Information Systems 48 (2015) 213–223

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Density-based data partitioning strategy to approximate large-scale subgraph mining

Sabeur Aridhi [a,b,c,*], Laurent d'Orazio [a,b], Mondher Maddouri [c,d], Engelbert Mephu Nguifo [a,b,*]

[a] Clermont University, Blaise Pascal University, LIMOS, BP 10448, F-63000 Clermont-Ferrand, France
[b] CNRS, UMR 6158, LIMOS, F-63173 Aubiere, France
[c] University of Tunis El Manar, LIPAH – FST, Academic Campus, Tunis 2092, Tunisia
[d] Taibah University, Almadinah, Kingdom of Saudi Arabia

ARTICLE INFO

Available online 14 September 2013

Keywords:
Frequent subgraph mining
Graph partitioning
Graph density
MapReduce
Cloud computing

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 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 an urgent need of efficient and scaling approaches for frequent subgraph discovery. In this paper, we propose a novel approach for large-scale subgraph mining by means of a density-based partitioning technique, using the MapReduce framework. Our partitioning aims to balance computational load on a collection of machines. We experimentally show that our approach decreases significantly the execution time and scales the subgraph discovery process to large graph databases.

1. Introduction

Graphs show up in diverse set of disciplines, ranging from computer networks, social networks to bioinformatics, chemoinformatics and others. These fields exploit the representation power of graph format to describe their associated data, e.g., social networks consist of individuals and their relationships. In bioinformatics, the protein structure can be considered as a graph where nodes represent the amino acids and edges represent the interactions between them. 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, junction sites. Moreover, in a social network, frequent substructures can help to identify the few most likely paths of transmission for a rumor or joke from one person to another [1]. 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 [2], Gaston [3] and gSpan [4]. However, existing approaches are mainly used on centralized computing systems and evaluated on relatively small databases [5]. Nowadays, there is an exponential growth in both the graph size and the number of graphs in databases, which makes the above cited approaches face the scalability issue. Several parallel or distributed solutions have been proposed for frequent subgraph mining on a single large graph [6–9]. However, the problem of subgraph mining from large-scale graph databases is still challenging.

In this paper, we propose a scalable and distributed approach for large scale frequent subgraph mining based

* Corresponding author at: Clermont University, Blaise Pascal University, LIMOS, BP 10448, F-63000 Clermont-Ferrand, France. Tel.: +33 473407629.
E-mail addresses: aridhi@isima.fr (S. Aridhi), dorazio@isima.fr (L. d'Orazio), mondher.maddouri@fst.rnu.tn (M. Maddouri), mephu@isima.fr (E. Mephu Nguifo).

0306-4379/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2013.08.005

on MapReduce framework [10]. Our approach is not the first one to use MapReduce to solve the distributed frequent subgraph mining task, it differs from and enhances previous works in two crucial aspects. First, previous attempts try to construct the final set of frequent subgraphs iteratively using MapReduce, possibly resulting in a big number of MapReduce passes and an exponential growth of intermediate data especially with large-scale datasets. On the contrary, our approach mines the final set of frequent subgraphs from different partitions of the original dataset by a unique execution of a subgraph mining algorithm. In addition, it offers the possibility to apply any of the known subgraph mining algorithms in a distributed way. Second, our approach differs from previous algorithms by providing a density-based data partitioning technique of the input data. In previous works, the default MapReduce partitioning technique was used to partition the input data, which can be the origin of imbalanced computational load among map tasks [11].

The contributions of this paper are as follows:

- 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.

This paper is organized as follows. In the next section, we define the problem of large-scale subgraph mining. In Section 3, we present our approach of large-scale subgraph mining with MapReduce. Then, we describe our experimental study and we discuss the obtained results in Section 4. Finally, in Section 5, we present an overview of some related works dealing with the concept of large scale subgraph mining.

2. Problem definition

In this section, we present definitions and notations used in this paper. Then, we present the MapReduce framework. Finally, we define the problem we are addressing and specify our assumptions.

2.1. Definitions

A graph is a collection of objects denoted as $G=(V,E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. A graph G' is a subgraph of another graph G, if there exists a subgraph isomorphism from G' to G, denoted as $G' \subseteq G$. The definitions of subgraph and subgraph isomorphism are given as follows.

Definition 1 (*Subgraph*). A graph $G'=(V',E')$ is a subgraph of another graph $G=(V,E)$ iff $V' \subseteq V$ and $E' \subseteq E$.

