime values. Otherwise, the other values of chunk size do not notably affect the results.

As shown in Figure 4.12, the runtime of our approach is slightly inversely proportional to the replication factor (number of copies of data). This is explained by the high availability of data for MapReduce tasks. Also, a high replication factor helps ensure that the data can survive the failure of a node.

4.4 Conclusion

In this chapter, we addressed the issue of distributing the frequent subgraph mining process which is the first axis of this thesis. We have described our proposal for distributed subgraph mining from large scale graph databases in the cloud. The proposed approach relies on a density-based partitioning to build balanced partitions of a graph database over a set of machines. By running experiments on a variety of datasets, we have shown that our method is interesting in the case of large scale databases.

In the next chapter, we will address the monetary aspect of cloud-based applications. We will propose cost models for the distributed subgraph mining task in the cloud. These cost models are intended to assist users to configure the proposed approach of distributed frequent subgraph mining according to one (or more) objective(s). Such objectives include financial budget, response time and quality of the result constraints.

Key points

- We propose a MapReduce-based framework for approximate large scale frequent subgraph mining.
- We propose a density-based data partitioning technique using MapReduce in order to enhance the default data partitioning technique provided by MapReduce.
- We experimentally show that the proposed solution is reliable and scalable in the case of huge graph datasets.

Publications

International journals:

- Sabeur Aridhi, Laurent d'Orazio, Mondher Maddouri and Engelbert Mephu Nguifo. Density-based data partitioning strategy to approximate large scale subgraph mining. Information Systems, Elsevier, 2013.

Oral presentations:

- Sabeur Aridhi, Laurent d'Orazio, Mondher Maddouri and Engelbert Mephu Nguifo. Un partitionnement basé sur la densité de graphe pour approcher la fouille distribuée de sous-graphes fréquents. Big Data Mining and Visualization, Paris, 2013.
- Sabeur Aridhi, Laurent d'Orazio, Mondher Maddouri and Engelbert Mephu Nguifo. Fouille de sous-graphes fréquents dans les nuages. Journée sur le Décisionnel dans le Nuage (Cloud BI), Lyon, 2013.

CHAPTER 5

Cost models for distributed pattern mining in the cloud

Contents

Goals

In most cases, the distribution of the pattern mining process generates a loss of information in the output results. Reducing this loss may affect the performance of the distributed approach and thus, the monetary cost when using cloud environments. In this context, cost models are needed to help selecting the best parameters of the approach in order to achieve better performance. In this chapter, we define new cost models for managing and mining patterns. These cost models consist of monetary cost components that are primordial in a cloud. We define also a set of objective functions with respect to the needs and the financial budget of customers.

5.1 Background and related works

In this section, we present a description of works that propose cost models for distributed data mining techniques. We first survey existing works. Then, we summarize them according to the associated technique, the used architecture model and the defined cost model.

5.1.1 Existing cost models for distributed data mining techniques

Several Distributed Data Mining (DDM) systems have been proposed [Ghoting 2011a, Chu 2006, Riondato 2012, Foundation 2010]. However, they do not incorporate an optimiser to be able to estimate the costs associated with the various distributed data mining scenarios. Consequently, several cost models have been developed for estimating costs of distributed data mining applications [Krishnaswamy 2004, Marbán 2008, Ogunde 2011, Nguyen 2012].

Krishnaswamy etal.'s approach. In [Krishnaswamy 2004], the authors develop a priori estimates of the computation and communication components of response time as the costing strategy to support optimization in distributed data mining. The authors define the response time of a task in a distributed data mining context as the sum of three components: communication, computation and knowledge integration. The response time for distributed data mining T is computed as follows:

$$T = t_{dm} + t_{com} + t_{ki}$$

where t_{dm} is the time taken to perform data mining, t_{com} is the time involved in communication and t_{ki} is the time taken to perform knowledge integration (the time taken to integrate the results from the distributed datasets).

DMCOMO. In [Marbán 2008], the authors propose DMCOMO, a cost model for data mining applications. DMCOMO aims to estimate the global cost of a generic data mining project. The authors proposed six cost drivers of the proposed model such as:

- Data cost drivers: It makes reference to the effort of data management in

the project.

- Data mining models: It makes reference to the number of data mining models to be created.
- Platform: It makes reference to the development platform. It includes the number of data sources where data are stored, the number of different data servers and the communication effort.
- Techniques and tools: It considers the effort of deciding which tool, technique and machine will be used to generate the models.
- Project: Features of the project such as the number of participating departments must be considered to estimate the effort of the data mining project.
- Staff: It considers the required effort from staff to reach an agreement in decisions of the project staff.

Once cost drivers have been defined and information about the cost drivers were gathered, the equation of DMCOMO was created through a multivariate linear regression [Griffiths 1993, Weisberg 1985]. The equation will be similar to:

$$y = a_0 + \sum_{i=1}^{n} a_i x_i + e_i,$$

where y is the dependent variable that corresponds to the effort measured in $men \times month$ (MM) that one wishes to estimate, x_i is the i^{th} independent variable that corresponds to the i^{th} cost driver, n is the number of independent variables (number of cost drivers), a_i are constants and e_i is the error in the i^{th} estimation. As a result of linear regression, a_i values are obtained and hence, the effort is computed.

Ogunde etal.'s approach. In [Ogunde 2011], the authors present an optimized model for estimating the response time of Distributed Association Rule Mining (DARM). In this work, three estimates were defined:

- **The communication cost estimates:** They involve the time needed for the computing agent to travel from the agent zone (AZ) to the data sources.

- **The local association rule mining costs:** They make reference to the time needed for mining association rule locally at each data source.

- **The results information transfer costs:** They make reference to the time needed for the computing agent to travel back to the agent zone with results information concerning each local mining.

The overall response time for the distributed association rule mining T would be calculated as follows:

$$T = t_{darm} + t_{dki},$$

where t_{darm} is the time taken to perform mining in a distributed environment and t_{dki} is the time taken to perform distributed knowledge integration and return the results to the requesting server. The term t_{darm} is defined by:

$$t_{darm} = t_1(AZ,i) + \max_{i=1}^{n} t_2(i) + t_3(i,AZ)$$

where the first term is the time taken by the computing agent to travel from the agent zone to data source i. The second term is the maximum of the times taken by the computing agent to mine at all data sources. The third term is the time taken for the agent to travel from the data source back to the agent zone with the results information. The authors discussed the values of t_{dki} according to the number of used data servers and the number of data mining agents.

MRShare. In [Nykiel 2010], the authors propose the MRShare framework that transforms a batch of MapReduce queries into a new batch that will be executed more efficiently, by merging jobs into groups and evaluating each group as a single query. The authors define a cost model for MapReduce that provide a solution that derives the optimal grouping of queries. The total cost of executing a set J of n individual jobs is the sum of the cost T_{read} to read the data, the cost T_{sort} to do the sorting and copying at the map and reduce nodes, and the cost T_{tr}

of transferring data between nodes. Thus, the cost in MapReduce is:

$$T(J) = T_{read}(J) + T_{sort}(J) + T_{tr}(J)$$

where the values of $T_{read}(J)$, $T_{sort}(J)$ and $T_{tr}(J)$ with grouping of queries are not the same without grouping.

Another attention was carried by [Nguyen 2012] to data management cost models in cloud environments. In their work, the authors propose new cost models that fit into the pay-as-you-go paradigm of cloud computing. They addressed the multi-criteria optimization problem of selecting a set of materialized views while optimizing the global monetary cost of storing and querying a database in a cloud environment. The total cloud data management cost C is defined by:

$$C = C_c + C_s + C_t$$

where C_c is the sum of computing costs, C_s is the sum of storage costs and C_t is the sum of data transfer costs. The proposed cost models complement the existing materialized view cost models with a monetary cost component that is primordial in the cloud [Nguyen 2012].

5.1.2 Summary of existing cost models

Table 5.1 presents the above mentioned cost models approaches. It describes the associated technique, the used architecture model and the defined cost model.

As shown in Table 5.1, the works [Krishnaswamy 2004, Marbán 2008, Ogunde 2011] deal with cost models of classic architectural models used in the development of DDM systems namely, client-server and software agents. However, the proposed cost models in these works do not fit into cloud computing paradigm where the users only pay for the resources they use.

In [Nguyen 2012, Nykiel 2010], the authors deal with data management and execution aspect of MapReduce framework in a cloud setting. However, they do not include cost models for data mining processes in the top of MapReduce.

[1] Cost models are computed by application of linear regression to correlated variables.

Table 5.1: Summary of recent cost models for cloud-based techniques.

Approach	Associated technique	Architecture model	Cost models
Krishnaswamy etal.'s approach [Krishnaswamy 2004]	All data mining tasks	Client-server and mobile agent	Response time
DMCOMO [Marbán 2008]	All data mining tasks	No architecture model used [1]	$men \times month$
Ogunde etal.'s approach [Ogunde 2011]	Association rule mining task	Mobile agent	Response time
MRShare [Nykiel 2010]	A MapReduce job	MapReduce	Response time

Moreover, they do not consider monetary costs in the case of cloud-based data mining applications.

To the best of our knowledge, cost models for MapReduce-based pattern mining applications in cloud environments have not been developed.

5.2 Cost models for distributed pattern mining in the cloud

In this section, we first introduce a simple use case that serves as a running example throughout this chapter. Then, we present the proposed cost models for distributed pattern mining in the cloud.

5.2.1 Running example

In order to illustrate our cost models, we rely on a subgraph mining example. Considering a graph dataset containing a set of community networks of a social network (nodes represent people and edges represent interactions between them). Social network analysts need to examine the community networks patterns per day, month, and year. The analysis consists of extraction of frequent subgraph patterns in community networks. It includes queries like "frequent subgraphs that occur in more than 30% of graphs in the database". Let us apply this pricing model onto our use case running on two small instances of the Windows Azure offer presented in Chapter 3. We suppose that our dataset contains ten million graphs and its size on disk is $100\,\mathrm{GB}$. The query example consists of retrieving community networks patterns that occur in more than 30% of graphs in the database and of producing a query result of $10\,\mathrm{GB}$.