Definition 2 (*Graph and subgraph 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. A graph G' has a subgraph isomorphism with G if:

- G' is a subgraph of G, and
- there exists an isomorphism between G' and G.

A task of major interest in this setting is frequent subgraph mining (FSM) with respect to a minimum support threshold. 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, comprises 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 3 (*Subgraph relative support*). Given a graph database $DB=\{G_1,\ldots,G_K\}$, the relative support of a subgraph G' is defined by

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

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 4 (*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 $SG(DB,\theta)=\{(A, Support(A,DB)) : A \text{ is a subgraph of } DB \text{ and } Support(A, DB) \geq \theta\}$.

Definition 5 (*Graph density*). 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 said to be sparse if the ratio is close to 0. The density of a graph $G=(V,E)$ is calculated by

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

2.2. MapReduce

MapReduce is a framework for processing highly distributable problems across huge datasets using a large number of computers. 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. We notice that the data distribution technique of MapReduce consists of the decomposition of the input data into equal-size partitions called

chunks. The only criterion required in this partitioning method is the size of the partition, which corresponds to the size of chunk in the MapReduce configuration.

MapReduce is an abstraction that allows Google engineers to perform simple computations while hiding the details of parallelization and data distribution [10]. The central features of the MapReduce framework are two functions, written by a user: *Map* and *Reduce*. The *Map* function processes a *key/value* pair to generate a set of intermediate *key/value* pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function accepts an intermediate key and a set of values for that key. It merges these values together to form a smaller set of values.

2.3. Problem formulation

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 6 (*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 support threshold (*GS*).

Definition 7 (*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 support threshold (*LS*).

Definition 8 (*Loss rate*). Given S_1 and S_2 two sets with $S_2 \subseteq S_1$ and $S_1 \neq \emptyset$, we define the loss rate in S_2 compared to S_1 by

$$LossRate(S_1, S_2) = \frac{|S_1 - S_2|}{|S_1|}, \tag{2}$$

We define the problem of distributed subgraph mining by finding a good partitioning of the database over SM and by minimizing well defined approximation of $SG(DB, \theta)$.

Definition 9 (*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

$$LossRate(SG, S) \leq \varepsilon. \tag{3}$$

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 10 (*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:

$$\mu = E[R]. \tag{4}$$

The cost measure of a partitioning technique is

$$Cost(PM) = \sqrt{E[(R-\mu)^2]}. \tag{5}$$

A large cost value indicates that the runtime values are far from the mean value and a small cost value indicates that the runtime values are near the mean value. The smaller the value of the cost is, the more efficient the partitioning is.

3. Density-based partitioning for large-scale subgraph mining with MapReduce

In this section, we present the proposed approach for large scale subgraph mining with MapReduce. It first describes the proposed framework to approximate large-scale frequent subgraph mining. Then, it presents our density-based partitioning technique.

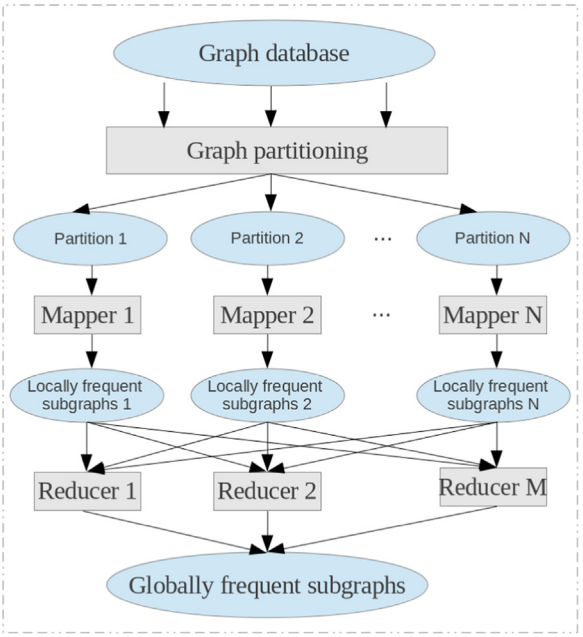

Fig. 1. A system overview of our approach. In this figure, ellipses represent data, squares represent computations on the data and arrows show the flow of data.

3.1. 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 Fig. 1).

As shown in Fig. 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 local support. Mapper i outputs *key/value* 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 *key/value* pairs, the reducer outputs the final list of *key/value* pairs after filtering according to the global support $\langle subgraph, Support(subgraph, DB)\rangle$.