According to the Windows Azure offer, the costs of the two small instances, used in our example, is $\$0.008 \cdot 2 = \$0.016$ per hour. The cost of bandwidth consumption (query result of $10\,\mathrm{GB}$) is $(10 - 5) \cdot \$0.12 = \$0.60$. In Chapter 3, we mention that Windows Azure Storage provides storage of non-relational data, including storage blob, table and disk. It provides two options for storage: lo-

cally and geographically redundant. The locally redundant storage option allows multiple replicas of data within a single sub-region to provide the highest level of durability. The geographically redundant storage option offers an extra level of durability by replicating data between two remote sub-regions. In our running example, the monthly storage price of our data ($100\,\mathrm{GB}$ dataset) with the locally redundant storage option is $\$0.07 \cdot 100 = \$7$.

5.2.2 Distributed pattern mining cost

Let C_{dm} be the data management cost and C_c be the cost of computing patterns in a distributed environment. We define the total cost C of distributed pattern mining by:

$$C = C_{dm} + C_c. \tag{5.1}$$

Depending on the model used to distribute the computations (i.e. MapReduce or other) and the different parameters within each model, the factors which determine C_{dm} and C_c change. In our work, we consider MapReduce-based approaches. Two types of parameters setting are required in this setting. First, parameters related to the pattern mining process such as the support threshold and the size of the database. Second, a number of choices are essential to fully specify how the MapReduce job should execute the distributed subgraph mining process:

- RF: The replication factor (number of copies of data),
- m: number of map tasks in the MapReduce job,
- r: number of reduce tasks in the MapReduce job, and
- CP: whether the output data from the map (reduce) tasks should be compressed before being written to disk.

Let us define some functions that we use to express our cost models.

- Function $s(\cdot)$ returns the size in GB of any dataset, e.g., $s(DS)$ is the size of the dataset DS and $s(R)$ is the size of the result data R,

- Function $ts(\cdot)$ returns the storage time of any dataset, e.g., $ts(DS)$ is the storage time of the dataset DS in the cloud and $ts(R)$ is the storage time of the result data R,

- Function $t_{map}(i, Part_i(DS))$ returns the runtime taken by the map task to process the i^{th} partition of DS,

- Function $t_{reduce}(k)$ returns the runtime taken by the reduce task of the k^{th} reducer.

5.2.2.1 Data management cost

We define the data management cost C_{dm} as the sum of data transfer cost C_t and storage cost C_s. Formally, the data management cost is:

$$C_{dm} = C_t + C_s. \tag{5.2}$$

Data transfer cost depends on several parameters: the size of the dataset, the size of the results and the pricing model applied by the Cloud Service Providers (CSP). The total data transfer cost C_t is the sum of the input data transfer and the output data transfer costs. The input data transfer cost is the product of the CSP's atomic transfer cost c_{ti} of the input data and the total size of input data. The output data transfer cost is the product of the CSP's atomic transfer cost c_{to} of the output data and the total size of result data ($s(R)$):

$$C_t = c_{ti} \cdot s(DS) + c_{to} \cdot s(R). \tag{5.3}$$

As illustrated in equation (5.3), the total data transfer cost is proportional to the total size of input and output data. We notice that most cloud providers such as Windows Azure and Amazon Web Service (AWS) do not charge for input data transfers. Consequently, total data transfer cost C_t become:

$$C_t = c_{to} \cdot s(R). \tag{5.4}$$

Example 1 *In our example, with* $10\,\text{GB}$ *of bandwidth consumption, data transfer cost is calculated by* $C_t = s(R) \cdot cto = (10-1) \cdot \$0.09 = \$0.81$ *in the case of Amazon Web Service pricing model and by* $C_t = s(R) \cdot cto = (10-5) \cdot \$0.12 = \$0.6$ *in the case of Windows Azure pricing model.*

Storage cost depends on parameters such as the size of the dataset, the storage time, the type of data replication (locally redundant storage or globally redundant storage) and the CSP's pricing policy. We assume that the storage period in the cloud is divided into intervals. In each interval, the size of the stored data is fixed. The total storage cost (C_s) is the CSP's fixed price per GB (c_s^{CSP}) multiplied by the size of intial $s(DS)$ and result data $s(R)$, multiplied by the sum of sizes of the initial dataset and the result data, multiplied by their respective storage time during the intervals:

$$C_s = \sum_{Intervals} c_s^{CSP} \cdot (s(DS) + s(R)) \cdot (t_{end} - t_{start}), \tag{5.5}$$

where t_{start}, t_{end} are start and end point of an interval.

Example 2 *Considering that* $100\,\text{GB}$ *of data has been stored for 12 months. At the beginning of the eighth month, we generate 10 GB of result data in the cloud. Thus, we have two intervals. The first with* $t_{end} = 8$ *and* $t_{start} = 0$ *and the second with* $t_{end} = 12$ *and* $t_{start} = 8$*. Using Amazon S3 storage pricing (*$c_s^{CSP} = \$0,14$ *per* GB *for the first* TB*), the storage cost is:* $C_s = 100 \cdot \$0.14 \cdot (8-0) + 110 \cdot \$0.14 \cdot (12-8) = \$173.6$*. In the case of Windows Azure storage pricing (*$c_s^{CSP} = \$0,07$ *per* GB *for the first* TB *and with the locally redundant storage option), the storage cost is:* $C_s = 100 \cdot \$0.07 \cdot (8-0) + 110 \cdot \$0.07 \cdot (12-8) = \$86.8$.

By combining equations (5.2), (5.3) and (5.5), the total data management cost is:

$$C_{dm} = \sum_{Intervals} c_s^{CSP} \cdot (s(DS) + s(R)) \cdot (t_{end} - t_{start}) + c_t \cdot (s(DS) + s(R)). \tag{5.6}$$

As illustrated in equation (5.6), the data management cost depends essentially on the size of input data and result data. Indeed, it depends on the nature of data under consideration.

Beside the data management cost, it is necessary to study the computing cost on the data. In the context of our work, this computing cost consists of pattern mining cost.

5.2.2.2 Mining cost

In a cloud environment, mining processes are executed on computing instances $\{I_i\}_{i=1\cdots n}$ with different performances in terms of number of CPUs, available RAM, etc., and thus, with different costs.

The cost for renting instance I_i is denoted by $c(I_i)$. This cost must be paid at each connection to the cloud. We define the cost of computing patterns by:

$$C_c = \sum_{i=0}^{n} c(I_i) \cdot T_{mining}, \tag{5.7}$$

where

$$T_{mining} = (t_{part} + \max_{j=0}^{m}(t_{map}(j, Part_j(DS))) + \max_{k=0}^{r}(t_{reduce}(k)) + CP \cdot t_{compress}), \tag{5.8}$$

where t_{part} is the partitioning time, $t_{map}(j, Part_j(DS))$ is the time taken by the j^{th} Map task to process the j^{th} partition of DS, $t_{reduce}(k)$ is the time taken by the k^{th} Reduce task, $t_{compress}$ is the compression time of the result files and m is the number of map tasks, and r is the number of reduce tasks. The values of t_{part}, $t_{map}(j, Part_j(DS))$ and $t_{reduce}(k)$ are estimated experimentally.

In our work, we assume that mining processes are executed on a constant number of instances (n) with the same performances. Let $c(I)$, be the cost for renting one instance. The computing cost defined in equation (5.7) is:

$$C_c = c(I) \cdot n \cdot T_{mining}, \tag{5.9}$$

5.3 Optimization process

In this section, we investigate how the parametrization of the pattern mining approach and of MapReduce framework may impact the mining process performance. We first present our objective functions. Then, we discuss the resolution methods.

5.3.1 Objective functions

Based on the ideas in [Nguyen 2012], we distinguish in this section four objective functions with respect to the needs and capacity of customers. Such needs include budget limit, response time limit and mining quality limit.

5.3.1.1 Response time

The idea here is to achieve better performance in terms of reponse time. Given a predefined financial budget B and a predefined mining quality limit Q, our objective in this scenario is to select the best parameters that minimize the mining process in the cloud:

$$Obj_1 = \begin{cases} \text{minimize } T_{mining}, \\ C = C_{dm} + C_c \leq B, \\ MiningQuality \geq Q. \end{cases} \tag{5.10}$$

Figure 5.1 presents the feasible solutions that minimize the response time with respect to financial budget B and a predefined mining quality limit Q. Each point in Figure 5.1 corresponds to a feasible solution of our objective function without considering constraints defined in equation (5.10). The red points correspond to solutions that verify our mining quality limit (X axis) and budget limit (Y axis) constraints.

We notice that each point in Figure 5.1 corrsponds to a response time value. The optimal solution in this context is the solution that presents the lower value of the response time.

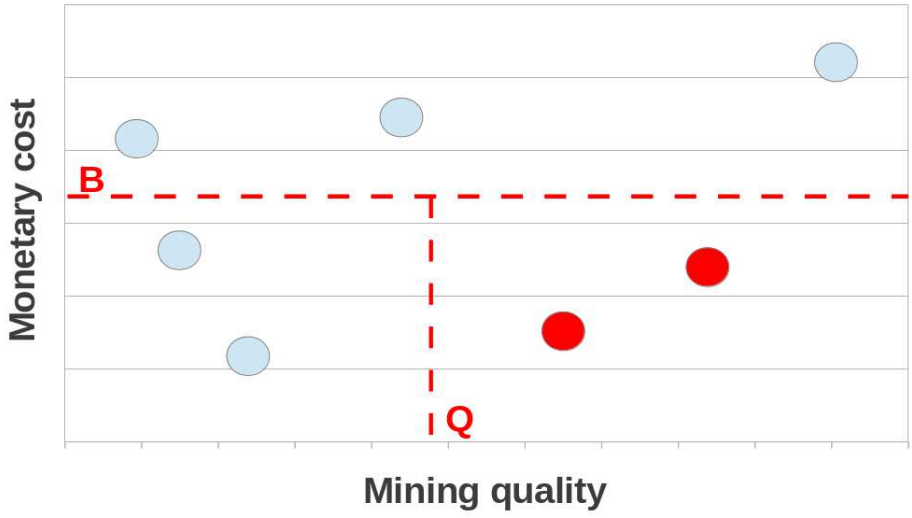


Figure 5.1: Minimizing response time under monetary cost and mining quality constraints.

5.3.1.2 Monetary cost

For a predefined response time limit T and a predefined mining quality limit Q, the objective in this scenario is to select the best parameters that minimize the monetary cost of the mining process in the cloud:

$$Obj_2 = \begin{cases} \text{minimize } C = C_{dm} + C_c, \\ T_{mining} \leq T, \\ MiningQuality \geq Q. \end{cases} \tag{5.11}$$

Figure 5.2 presents the set of feasible solutions that minimize the monetary cost with respect to a response time limit T and a predefined mining quality limit Q. In the Figure 5.2, red points correspond to solutions that verify our mining quality limit (X axis) and response time (Y axis) constraints.

From the set of feasible solutions, we select the optimized solution that presents the lower value of monetary cost.

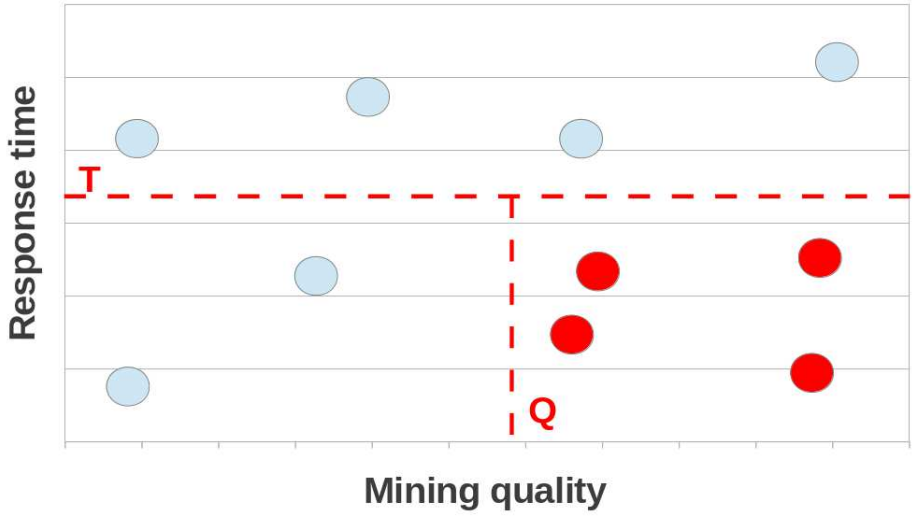


Figure 5.2: Minimizing monetary cost under mining quality and response time constraints.

5.3.1.3 Mining quality

The goal of this objective function is to achieve the optimized quality of results (the mining quality). Given a predefined response time limit T and a predefined financial budget B, our objective in this scenario is to select the best parameters that maximize the mining quality of the distributed pattern mining method in the cloud:

$$Obj_3 = \begin{cases} \text{maximize } MiningQuality, \\ T_{mining} \leq T, \\ C = C_{dm} + C_c \leq B. \end{cases} \tag{5.12}$$

In the case of the distributed subgraph mining task, the mining quality is defined by the $LossRate$ measure (see Definition 22 in Chapter 4). Therefore our objective in this scenario is to select the best parameters that minimize the value of $LossRate(S_1, S_2)$ where S_1 is the set of frequent subgraph generated by the exact solution and S_2 is the set of frequent subgraph generated by the distributed approach. Thus, the objective function (equation (5.12)) is given by:

$$Obj_3 = \begin{cases} \text{minimize } LossRate(S_1, S_2), \\ T_{mining} \leq T, \\ C = C_{dm} + C_c \leq B. \end{cases} \tag{5.13}$$

We show in Figure 5.3, the set of feasible solutions that maximize the mining quality with respect to a financial budget B and a predefined response time limit T. Points represented in Figure 5.3 corresponds to feasible solutions of our objective function without considering the constraints defined in equation (5.13). The red points correspond to solutions that verify our mining quality limit (X axis) and budget limit (Y axis) constraints.

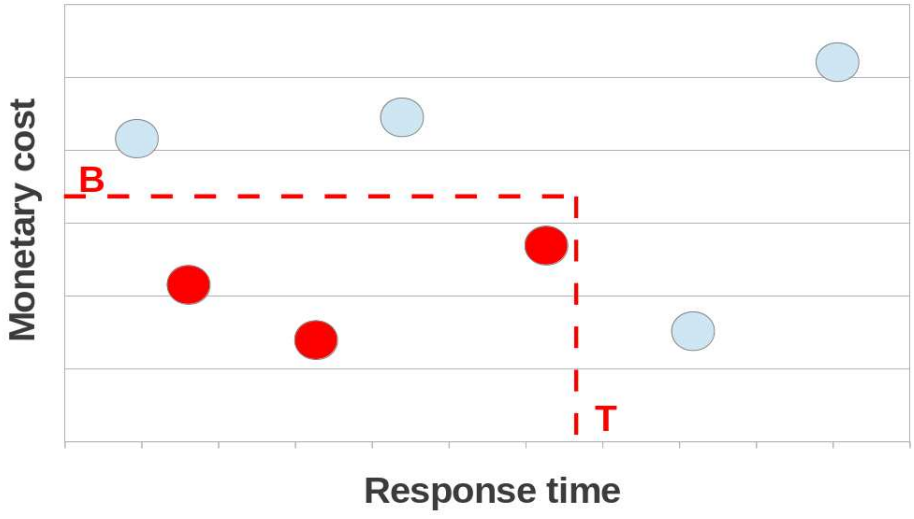


Figure 5.3: Maximizing mining quality under monetary cost and response time constraints.

The optimized solution here is the one that presents the higher value of mining quality. In the case of the distributed subgraph mining task, the optimized solution is the one that presents the lower value of $LossRate$.

5.3.1.4 Response time vs. monetary cost vs. mining quality tradeoff

Our objective in this scenario is to select the best parameters that offer the best tradeoff between query processing time, mining quality of the distributed pattern

mining method and financial cost:

$$Obj_4 = \begin{cases} \text{minimize } (T_{mining}, C) \text{ and maximize } (MiningQuality), \\ T_{mining} \leq T, \\ C = C_{dm} + C_c \leq B, \\ MiningQuality \geq Q. \end{cases} \tag{5.14}$$

In the case of the distributed subgraph mining task, the objective function (equation (5.14)) will be:

$$Obj_4 = \begin{cases} \text{minimize } (T_{mining}, C, LossRate(S_1, S_2)), \\ T_{mining} \leq T, \\ C = C_{dm} + C_c \leq B, \\ LossRate(S_1, S_2) \leq Q. \end{cases} \tag{5.15}$$

The above presented objective function consists of multi-objective function since more than one objective function to be optimized simultaneously. In the next section, we discuss the resolution of such objective function.

5.3.2 Optimization algorithm

Solving our multi-objective optimization problem can be done in different ways.

A possible method is to scalarize the problem [Zlochin 2004]. To do this, we convert the original problem into a single-objective optimization problem. We introduce weight parameters on processing time (α), mining quality (β) and financial cost (γ) in order to give the user control over this process. The objective function presented in equation (5.14) will be:

$$Obj_4 = \begin{cases} \text{minimize } (\alpha \cdot T_{mining} + \beta \cdot C + \gamma \cdot (1 - MiningQuality)), \\ T_{mining} \leq T, \\ C = C_{dm} + C_c \leq B, \\ MiningQuality \geq Q. \end{cases} \tag{5.16}$$

The resolution of such scalarized problem is simply done using the theory and methods of single criterion optimization [Zlochin 2004]. However, appropriate values of parameters of the scalarized problem (α, β, and γ in our case) should be found.

Another method to solve our multi-objective optimization problem is to compute all or a representative set of Pareto optimal solutions [Ehrgott 2005]. A Pareto optimal solution can be identified as non-dominated if none of the objective functions can be improved in value without degrading some of the other objective values (see Definition 25).

Definition 25 (Pareto optimal solution, Pareto front) *Let X be a set of feasible solutions, f_i is the i^{th} objective function and $k \geq 2$ is the number of objective functions. A feasible solution $x^1 \in X$ is said to (Pareto) dominate another solution $x^2 \in X$, if:*

$$\begin{cases} f_i(x^1) \leq f_i(x^2) \text{ for all indices } i \in \{1,2,\cdots,k\}, \text{ and} \\ f_j(x^1) < f_j(x^2) \text{ for at least one index } i \in \{1,2,\cdots,k\}. \end{cases}$$

A solution $x^1 \in X$ is called Pareto optimal, if there does not exist another solution that dominates it. The set of Pareto optimal solutions is called the Pareto front.

Pareto optimal solutions are very useful for decision makers who are faced with multiple objectives to make appropriate compromises, tradeoffs or choices.

5.4 Experimental validation

In this section, we first describe the overall setup of our preliminary experimentation effort. Then, we present the results we have obtained. We focused our experiments on solving the problem of distributed subgraph mining in the cloud. We adopted the Pareto-based multi-objective optimization solution which aim to produce all Pareto optimal solutions.

5.4.1 Experimental setup

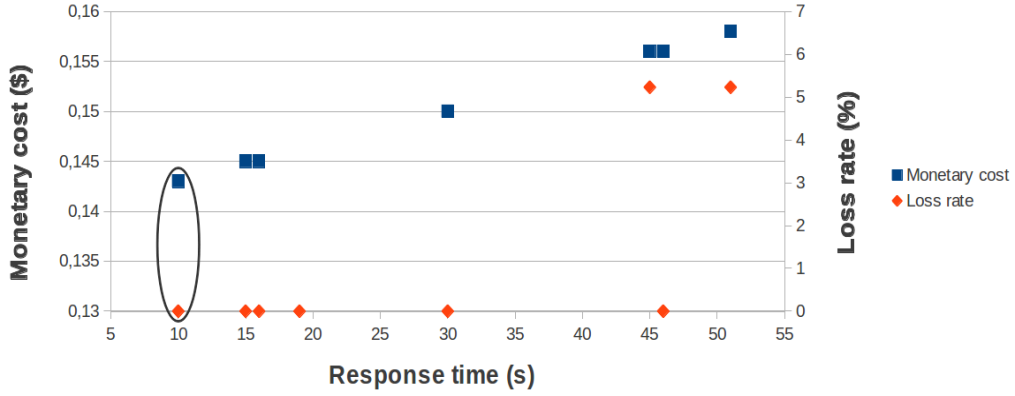
All experiments of our approach were carried out using a virtual cluster composed of five virtual machines. Each virtual machine is equipped with a Quad-Core AMD Opteron(TM) processor 6234 2.40 GHz vCPU and 4 GB of RAM. All used machines feature Hadoop (version 0.20.2) and operate on Linux Ubuntu.