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

Data partitioning. In this step, the input database is partitioned into N partitions. 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 two partitioning techniques for the graph database. 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 different partitioning technique taking into account the characteristics of the input data during the creation of partitions. We termed it Density-based Graph Partitioning (shortly called DGP). More precisely, DGP tends to balance graph density distribution in each partition (for more details, see Section 3.2).

Distributed subgraph mining. In this phase, we use a frequent subgraph mining technique that we run on each partition in parallel. Algorithms 1 and 2 present our *Map* and *Reduce* functions respectively:

Algorithm 1. Map function.

Require A partitioned graph database
$DB=\{Part_1(DB),\ldots,Part_N(DB)\}$, 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: $EmitIntermediate(s, Support(s, Part_i(DB)))$
4: **end for**

Algorithm 2. Reduce function.

Require 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: $GlobalSupportCount \leftarrow GlobalSupportCount + v$
4: **end for**
5: $GlobalSupport \leftarrow \frac{GlobalSupportCount}{N}$
6: **if** $GlobalSupport >= \theta$ **then**
7: $Emit(s, GlobalSupport)$
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.

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 means a zero value of $\varepsilon(\varepsilon=0)$. 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. Let $DB=\{G_1,\ldots,G_K\}$ 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 $Part(DB)=\{Part_1(DB),\ldots,Part_N(DB)\}$ 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 $s_i \le LS$, for all $i\in[1,N]$. Thus, $\sum_{i=1}^{N} s_i / N \le \sum_{i=1}^{N} LS/N$ and therefore, $s \le LS$. We have $LS=(1-\tau)\cdot\theta$ and therefore $LS\le\theta$, for all $\tau\in[0,1]$. Thus, we have $s\le\theta$ and so G' is not globally frequent, contradicting our condition.

3.2. The density-based graph partitioning method

3.2.1. Motivation and principle

The motivation behind dividing the input data into partitions is to reduce effectively 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 [11]. Considering the fact that the task of frequent subgraph mining depends on the density of graphs [5,12], 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. Fig. 2 gives an overview of the proposed partitioning method.

The proposed partitioning technique can be resumed into 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 *DensitiesComputing* that compute the densities of all instances in the database. Algorithm 3 presents the *Map* function of this step. 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 *Reduce* function 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.

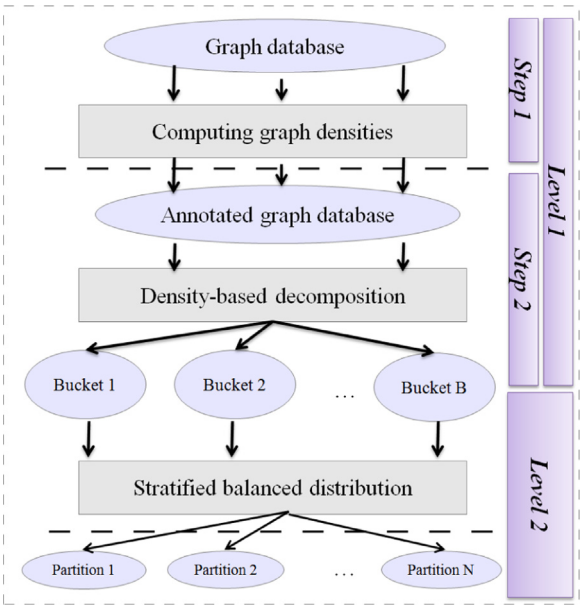

Fig. 2. The DGP method. In this figure, ellipses represent data, squares represent computations on the data and arrows show the flow of data.

Table 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

Algorithm 3. Map function of *DensitiesComputing*.

Require A graph database $DB = \{G_1, \ldots, G_K\}$
Ensure Annotated 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 second level of our partitioning method is to construct the output partitions. To do this, we first divide each bucket into N sub-partitions $B_i = \{P_{i1}, \ldots, P_{iN}\}$. Then, we construct the output partitions. Each one is constructed by appending one sub-partition from each bucket.

3.2.2. Illustrative example

Here, we give an illustrative example to explain the principle of the two partitioning techniques. Given a graph database of 12 graphs $DB = \{G_1, \ldots, G_{12}\}$. Table 1 presents the size on disk and the density of each graph in the database.

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:

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

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 $B_1 = \{G_1, G_4, G_2, G_5, G_6, G_7\}$ 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 Fig. 3).

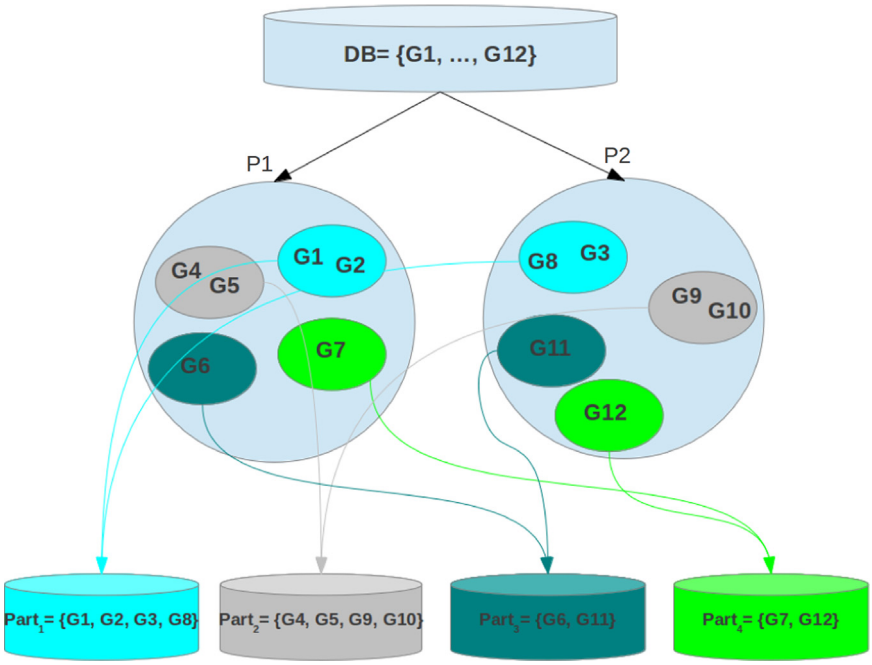

Fig. 3. Example of DGP method.

Table 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]

As shown in Fig. 3, each bucket contains a balanced set of graphs from B_1 and B_2. The final set of partitions will be

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

4. 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.

4.1. Experimental setup

4.1.1. Datasets

The datasets used in our experimental study are described in Table 2.

The synthetic datasets are generated by the synthetic data generator *GraphGen* provided by Kuramochi and Karypis [2]. For our tests, we generate various synthetic datasets 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 we tested is a chemical compound dataset which is available from the Developmental Therapeutics Program (DTP) at National Cancer Institute (NCI).[1]

4.1.2. Subgraph extractors

Three existing subgraph miners were used in our experiments: gSpan, FSG and Gaston.

gSpan is an algorithm for frequent graph-based pattern mining in graph datasets based on DFS lexicographic order and its properties [4]. gSpan discovers frequent substructures without candidate generation, which aims to avoid the candidate generation phase and the subgraph isomorphism test. Based on DFS codes, a hierarchical search tree is constructed. By pre-order traversal of the tree, gSpan discovers all frequent subgraphs with required support (threshold).

FSG uses the Apriori level-wise approach [13] to find frequent subgraphs. It uses a sparse graph representation in order to improve both storage and computation. 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 [2].

Gaston 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 graphs [3].

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

4.1.3. Implementation platform

We implemented our approach in Perl language and we used Hadoop, (version 0.20.1), an open source version of MapReduce. The databases files are stored in the Hadoop Distributed File System (HDFS), an open source implementation of GFS [14].

All the experiments of our approach were carried out using a local cluster with five nodes. 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.

4.2. Experimental results

4.2.1. Accuracy and speedup

Table 3 shows the obtained results using the sequential version of the used subgraph extractors.

Table 4 shows the obtained results obtained using our proposed approach with the default MapReduce partitioning technique and those obtained with the density-based partitioning technique with two buckets. For each dataset and support value, we note the results of the classic subgraph mining algorithm and those of the proposed method.

We mention that we could not conduct our experiment with the sequential algorithms in the case of *DS*4, *DS*5 and *DS*6 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 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). For example, in Table 4, for *DS*1 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, 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 Fig. 4 the effect of the proposed partitioning methods on the rate of lost subgraphs.

We can easily see in Fig. 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 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 diminution of 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 terms of density. Consequently, this balanced distribution of the data provides an effective load balancing scheme for distributed computations over worker nodes. Fig. 5 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. Fig. 6 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 Figs. 5 and 6, the density-based partitioning method allows a balanced distribution of workload across the distributed worker nodes especially with high number of buckets.