For our experiments, we have generated our data based on the obtained results from our proposed approach for distributed subgraph mining in the cloud. We used results that correspond to the distributed subgraph mining from the $DS2$ dataset (see Section 4.3.2.1 of Chapter 4) to form our set of multi-objective points. For each set of parameters, we noticed the values of our objectives such as the response time (T_{mining}), the monetary cost (C) and the mining quality ($LossRate$ in the case of subgraph patterns). The used parameters include MapReduce parameters (see Section 5.2.2) and distributed subgraph mining approach parameters (see Chapter 4). Monetary cost values are estimated based on the Windows Azure pricing model (see Chapter 3, Section 3.4.2). We suppose that our experimental environment is close to the large cloud instances provided by Windows Azure. Consequently, we use the corresponding costs to compute the values of the monetary cost C of each experiment using our virtual cluster.

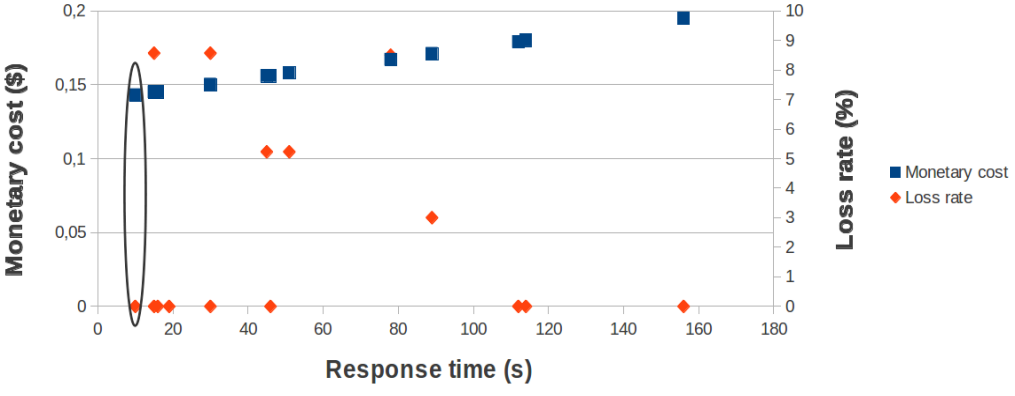
5.4.2 Experimental results

During our experimental study, we examined the four objective functions described in Section 5.3.1.

Figure 5.4 draws the set of feasible solutions that minimize the response time (the Obj_1 objective function) of our distributed subgraph mining approach under monetary cost (C) and mining quality ($LossRate$) constraints. Each solution is represented by two points (a blue square point and a red diamond point). The blue square point corresponds to monetary cost in function of response time. The red diamond point corresponds to the loss rate in function of response time. Thus, the two points representing a solution have the same value of response time.



(a) Budget limit = \$0.16 and LossRate limit = 7%



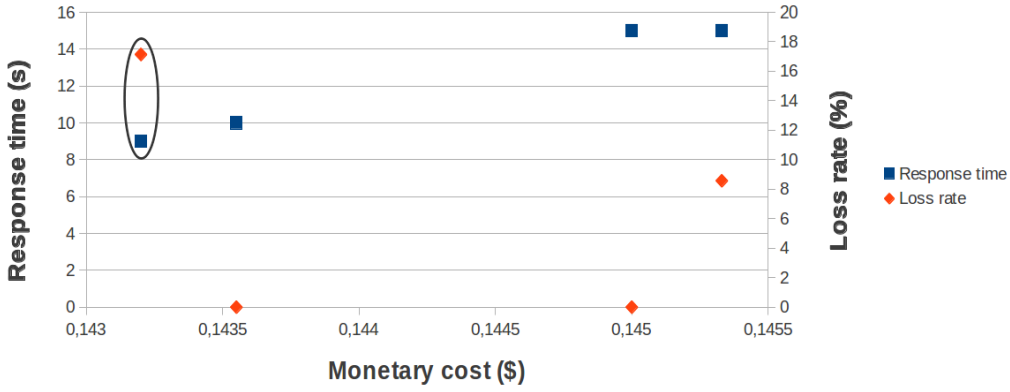
(b) Budget limit = \$0.20 and LossRate limit = 10%

Figure 5.4: Minimizing response time under monetary cost and loss rate constraints.

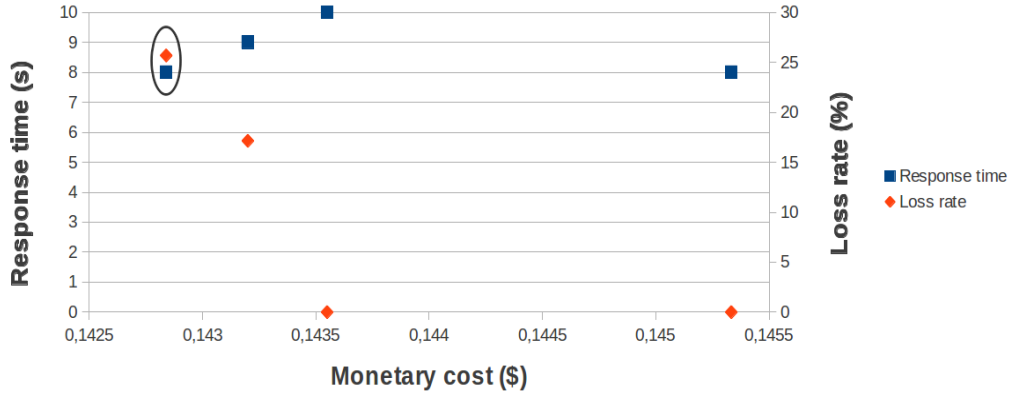
Optimal solutions are determined by selecting solutions that present lower values of response time. For example, with budget limit = \$0.16 and loss rate limit = 7% (see Figure 5.4a), we distinguish one optimal solution (surrounded by

an ellipse) which allows the lower value of response time. We notice that we can find more than one solution that allows a lower response time value.

In the Figure 5.5, we present the set of feasible solutions to solve the objective function associated to the monetary cost (Obj_2). Feasible solutions are represented by couples of points. Each couple consists of one blue square point and one red diamond point. The blue square point corresponds to response time in function of monetary cost. The red diamond point corresponds to the loss rate in function of monetary cost.



(a) Response time limit = 15 and LossRate limit = 20%



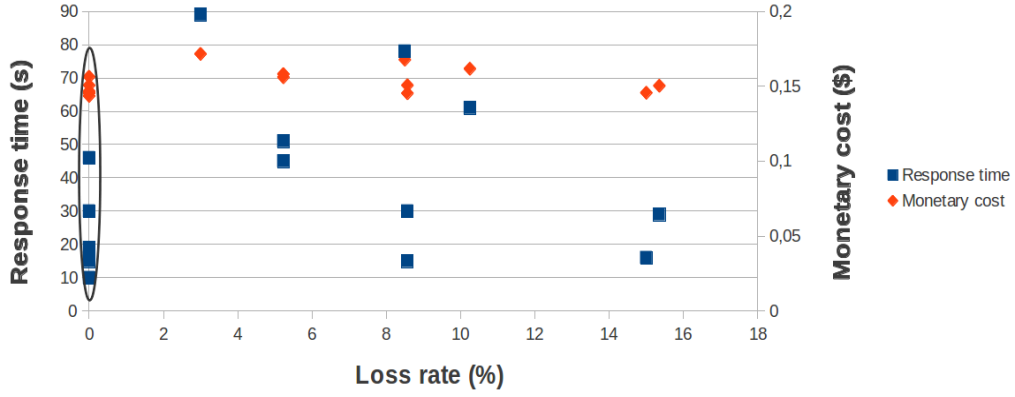
(b) Response time limit = 10 and LossRate limit = 30%

Figure 5.5: Minimizing loss rate under monetary cost and response time constraints.

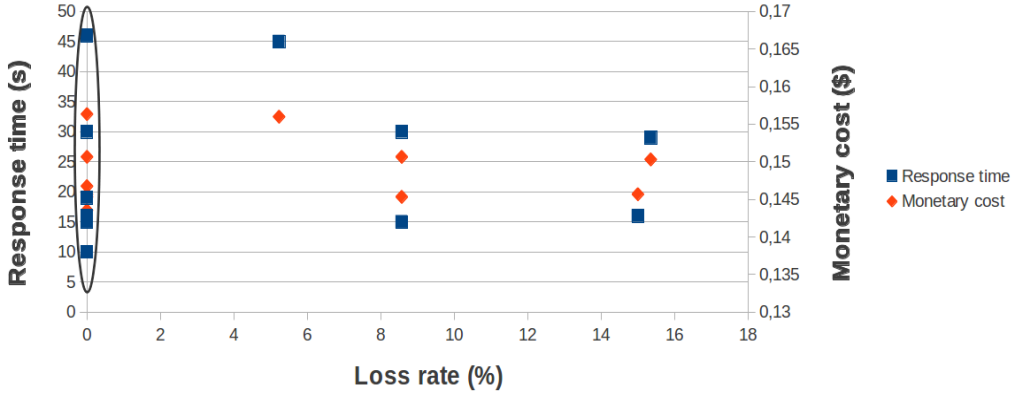
As shown in Figure 5.5, we can identify optimal solutions by selecting the solutions that present lower values of monetary cost in comparision with the set

of feasible solutions. For example, with response time limit = 15 and loss rate limit = 20% (see Figure 5.5a), we select one optimal solution (surrounded by an ellipse) which allows the lower value of loss rate.

Figure 5.6 illustrates the set of feasible solutions that optimize the objective function (Obj_3) related to the loss rate of our subgraph mining approach in the cloud. Similarly, feasible solutions are represented by couples of points. A couple of points contains one blue square point and one red diamond point. The blue square point corresponds to response time in function of loss rate. The red diamond point corresponds to the monetary cost in function of loss rate.



(a) Budget limit = \$0.17 and Response time limit = 90



(b) Budget limit = \$0.17 and Response time limit = 50

Figure 5.6: Minimizing loss rate under monetary cost and response time constraints.

We recall that each feasible solution consists of a parametrization of the cloud-

based subgraph mining approach. As illustrated in Figure 5.6a, the set of optimal solutions (surrounded by an ellipse) contains more than one optimal solution (six optimal solutions) that minimize the loss rate. Therefore, we have six possible parametrizations of our cloud-based subgraph mining approach.