In order to evaluate the capability of the density-based partitioning method to balance the computations over the used nodes, we show in Fig. 7 the cost of this partitioning method in comparison with the MapReduce-based partitioning method. In addition, we show in Fig. 8 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.

As shown in Figs. 7 and 8, 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

Table 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	94	17
	50	38	106	35	61	35	9

Table 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	39	198	179	69
		0.3	227	207	68	364	344	79
		0.6	312	352	79	371	351	79
	50	0	17	0	0	23	6	1
		0.3	41	23	7	41	23	7
		0.6	41	23	7	41	23	7
DS2	30	0	145	124	31	146	125	38
		0.3	156	136	31	156	136	39
		0.6	156	136	31	156	136	39
	50	0	25	7	0	25	7	2
		0.3	26	9	0	26	9	3
		0.6	26	9	0	26	9	3
DS3	30	0	77	70	70	80	77	42
		0.3	97	92	80	88	93	42
		0.6	97	93	72	97	93	69
	50	0	36	31	29	37	32	29
		0.4	38	35	29	38	35	29
		0.6	38	35	29	38	35	29
DS4	30	0	137	116	19	78	117	19
		0.3	155	135	19	78	135	19
		0.6	155	135	19	155	135	20
	50	0	24	6	0	24	6	0
		0.3	26	9	0	26	9	0
		0.6	26	9	0	26	9	0
DS5	30	0	131	121	9	104	118	0
		0.3	155	135	9	104	135	0
		0.6	155	135	9	155	135	0
	50	0	24	7	0	18	6	0
		0.3	26	9	0	18	9	0
		0.6	26	9	0	18	9	0
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

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 smaller runtime than gSpan (see Fig. 7). 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 Fig. 9 the runtime of our approach for each number of mapper machines.

As illustrated in Fig. 9, 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.

4.2.2. 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 10 MB to 100 MB. Secondly, we varied the number of copies of data and calculated the runtime of the distributed subgraph mining process.

The experimentations presented in Fig. 10 show that with small values of chunk size and with big datasets, the runtime of our approach is very important. Otherwise, the other values of chunk size do not notably affect the results.

As shown in Fig. 11, 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.

5. Related work

Subgraph mining algorithms consist of two groups, the Apriori-based algorithms and the non-Apriori-based algorithms (or pattern growth approaches). The Apriori approach shares similar characteristics with the Apriori-based itemset mining [13]. AGM [15] and FSG [2] are two

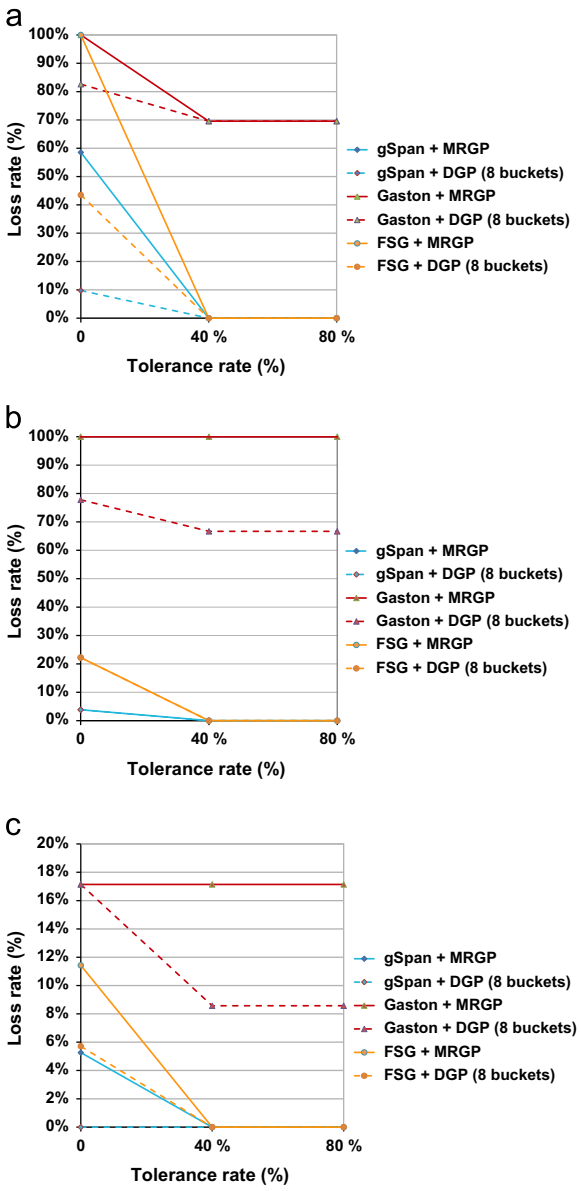

Fig. 4. Effect of the partitioning method on the rate of lost subgraphs. (a) DS1, (b) DS2, and (c) DS3.