In order to solve the multi-objective function defined in Section 5.3.1.4, we computed all Pareto optimal solutions from our data (a set of multi-objective points). Figure 5.7 presents the set of feasible solutions and distinguishes the set of Pareto optimal solutions. Feasible solutions are represented by blue points while Pareto optimal solutions are represented by red points.

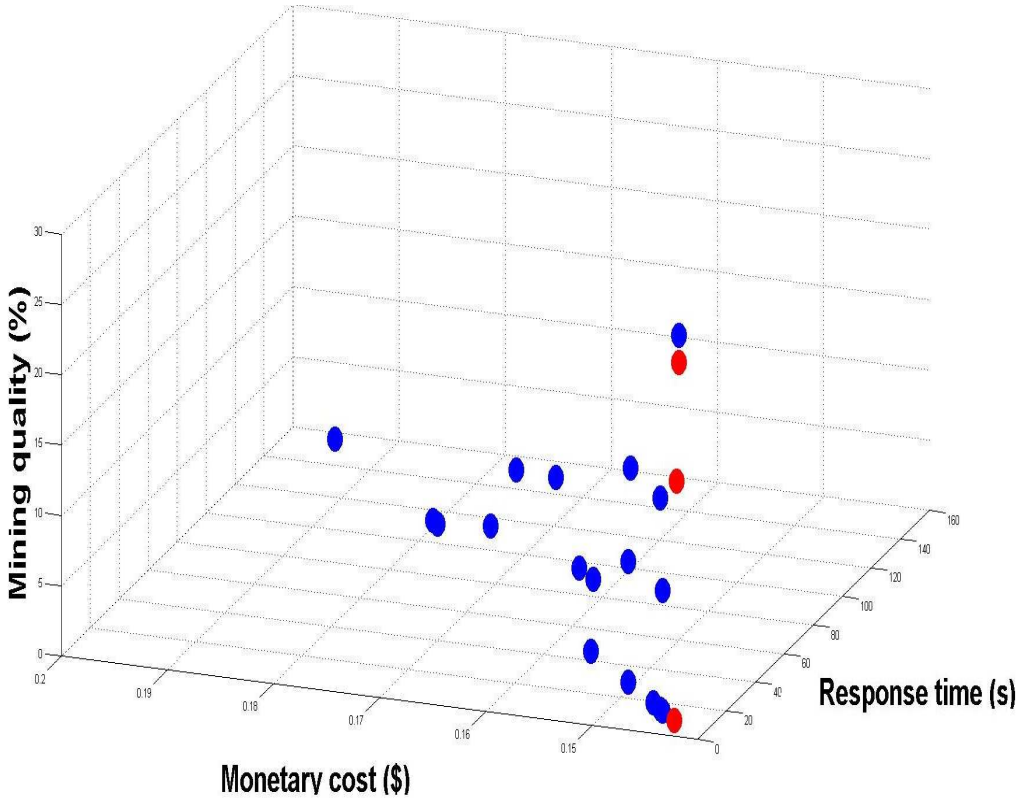


Figure 5.7: A Pareto optimal solutions for minimizing loss rate, monetary cost and response time. We used Response time limit = 200s, Budget limit = $0.50 and Mining quality limit = 30%.

The Pareto optimal solutions illustrated in Figure 5.7 aim to quantify the trade-offs in satisfying the different objectives (response time, monetary cost and

mining quality). As shown in Figure 5.7, we have three Pareto optimal solutions (colored in red). We show in Table 5.2 the details (parameters values) of our Pareto optimal solutions:

Table 5.2: Pareto optimal solutions.

Optimal solution	Cloud parameters			Mining approach parameters
	Number of cloud instances	Data compression (CP)	Replication factor (RF)	
First solution	4	No	3	$\tau = 60\%$
Second solution	4	Yes	3	$\tau = 0\%$
Third solution	4	No	3	$\tau = 0\%$

We notice that the above presented optimal solutions may help the parametrization of cloud-based subgraph mining applications. They provide suggestions for the choice of parameters (cloud parameters and mining approach parameters). However, it is suitable to provide one suggestion instead of many. This can be done by ranking optimal solutions based on a user-defined parameter.

5.5 Conclusion

In this chapter, we presented the second contribution of this thesis. It consists of two levels. The first level is novel cost models for pattern mining in the cloud. We focused the defined cost models on subgraph patterns in cloud computing. The proposed cost models consist of monetary cost components that are primordial in the cloud.

The second level consists of the definition of a set of objective functions with respect to the needs and the financial capacity of customers. An experimental study was carried out in the case of cloud-based subgraph mining. It provided a first evaluation of our approach by selecting the optimal solutions that minimize our objectives such as the monetary cost, the response time and the loss rate.

By this chapter, we finish with the second axis of this thesis. However, several other extensions are open and under development. We give more details about these ongoing works in the concluding chapter.

Key points

- We presented the background information related to cost models for distributed pattern mining in the cloud.
- We defined a set of objective functions with respect to the needs and the financial capacity of customers.
- We discussed the resolution of the defined objective functions.
- We present experimental results that provide a first evaluation of the proposed cost models.

Chapter 6

Conclusion and prospects

Contents

Goals

In this chapter, we conclude the thesis by summarizing our contributions. Then, we highlight the ongoing works we are conducting in extension to this thesis.

6.1 Summary of contributions

This thesis deals with distributed frequent subgraph mining from huge graph databases in the cloud. Firstly, it attends to propose a scalable approach for large scale frequent subgraph mining in the cloud. Secondly, it handles with monetary cost models for distributed pattern mining in the cloud and it focuses on subgraph patterns.

In this section, we recall the main lines that trace the results of our research. We first present the proposed framework for distributing frequent subgraph mining. Then, we give an overview of the proposed cost models for distributed pattern mining in the cloud.

6.1.1 A framework for distributing frequent subgraph mining in the cloud

The first contribution of this thesis consists of a MapReduce-based framework to approximate large scale frequent subgraph mining. The proposed approach allows many partitioning techniques of the input graph database. In this thesis, we proposed a data partitioning technique that considers data characteristics. It uses the density of graphs in order to partition the graph database. Such a partitioning technique allows a balanced computational load over the distributed collection of machines. We experimentally show that the performance and scalability of our approach are satisfying for large scale graph databases.

6.1.2 Cost models for distributing frequent pattern mining in the cloud

We addressed the multi-criteria optimization problem of tuning thresholds of distributed frequent pattern mining in the cloud. The purpose is to optimize the global monetary cost of storing and querying data in the cloud. We designed cost models for managing and mining graph data with a large scale subgraph mining framework over a cloud architecture. Besides, we defined objective functions that

consider the needs of customers such as budget limit, response time limit and result quality limit. We discussed the use of the proposed cost models in the case of subgraph patterns and we discussed the resolution of the defined multi-objective functions. We validated experimentally the defined cost models and objective functions in the case of distributed subgraph mining in the cloud.

6.2 Future works and prospects

In this section, we present the main axes of our future works. We are currently working on two major axes. In the first axis, we are working on improving the distributed framework for large scale frequent subgraph mining. The second axis aims to improve the proposed cost models.

6.2.1 First axis: improvement of the distributed frequent subgraph mining in the cloud

The distributed frequent subgraph mining approach consists of two main steps: (1) the partitioning step and (2) the distributed computing step. Improvements of our approach will focus on both steps.

Improvements of the partitioning method: In the partitioning step of our approach, we divide the graph database into a set of buckets that serve to construct the final set of partitions. In the experimental study, we varied the number of buckets from two to five buckets. We have shown that the use of high number of buckets increases balancing. An extended experimental study might be carried out to give more precisions about our observations and answer questions like:

- What is the maximum number of buckets and/or partitions to use in order to reach best performance?
- What is the relation between the number of buckets and the chunk size and/or the number of partitions?,

- What is the size of chunk to use in the partitioning step and in the distributed subgraph mining step?
- How to tune the tolerance rate value in order to achieve better performance and result quality?.

Future works include the generalization of the partitioning method in order to offer the possibility of using different topological graph properties [Li 2012] instead of the density. Such partitioning method will provide the possibility to use a panoply of topological graph attributes in the partitioning step. The choice of the property basically depends on the data under consideration.

Another possible future work in this context is the study of the relation between database characteristics and the choice of the partitioning technique. This study aims to help the user to choose the appropriate partitioning method according to data characteristics.

Improvements of the distributed subgraph mining step: This step is a two stage body. The first stage consists in applying an existing frequent subgraph mining technique on each partition of the input database in parallel. A set of locally frequent subgraphs is produced from each partition. The second stage consists in generating the set of globally frequent subgraphs. Experiments of our approach have shown that the result quality of the distributed subgraph mining depends on the tolerance rate threshold. This threshold determines the minimum support in the stage of mining locally frequent subgraphs. Otherwise, tolerance rate values are fixed by the user. In future works, we will first study the behavior of our approach according to various tolerance rate values. Then, we plan to automatize the selection of tolerance rate values. This selection depends essentially on the size and the number of partitions.

In the following, another future directions related to the improvement of our approach:

Performance and scalability improvement: This direction consists of improving the runtime of our approach with task and node failures [Yang 2010, Quiane-Ruiz 2011]. This task can be done by incorporating mechanisms of task and node failures management. Such mechanisms should ensure minimal loss of

information in the case of failures. Moreover, our approach should continue to operate without interruption during the repair process.

Portability improvement: Future works include the extension of our approach to different parallel programming models such as Open Computing Language (OpenCL) and Message Passing Interface (MPI). This extension will allow our approach to run portably on various architectures and platforms.

Deployment of the approach: We aim to study the integration of our approach to recent distributed machine learning toolkits such as the Apache Mahout project and SystemML. Such integration can be done by the adaptation of our calculations to Mahout primitives [Foundation 2010] in the case of Mahout and to DML primitives [Ghoting 2011b] in the case of SystemML.