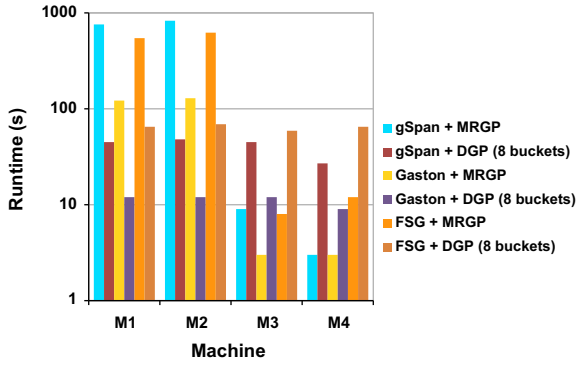

Fig. 5. Effect of the partitioning method on the distribution of computations. We used $\theta = 30\%$ and $\tau = 0.3$.

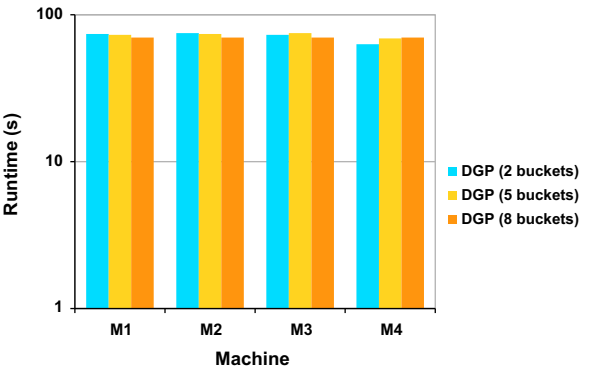

Fig. 6. Effect of the number of buckets of the density-based partitioning method on the distribution of computations. We used gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

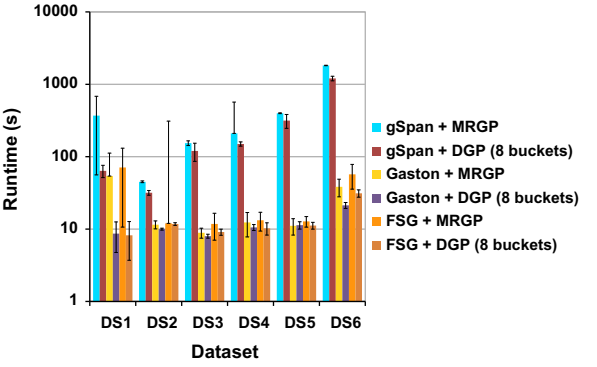

Fig. 7. Cost of partitioning methods. We used $\theta = 30\%$ and $\tau = 0.3$.

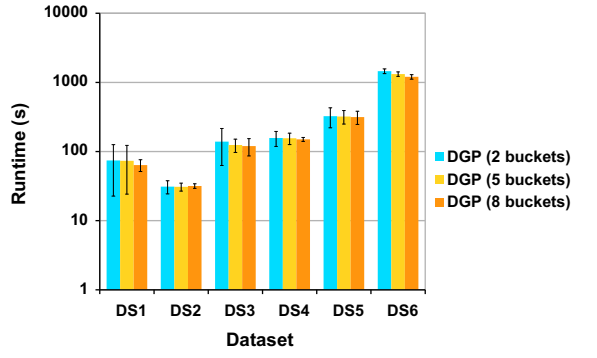

Fig. 8. Effect of the number of buckets on the cost of the density-based partitioning method. We used gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

frequent substructures mining algorithms that use the Apriori approach. Non-Apriori-based or pattern growth algorithms such as gSpan [4], MoFa [16], FFSM [17], Gaston [3] and ORIGAMI [18] have been developed to avoid the overheads of the Apriori-based algorithms. All these algorithms adopt the pattern growth methodology [19], which intends to extend patterns from a single pattern directly. The use of parallel and/or distributed algorithms for