6.2.2 Second axis: improvement of cost models

As mentioned in Section 6.1.2, the second axis of this thesis consists of the design of cost models of mining pattern in a cloud setting. Improvements of the proposed cost models can be resumed into several points:

First, we aim to extend the experimental validation of the proposed cost models to a wider-scale experimentation. In this context, additional experiments will be carried out in which we solve the defined objective functions using more methods of solving multi-objective optimization problems. Several solving methods can be used:

- A priori methods: using these methods, the preferences of the decision maker are first asked and then a solution best satisfying these preferences is found.
- A posteriori methods: in this setting, a representative set of Pareto optimal solutions is first found and then the decision maker must choose one of them.
- Interactive methods: these methods allow the decision maker to iteratively search for the most preferred solution. In each iteration of the interactive method, the decision maker is shown Pareto optimal solutions and describes how the solutions could be improved.

Beside using more methods of solving multi-objective optimization problems, we aim to run experiments on a variable number of cloud cloud instances, thus, experimenting the effect of primordial elasticity characteristic of the cloud on our cost models.

Second, we plan to study the possibility of bringing the problem of reaching a tradeoff between the frequent subgraph mining process in the cloud under budget and mining quality constraints to a machine learning problem. This will allow the use of machine learning techniques in order to predict parameters and thresholds of the distributed mining process. A possible way to do this is to use a supervised classifier and to generate a classification model that allows prediction of parameters values for frequent subgraph mining process in the cloud.

Third, we aim to improve our cost models and generalize it to different cloud service providers. It is necessary to update our cost models in order to make our cost models compatible with different cloud service providers.

Bibliography

[Aggarwal 2010] Charu C. Aggarwal and Haixun Wang. Managing and mining graph data. Springer Publishing Company, Incorporated, 1st édition, 2010. (Cited on pages 15, 16 and 17.)

[Agrawal 1994] Rakesh Agrawal and Ramakrishnan Srikant. *Fast Algorithms for Mining Association Rules in Large Databases*. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. (Cited on pages 22 and 25.)

[Ahmed 2002] M.N. Ahmed, S.M. Yamany, N. Mohamed, A.A. Farag and T. Moriarty. *A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data*. Medical Imaging, IEEE Transactions on, vol. 21, no. 3, pages 193–199, 2002. (Cited on page 48.)

[Aridhi 2013] Sabeur Aridhi, Laurent d'Orazio, Mondher Maddouri and Engelbert Mephu Nguifo. *Density-based data partitioning strategy to approximate large-scale subgraph mining*. Information Systems, no. 0, pages –, 2013. (Cited on pages 4, 5 and 65.)

[Armbrust 2009] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin and Matei Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. 2009. (Cited on page 34.)

[Armbrust 2010] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia. *A view of cloud computing*. Commun. ACM, vol. 53, no. 4, pages 50–58, April 2010. (Cited on pages 2 and 33.)

[Auron 1982] Philip E. Auron, Wayne P. Rindone, Calvin P.H. Vary, James J. Celentano and John N. Vournakis. *Computer-aided prediction of RNA secondary structures*. Nucleic Acids Research, vol. 10, no. 1, pages 403–419, 1982. (Cited on page 19.)

[Bahmani 2012] Bahman Bahmani, Ravi Kumar and Sergei Vassilvitskii. *Densest subgraph in streaming and MapReduce*. volume 5, pages 454–465. VLDB Endowment, January 2012. (Cited on pages 54 and 55.)

[Barry 2013] Douglas K. Barry. Web services, service-oriented architectures, and cloud computing, second edition: The savvy manager's guide. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd édition, 2013. (Cited on page 42.)

[Blin 2010] Guillaume Blin, Florian Sikora and Stephane Vialette. *Querying Graphs in Protein-Protein Interactions Networks Using Feedback Vertex Set*. IEEE/ACM Trans. Comput. Biol. Bioinformatics, vol. 7, no. 4, pages 628–635, October 2010. (Cited on page 15.)

[Bonchi 2011] Francesco Bonchi, Carlos Castillo, Aristides Gionis and Alejandro Jaimes. *Social Network Analysis and Mining for Business Applications*. volume 2, pages 22:1–22:37, New York, NY, USA, May 2011. ACM. (Cited on pages 2 and 15.)

[Borgelt 2002] Christian Borgelt and Michael R. Berthold. *Mining Molecular Fragments: Finding Relevant Substructures of Molecules*. In Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02, pages 51–58, Washington, DC, USA, 2002. IEEE Computer Society. (Cited on pages 2 and 15.)

[Breiman 2001] Leo Breiman. *Random Forests*. Mach. Learn., vol. 45, no. 1, pages 5–32, October 2001. (Cited on page 47.)

[Chaoji 2008] Vineet Chaoji, Mohammad Al Hasan, Saeed Salem, Jeremy Besson and Mohammed J. Zaki. *ORIGAMI: A Novel and Effective Approach for*

Mining Representative Orthogonal Graph Patterns. volume 1, pages 67–84, New York, NY, USA, June 2008. John Wiley & Sons, Inc. (Cited on pages 3 and 25.)

[Cheng 1995] Yizong Cheng. *Mean Shift, Mode Seeking, and Clustering*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 17, no. 8, pages 790–799, August 1995. (Cited on page 48.)

[Cheng 2008] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang and Philip S. Yu. *Fast computing reachability labelings for large graphs with high compression rate*. In Proceedings of the 11th international conference on Extending database technology: Advances in database technology, EDBT '08, pages 193–204, New York, NY, USA, 2008. ACM. (Cited on page 16.)

[Chu 2006] Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng and Kunle Olukotun. *Map-Reduce for Machine Learning on Multicore*. In Bernhard Schölkopf, John C. Platt and Thomas Hoffman, editeurs, NIPS, pages 281–288. MIT Press, 2006. (Cited on page 89.)

[Ciurana 2009] Eugene Ciurana. Developing with google app engine. Apress, Berkely, CA, USA, 1 édition, 2009. (Cited on page 39.)

[Cohen 2009] Jonathan Cohen. *Graph Twiddling in a MapReduce World*. volume 11, pages 29–41. IEEE Computer Society, July 2009. (Cited on page 51.)

[Cook 1994] Diane J. Cook and Lawrence B. Holder. *Substructure discovery using minimum description length and background knowledge*. J. Artif. Int. Res., vol. 1, no. 1, pages 231–255, February 1994. (Cited on page 20.)

[Cook 2000] Diane J. Cook and Lawrence B. Holder. *Graph-Based Data Mining*. IEEE Intelligent Systems, vol. 15, no. 2, pages 32–41, March 2000. (Cited on page 20.)

[Coomans 1982] D. Coomans and D.L. Massart. *Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules*. Analytica Chimica Acta, vol. 136, no. 0, pages 15 – 27, 1982. (Cited on page 47.)

[de Menibus 2011] Benjamin Hellouin de Menibus and Takeaki Uno. *Maximal Matching and Path Matching Counting in Polynomial Time for Graphs of Bounded Clique Width*. In TAMC, pages 483–494, 2011. (Cited on page 16.)

[Dean 2008] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: simplified data processing on large clusters*. Commun. ACM, vol. 51, no. 1, pages 107–113, January 2008. (Cited on pages 4, 43 and 44.)

[Di Jorio 2009] Lisa Di Jorio, Anne Laurent and Maguelonne Teisseire. *Mining Frequent Gradual Itemsets from Large Databases*. In Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII, IDA '09, pages 297–308, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 49.)

[Di Jorio 2010] Lisa Di Jorio. *Recherche de motifs graduels et application aux données médicales*. These, Université Montpellier II - Sciences et Techniques du Languedoc, October 2010. (Cited on page 50.)

[Ehrgott 2005] Matthias Ehrgott. Multicriteria optimization. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. (Cited on page 104.)

[Eidson 2001] Eidson and Thomas M. Eidson. *A Component-based Programming Model for Composite, Distributed Applications*. Rapport technique, Institute for Computer, 2001. (Cited on page 43.)

[Erciyes 2013] Kayhan Erciyes. Distributed graph algorithms for computer networks. Springer Publishing Company, Incorporated, 2013. (Cited on page 19.)

[Ericson 2013] Kathleen Ericson and Shrideep Pallickara. *On the performance of high dimensional data clustering and classification algorithms*. Future Gener. Comput. Syst., vol. 29, no. 4, pages 1024–1034, June 2013. (Cited on page 48.)

[Erl 2004] Thomas Erl. Service-oriented architecture: A field guide to integrating xml and web services. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. (Cited on page 42.)

[Esteves 2011] Rui Maximo Esteves, Rui Pais and Chunming Rong. *K-means Clustering in the Cloud – A Mahout Test*. In Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, WAINA '11, pages 514–519, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on page 48.)

[Faloutsos 2004] Christos Faloutsos, Kevin S. Mccurley and Andrew Tomkins. *Connection subgraphs in social networks*. In Workshop on Link Analysis, Counterterrorism, and Privacy, SIAM International Conference on Data Mining, 2004. (Cited on pages 2, 3 and 15.)

[Fayyad 1997] Usama M. Fayyad. *Knowledge Discovery in Databases: An Overview*. In ILP, pages 3–16, 1997. (Cited on page 10.)

[Foundation 2010] Apache Software Foundation, Isabel Drost, Ted Dunning, Jeff Eastman, Otis Gospodnetic, Grant Ingersoll, Jake Mannix, Sean Owen and Karl Wettin. *Apache Mahout*, 2010. (Cited on pages 48, 50, 89 and 117.)

[Ghemawat 2003] Sanjay Ghemawat, Howard Gobioff and Shun T. Leung. *The Google file system*. In Proceedings of the 19th ACM symposium on Operating systems principles, volume 37 of *SOSP '03*, pages 29–43, New York, NY, USA, October 2003. ACM. (Cited on page 74.)

[Ghoting 2011a] Amol Ghoting, Prabhanjan Kambadur, Edwin Pednault and Ramakrishnan Kannan. *NIMBLE: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce*. In Proceedings

of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '11, pages 334–342, New York, NY, USA, 2011. ACM. (Cited on pages 46, 47, 50 and 89.)

[Ghoting 2011b] Amol Ghoting, Rajasekar Krishnamurthy, Edwin P. D. Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian and Shivakumar Vaithyanathan. *SystemML: Declarative machine learning on MapReduce.* In Serge Abiteboul, Klemens Böhm, Christoph Koch and Kian-Lee Tan, editeurs, ICDE, pages 231–242. IEEE Computer Society, 2011. (Cited on pages 48, 49, 50 and 117.)

[Goto 2002] S. Goto, Y. Okuno, M. Hattori, T. Nishioka and M. Kanehisa. *LIGAND: database of chemical compounds and reactions in biological pathways.* Nucleic Acids Res, vol. 30, no. 1, pages 402–404, January 2002. (Cited on pages 2 and 15.)