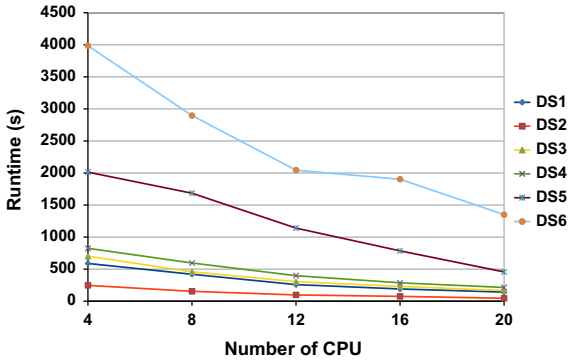

Fig. 9. Effect of the number of workers on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

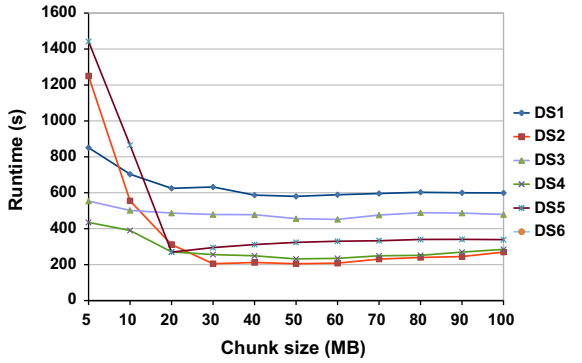

Fig. 10. Effect of chunk size on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

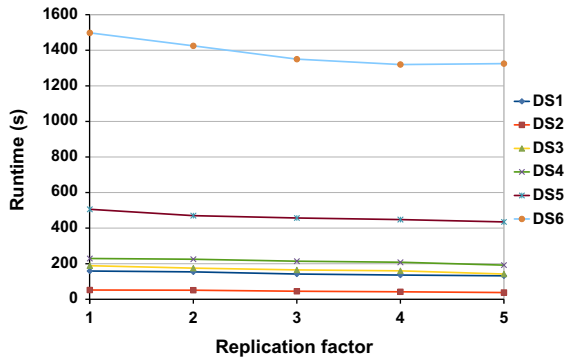

Fig. 11. Effect of replication factor on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

frequent subgraph mining comes from the impossibility to handle large graph and large graph databases on single machine. In this scope, several parallel and/or distributed solutions have been proposed to alleviate this problem [8,9,20–22]. In [20], the authors propose a MapReduce-based algorithm for frequent subgraph mining. The algorithm takes a large graph as input and finds all the subgraphs that match a given motif. The input large graph is represented as Personal Centre Network of every vertex in the graph [20]. 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.

In [9], the authors propose the MRPF algorithm for finding patterns from a complex and large network. 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 patterns reach the target size. The proposed algorithm is applied to prescription network in order to find some commonly used prescription network motifs that provide the possibility to discover the law of prescription compatibility.

In [21], the authors propose an approach to subgraph search over a graph database under the MapReduce framework. The main idea of the proposed 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.

The work presented in [23] presents an iterative MapReduce-based approach for frequent subgraph mining. The authors propose two heterogeneous MapReduce jobs per iteration: one for gathering subgraphs for the construction of the next generation of subgraphs, and the other for counting these structures to remove irrelevant data.

Another attention was carried to the discovery and the study of dense subgraphs from massive graphs. In [8], an algorithm for finding the densest subgraph in a massive graph is proposed. The algorithm is based on the streaming model of MapReduce. In the work presented in [22], the authors propose a statistical significance measure that compares the structural correlation of attribute sets against their expected values using null models. The authors define a structural correlation pattern as a dense subgraph induced by a particular attribute set.

Most of the above-cited solutions deal with large-scale subgraph mining in the case of one large graph as input. Only a few works include the subgraph mining process from large graph databases which is the addressed issue in this work.

6. Conclusion

In this paper, we addressed the issue of distributing the frequent subgraph mining process. We have described our proposed approach for large-scale subgraph mining from large-scale graph databases. 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 the proposed method is interesting in the case of large scale databases. The performance and scalability of our approach are satisfying for large-scale databases.

In the future work, we will study the use of other topological graph properties instead of the density in the

partitioning step. Also, we will study the relation between database characteristics and the choice of the partitioning technique.

A notable interest will be dedicated to the improvement of the runtime of our approach with task and node failures [24,25]. Furthermore, we plan to study the impact of the chunk size on the performance and the scalability of our approach.