[Griffiths 1993] William E. Griffiths, R. Carter Hill and George G. Judge. Learning and practicing econometrics. Wiley, New York, NY [u.a.], 1993. (Cited on page 90.)

[Han 2000] Jiawei Han and Jian Pei. *Mining frequent patterns by pattern-growth: methodology and implications*. SIGKDD Explor. Newsl., vol. 2, no. 2, pages 14–20, December 2000. (Cited on page 47.)

[Hill 2012] Steven Hill, Bismita Srichandan and Rajshekhar Sunderraman. *An iterative MapReduce approach to frequent subgraph mining in biological datasets*. In Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine, BCB '12, pages 661–666, New York, NY, USA, 2012. ACM. (Cited on pages 3, 53 and 55.)

[Huan 2003a] Jun Huan, Wei Wang and Jan Prins. *Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism*. In Proceedings of the Third IEEE International Conference on Data Mining, ICDM '03, pages 549–, Washington, DC, USA, 2003. IEEE Computer Society. (Cited on page 24.)

[Huan 2003b] Jun Huan, Wei Wang and Jan Prins. *Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism*. In Proceedings of the Third IEEE International Conference on Data Mining, ICDM '03, pages 549–, Washington, DC, USA, 2003. IEEE Computer Society. (Cited on page 69.)

[Huan 2004] Jun Huan, Wei Wang and Jan Prins. *Spin: Mining maximal frequent subgraphs from graph databases*. In In KDD, pages 581–586, 2004. (Cited on page 25.)

[Inc 2013a] Amazon Inc. Amazon ec2 offer, 2013. (Cited on pages 34 and 39.)

[Inc 2013b] Google Inc. Google cloud platform, 2013. (Cited on page 34.)

[Inc 2013c] Microsoft Inc. Windows azure offer, 2013. (Cited on pages 34 and 39.)

[Jiang 2013] Chuntao Jiang, Frans Coenen and Michele Zito. *A survey of frequent subgraph mining algorithms*. Knowledge Eng. Review, vol. 28, no. 1, pages 75–105, 2013. (Cited on page 21.)

[Kang 2008] U Kang, Charalampos Tsourakakis, Ana Paula Appel, Christos Faloutsos and Jure Leskovec. *HADI: Fast Diameter Estimation and Mining in Massive Graphs with Hadoop*, 2008. (Cited on pages 51 and 55.)

[Kang 2009] U. Kang, Charalampos E. Tsourakakis and Christos Faloutsos. *PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations*. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, ICDM '09, pages 229–238, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on pages 51 and 55.)

[Kang 2013] U. Kang and Christos Faloutsos. *Big graph mining: algorithms and discoveries*. SIGKDD Explor. Newsl., vol. 14, no. 2, pages 29–36, April 2013. (Cited on page 51.)

[Kashef 2012] Mohammad Mahdi Kashef and Jörn Altmann. *A cost model for hybrid clouds*. In Proceedings of the 8th international conference on Economics of Grids, Clouds, Systems, and Services, GECON'11, pages 46–60, Berlin, Heidelberg, 2012. Springer-Verlag. (Cited on page 4.)

[Kijima 2012] Shuji Kijima, Yota Otachi, Toshiki Saitoh and Takeaki Uno. *Subgraph isomorphism in graph classes*. Discrete Mathematics, vol. 312, no. 21, pages 3164–3173, 2012. (Cited on page 26.)

[Krishnaswamy 2004] S. Krishnaswamy, S. W. Loke and A. Zaslasvky. *A hybrid model for improving response time in distributed data mining*. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 34, no. 6, pages 2466–2479, 2004. (Cited on pages 4, 89, 92 and 93.)

[Kuramochi 2001] Michihiro Kuramochi and George Karypis. *Frequent Subgraph Discovery*. In Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society. (Cited on pages 3, 25, 26 and 73.)

[Kwon 2012] YongChul Kwon, Magdalena Balazinska, Bill Howe and Jerome A. Rolia. *SkewTune: mitigating skew in mapreduce applications*. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pages 25–36, 2012. (Cited on pages 54 and 69.)

[Laurent 2012] Anne Laurent, Benjamin Négrevergne, Nicolas Sicard and Alexandre Termier. *Efficient Parallel Mining of Gradual Patterns on Multicore Processors*. In Fabrice Guillet, Gilbert Ritschard and Djamel Abdelkader Zighed, editeurs, Advances in Knowledge Discovery and Management, volume 398 of *Studies in Computational Intelligence*, pages 137–151. Springer Berlin Heidelberg, 2012. (Cited on page 49.)

[Li 2012] Geng Li, Murat Semerci, Bülent Yener and Mohammed J. Zaki. *Effective graph classification based on topological and label attributes*. Statistical Analysis and Data Mining, vol. 5, no. 4, pages 265–283, 2012. (Cited on page 116.)

[Liu 2009] Yang Liu, Xiaohong Jiang, Huajun Chen, Jun Ma and Xiangyu Zhang. *MapReduce-Based Pattern Finding Algorithm Applied in Motif Detection for Prescription Compatibility Network*. In Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies, APPT

'09, pages 341–355, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on pages 52, 54 and 55.)

[Luo 2011] Yifeng Luo, Jihong Guan and Shuigeng Zhou. *Towards efficient subgraph search in cloud computing environments*. In Proceedings of the 16th international conference on Database systems for advanced applications, DASFAA'11, pages 2–13, Berlin, Heidelberg, 2011. Springer-Verlag. (Cited on pages 3, 54 and 55.)

[MacQueen 1967] J. B. MacQueen. *Some Methods for Classification and Analysis of MultiVariate Observations*. In L. M. Le Cam and J. Neyman, editeurs, Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press, 1967. (Cited on pages 47 and 48.)

[Malewicz 2010] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser and Grzegorz Czajkowski. *Pregel: a system for large-scale graph processing*. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM. (Cited on pages 51, 54 and 55.)

[Marbán 2008] Oscar Marbán, Ernestina Menasalvas and Covadonga Fernández-Baizán. *A cost model to estimate the effort of data mining projects (DMCoMo)*. Information Systems, vol. 33, no. 1, pages 133–150, March 2008. (Cited on pages 4, 89, 92 and 93.)

[Middleton 2013] N. Middleton and R. Schneeman. Heroku. O'Reilly Media, Incorporated, 2013. (Cited on page 39.)

[Miyashita 1989] Yoshikatsu Miyashita, Masami Ishikawa and Shin-Ichi Sasaki. *Classification of brandies by pattern recognition of chemical data*. Journal of the Science of Food and Agriculture, vol. 49, no. 3, pages 325–333, 1989. (Cited on page 18.)

[Négrevergne 2010] Benjamin Négrevergne, Jean-François Méhaut, Alexandre Termier and Takeaki Uno. *Découverte d'itemsets fréquents fermés sur architecture multicoeurs*. In EGC, pages 465–470, 2010. (Cited on page 49.)

[Negrevergne 2013] Benjamin Negrevergne, Alexandre Termier, Marie-Christine Rousset and Jean-François Méhaut. *Para Miner: a generic pattern mining algorithm for multi-core architectures*. Data Mining and Knowledge Discovery, pages 1–41, 2013. (Cited on page 49.)

[Nguyen 2012] Thi-Van-Anh Nguyen, Sandro Bimonte, Laurent d'Orazio and Jérôme Darmont. *Cost models for view materialization in the cloud*. In Proceedings of the 2012 Joint EDBT/ICDT Workshops, EDBT-ICDT '12, pages 47–54, New York, NY, USA, 2012. ACM. (Cited on pages 4, 89, 92 and 99.)

[Nijssen 2004] Siegfried Nijssen and Joost N. Kok. *A quickstart in frequent structure mining can make a difference*. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '04, pages 647–652, New York, NY, USA, 2004. ACM. (Cited on pages 3, 25 and 26.)

[Nykiel 2010] Tomasz Nykiel, Michalis Potamias, Chaitanya Mishra, George Kollios and Nick Koudas. *MRShare: sharing across multiple queries in MapReduce*. Proc. VLDB Endow., vol. 3, no. 1-2, pages 494–505, September 2010. (Cited on pages 91, 92 and 93.)

[Ogunde 2011] A. O. Ogunde, O. Folorunso, A. S. Sodiya, J. A. Oguntuase and G. O. Ogunleye. *Improved cost models for agent-based association rule mining in distributed databases*. In Annals. Computer Science Series, 9th Tome - 1st Fasc., pages 231–251, 2011. (Cited on pages 4, 89, 90, 92 and 93.)

[Pearson 1901] K. Pearson. *On lines and planes of closest fit to systems of points in space*. Philosophical Magazine, vol. 2, no. 6, pages 559–572, 1901. (Cited on page 48.)

[Pizzuti 2012] Clara Pizzuti, Simona E. Rombo and Elena Marchiori. *Complex detection in protein-protein interaction networks: a compact overview for researchers and practitioners*. In Proceedings of the 10th European conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, EvoBIO'12, pages 211–223, Berlin, Heidelberg, 2012. Springer-Verlag. (Cited on pages 2 and 15.)

[Qiu 2008] Xiaohong Qiu, Geoffrey Fox, Huapeng Yuan, Seung-Hee Bae, George Chrysanthakopoulos and Henrik Nielsen. *Parallel Data Mining on Multicore Clusters*. In Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing, GCC '08, pages 41–49, Washington, DC, USA, 2008. IEEE Computer Society. (Cited on page 49.)

[Quiane-Ruiz 2011] Jorge-Arnulfo Quiane-Ruiz, Christoph Pinkel, Jorg Schad and Jens Dittrich. *RAFTing MapReduce: Fast recovery on the RAFT*. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, ICDE '11, pages 589–600, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on page 116.)

[Ranu 2012] Sayan Ranu and Ambuj K. Singh. *Indexing and mining topological patterns for drug discovery*. In Proceedings of the 15th International Conference on Extending Database Technology, EDBT '12, pages 562–565, New York, NY, USA, 2012. ACM. (Cited on page 18.)

[Riondato 2012] Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca and Eli Upfal. *PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce*. In Proceedings of the 21st ACM international conference on Information and knowledge management, CIKM '12, pages 85–94, New York, NY, USA, 2012. ACM. (Cited on pages 4 and 89.)

[Saidi 2010] Rabie Saidi, Mondher Maddouri and Engelbert Mephu Nguifo. *Protein sequences classification by means of feature extraction with substitution matrices*. BMC Bioinformatics, vol. 11, page 175, 2010. (Cited on page 19.)

[Saidi 2012] Rabie Saidi, Sabeur Aridhi, Engelbert Mephu Nguifo and Mondher Maddouri. *Feature extraction in protein sequences classification: a new stability measure*. In Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine, BCB '12, pages 683–689, New York, NY, USA, 2012. ACM. (Cited on pages 2, 15 and 19.)

[Sicard 2010] Nicolas Sicard, Anne Laurent, Federico Del Razo López and Perfecto Malaquias Quintero Flores. *Towards multi-core parallel fuzzy tree mining*. In FUZZ-IEEE, pages 1–7, 2010. (Cited on page 50.)

[Speičys 2008] Laurynas Speičys and Christian S. Jensen. *Enabling Location-based Services–Multi-Graph Representation of Transportation Networks*. Geoinformatica, vol. 12, no. 2, pages 219–253, June 2008. (Cited on page 15.)

[Suryawanshi 2013] Sujata J. Suryawanshi and S. M. Kamalapur. *Algorithms for Frequent Subgraph Mining*. International Journal of Advanced Research in Computer and Communication Engineering, vol. 2, no. 3, pages 1545–1548, 2013. (Cited on page 21.)

[Tejedor 2011] Enric Tejedor, Jorge Ejarque, Francesc Lordan, Roger Rafanell, Javier Alvarez, Daniele Lezzi, Raul Sirvent and Rosa M. Badia. *A Cloud-unaware Programming Model for Easy Development of Composite Services*. In Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, pages 375–382, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on page 43.)

[Thomas 2010] Lini T. Thomas, Satyanarayana R. Valluri and Kamalakar Karlapalem. *MARGIN: Maximal frequent subgraph mining*. ACM Trans. Knowl. Discov. Data, vol. 4, no. 3, pages 10:1–10:42, October 2010. (Cited on page 25.)

[Tsourakakis 2009] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller and Christos Faloutsos. *DOULION: counting triangles in massive graphs with a coin*. In Proceedings of the 15th ACM SIGKDD international conference

on Knowledge discovery and data mining, KDD '09, pages 837–846, New York, NY, USA, 2009. ACM. (Cited on pages 51, 54 and 55.)

[Wang 2006] Haixun Wang, Hao He2, Jun Yang, Philip S. Yu and Jeffrey Xu Yu. *Dual Labeling: Answering Graph Reachability Queries in Constant Time*. In Proceedings of the 22nd International Conference on Data Engineering, ICDE '06, pages 75–, Washington, DC, USA, 2006. IEEE Computer Society. (Cited on page 16.)

[Wegner 2012] Joerg Kurt Wegner, Aaron Sterling, Rajarshi Guha, Andreas Bender, Jean-Loup Faulon, Janna Hastings, Noel O'Boyle, John Overington, Herman Van Vlijmen and Egon Willighagen. *Cheminformatics*. Commun. ACM, vol. 55, no. 11, pages 65–75, November 2012. (Cited on page 18.)

[Weisberg 1985] Sanford Weisberg. Applied linear regression. Wiley series in probability and mathematical statistics. Wiley, New York [u.a.], 2. ed édition, 1985. (Cited on page 90.)

[Woollen 2010] Rob Woollen. *The internal design of salesforce.com's multi-tenant architecture*. In Proceedings of the 1st ACM symposium on Cloud computing, SoCC '10, pages 161–161, New York, NY, USA, 2010. ACM. (Cited on page 39.)

[Wörlein 2005] Marc Wörlein, Thorsten Meinl, Ingrid Fischer and Michael Philippsen. *A quantitative comparison of the subgraph miners mofa, gspan, FFSM, and gaston*. In Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases, PKDD'05, pages 392–403, Berlin, Heidelberg, 2005. Springer-Verlag. (Cited on pages 3 and 69.)

[Wu 2010] Bin Wu and YunLong Bai. *An efficient distributed subgraph mining algorithm in extreme large graphs*. In Proceedings of the 2010 international conference on Artificial intelligence and computational intelligence: Part I, AICI'10, pages 107–115, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on pages 53, 54 and 55.)

[Yan 2002] Xifeng Yan and Jiawei Han. *gSpan: Graph-Based Substructure Pattern Mining*. In Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02, pages 721–724, Washington, DC, USA, 2002. IEEE Computer Society. (Cited on pages 3, 23, 24, 25 and 26.)

[Yan 2003] Xifeng Yan and Jiawei Han. *CloseGraph: mining closed frequent graph patterns*. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '03, pages 286–295, New York, NY, USA, 2003. ACM. (Cited on pages 3 and 24.)

[Yang 2010] Christopher Yang, Christine Yen, Ceryen Tan and Samuel R. Madden. *Osprey: Implementing MapReduce-style fault tolerance in a shared-nothing distributed database*. ICDE '10, pages 657–668, Los Alamitos, CA, USA, 2010. IEEE Computer Society. (Cited on page 116.)

[Ye 2005] Jieping Ye. *Dimension reduction algorithms in data mining, with applications*. PhD thesis, Minneapolis, MN, USA, 2005. AAI3172868. (Cited on page 48.)

[YongChul K. 2011] Magdalena B. YongChul K. *A Study of Skew in MapReduce Applications*. In Proceedings of the 5th Open Cirrus Summit, 2011. (Cited on pages 54 and 56.)

[Zhang 2004] Harry Zhang. *The Optimality of Naive Bayes.* In Valerie Barr and Zdravko Markov, editeurs, FLAIRS Conference. AAAI Press, 2004. (Cited on page 48.)

[Zhao 2007] Yonghua Zhao, Xuebin Chi and Qiang Cheng. *An implementation of parallel eigenvalue computation using dual-level hybrid parallelism*. In Proceedings of the 7th international conference on Algorithms and architectures for parallel processing, ICA3PP'07, pages 107–119, Berlin, Heidelberg, 2007. Springer-Verlag. (Cited on pages 51 and 55.)

[Zlochin 2004] Mark Zlochin, Mauro Birattari, Nicolas Meuleau and Marco Dorigo. *Model-Based Search for Combinatorial Optimization: A Critical*

Survey. Annals of Operations Research, vol. 131, no. 1-4, pages 373–395, 2004. (Cited on pages 103 and 104.)

[Zuker 1984] Michael Zuker and David Sankoff. *RNA secondary structures and their prediction*. Bulletin of Mathematical Biology, vol. 46, no. 4, pages 591–621, 1984. (Cited on page 19.)

Distributed Frequent Subgraph Mining in the Cloud

Abstract: Recently, graph mining approaches have become very popular, especially in certain domains such as bioinformatics, chemoinformatics and social networks. One of the most challenging tasks in this setting is frequent subgraph discovery. This task has been highly motivated by the tremendously increasing size of existing graph databases. Due to this fact, there is urgent need of efficient and scaling approaches for frequent subgraph discovery especially with the high availability of cloud computing environments. This thesis deals with distributed frequent subgraph mining in the cloud. First, we provide the required material to understand the basic notions of our two research fields, namely graph mining and cloud computing. Then, we present the contributions of this thesis.

In the first axis, we propose a novel approach for large-scale subgraph mining, using the MapReduce framework. The proposed approach provides a data partitioning technique that consider data characteristics. It uses the densities of graphs in order to partition the input data. Such a partitioning technique allows a balanced computational loads over the distributed collection of machines and replace the default arbitrary partitioning technique of MapReduce. We experimentally show that our approach decreases significantly the execution time and scales the subgraph discovery process to large graph databases.

In the second axis, we address the multi-criteria optimization problem of tuning thresholds related to distributed frequent subgraph mining in cloud computing environments while optimizing the global monetary cost of storing and querying data in the cloud. We define cost models for managing and mining data with a large scale subgraph mining framework over a cloud architecture. We present an experimental validation of the proposed cost models in the case of distributed subgraph mining in the cloud.

Keywords: frequent subgraph mining, graph partitioning, graph density, MapReduce, cloud computing, cost models.

Fouille de sous-graphes fréquents dans les nuages

Résumé: Durant ces dernières années, l'utilisation de graphes a fait l'objet de nombreux travaux, notamment en bases de données, apprentissage automatique, bioinformatique et en analyse des réseaux sociaux. Particulièrement, la fouille de sous-graphes fréquents constitue un défi majeur dans le contexte de très grandes bases de graphes. De ce fait, il y a un besoin d'approches efficaces de passage à l'échelle pour la fouille de sous-graphes fréquents surtout avec la haute disponibilité des environnements de cloud computing. Cette thèse traite la fouille distribuée de sous-graphe fréquents sur cloud. Tout d'abord, nous décrivons le matériel nécessaire pour comprendre les notions de base de nos deux domaines de recherche, à savoir la fouille de sous-graphe fréquents et le cloud computing. Ensuite, nous présentons les contributions de cette thèse.

Dans le premier axe, une nouvelle approche basée sur le paradigme MapReduce pour approcher la fouille de sous-graphes fréquents à grande échelle. L'approche proposée offre une nouvelle technique de partitionnement qui tient compte des caractéristiques des données et qui améliore le partitionnement par défaut de MapReduce. Une telle technique de partitionnement permet un équilibrage des charges de calcul sur une collection de machine distribuée et de remplacer la technique de partitionnement par défaut de MapReduce. Nous montrons expérimentalement que notre approche réduit considérablement le temps d'exécution et permet le passage à l'échelle du processus de fouille de sous-graphe fréquents à partir de grandes bases de graphes.

Dans le deuxième axe, nous abordons le problème d'optimisation multi-critères des paramètres liés à l'extraction distribuée de sous-graphes fréquents dans un environnement de cloud tout en optimisant le coût monétaire global du stockage et l'interrogation des données dans le nuage. Nous définissons des modèles de coûts de gestion et de fouille de données avec une plateforme de fouille de sous-graphe à grande échelle sur une architecture cloud. Nous présentons une première validation expérimentale des modèles de coûts proposés.

Mots-clés: Fouille de sous-graphes, partitionnement de graphes, densité de graphe, MapReduce, Informatique dans les nuages, modèles de coûts.