Acknowledgements

This work was partially supported by the French Region of Auvergne thru the project LIMOS-AGEATIS-NUMTECH, by the French-Tunisian PHC project EXQUI, and the CNRS Mastodons project PETASKY. We would like to thank Abdoulaye Banire Diallo, Jan-Thierry Wegener and Hana Alouaoui for their useful comments, as well as the anonymous reviewers whose remarks help to significantly improve the paper.

References

[1] C. Faloutsos, K.S. Mccurley, A. Tomkins, Connection subgraphs in social networks, in: Workshop on Link Analysis, Counterterrorism, and Privacy, SIAM International Conference on Data Mining, 2004.
[2] M. Kuramochi, G. Karypis, Frequent Subgraph Discovery, in: Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 313–320.
[3] S. Nijssen, J.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, ACM, New York, NY, USA, 2004, pp. 647–652.
[4] X. Yan, J. Han, gSpan: graph-based substructure pattern mining, in: Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 721–724.
[5] M. Wörlein, T. Meinl, I. Fischer, M. 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, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 392–403.
[6] G. Malewicz, M.H. Austern, A.J. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, in: Proceedings of the 2010 International Conference on Management of data, SIGMOD '10, ACM, New York, NY, USA, 2010, pp. 135–146.
[7] S. Tatikonda, S. Parthasarathy, Mining Tree-Structured Data on Multicore Systems, vol. 2, VLDB Endowment, 2009, pp. 694–705.
[8] B. Bahmani, R. Kumar, S. Vassilvitskii, Densest Subgraph in Streaming and Mapreduce, vol. 5, VLDB Endowment, 2012, pp. 454–465.
[9] Y. Liu, X. Jiang, H. Chen, J. Ma, X. 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, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 341–355.
[10] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, vol. 51, ACM, New York, NY, USA, 2008, pp. 107–113.
[11] Y. Kwon, M. Balazinska, B. Howe, J.A. Rolia, Skewtune: mitigating skew in mapreduce applications, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012, pp. 25–36.
[12] J. Huan, W. Wang, J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, in: Proceedings of the Third IEEE International Conference on Data Mining, ICDM '03, IEEE Computer Society, Washington, DC, USA, 2003, p. 549.
[13] R. Agrawal, R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 487–499.
[14] S. Ghemawat, H. Gobioff, S.T. Leung, The Google file system, in: Proceedings of the 19th ACM Symposium on Operating Systems Principles, vol. 37 of SOSP '03, ACM, New York, NY, USA, 2003, pp. 29–43.
[15] A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, PKDD '00, Springer-Verlag, London, UK, 2000, pp. 13–23.
[16] C. Borgelt, M.R. Berthold, Mining molecular fragments: finding relevant substructures of molecules, in: Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 51–58.
[17] J. Huan, W. Wang, J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, in: Proceedings of the Third IEEE International Conference on Data Mining, ICDM '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 549–552.
[18] V. Chaoji, M. Al Hasan, S. Salem, J. Besson, M.J. Zaki, ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns, vol. 1, John Wiley & Sons, Inc., New York, NY, USA, 2008, pp. 67–84.
[19] J. Han, J. Pei, Y. Yin, Mining Frequent Patterns Without Candidate Generation, vol. 29, ACM, New York, NY, USA, 2000, pp. 1–12.
[20] B. Wu, Y. 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, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 107–115.
[21] Y. Luo, J. Guan, S. Zhou, Towards efficient subgraph search in cloud computing environments, in: Proceedings of the 16th International Conference on Database Systems for Advanced Applications, DASFAA'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 2–13.
[22] A. Silva, W. Meira, Jr., M.J. Zaki, Mining Attribute-Structure Correlated Patterns in Large Attributed Graphs, vol. 5, VLDB Endowment, 2012, pp. 466–477.
[23] S. Hill, B. Srichandan, R. Sunderraman, An iterative MapReduce approach to frequent subgraph mining in biological datasets, in: Proceedings of the 3rd ACM Conference on Bioinformatics, Computational Biology and Biomedicine, BCB '12, ACM, New York, NY, USA, 2012, pp. 661–666.
[24] C. Yang, C. Yen, C. Tan, S.R. Madden, Osprey: Implementing MapReduce-style fault tolerance in a shared-nothing distributed database, in: International Conference on Data Engineering '10, IEEE Computer Society, Los Alamitos, CA, USA, 2010, pp. 657–668.
[25] J.-A. Quiane-Ruiz, C. Pinkel, J. Schad, J. Dittrich, RAFTing MapReduce: fast recovery on the RAFT, in: Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, ICDE '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 589–600.