A novel MapReduce-based approach for distributed frequent subgraph mining

Sabeur Aridhi, Laurent d'Orazio, Monder Maddouri, Engelbert Mephu

A novel MapReduce-based approach for distributed frequent subgraph mining*

Sabeur Aridhi [1,2,3], Laurent d'Orazio [1,2], Mondher Maddouri [4] and Engelbert Mephu Nguifo [1,2]

[1] *CNRS, UMR 6158, LIMOS, F-63173 Aubiere, France.*

[2] *Clermont University, Blaise Pascal University, LIMOS, BP 10448, F-63000 Clermont-Ferrand, France.*

[3] *University of Tunis El Manar, LIPAH - FST, Academic Campus, Tunis 2092, Tunisia.*

[4] *Taibah University, Almadinah, Kingdom of Saudi Arabia.*

{aridhi,dorazio,mephu}@isima.fr - maddourimondher@yahoo.fr.

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. Dans ce papier, nous présentons 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 étude des performances de notre approche a été réalisée et a montré son efficacité.

Mots Clef

Fouille de sous-graphes, partitionnement de graphes, densité de graphe, MapReduce.

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 to approximate 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.

Keywords

Frequent subgraph mining, MapReduce, cloud computing, graph density, graph partitioning.

*An extended version of this paper will be published in the journal *Information Systems* (http ://dx.doi.org/10.1016/j.is.2013.08.005)

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. 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 [13], Gaston [16], ORIGAMI [8] and gSpan [20]. However, existing approaches are mainly used on centralized computing systems and evaluated on relatively small databases [18]. 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. In this context, several distributed solutions have been proposed [9, 15]. 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.

In this paper, we propose a scalable and distributed approach for large scale frequent subgraph mining based on MapReduce framework [6]. 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 our work, 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.

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 in Section 4.

2 Problem definition

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

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

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*

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

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*

$$LossRate(S_1, S_2) = \frac{\mid S_1 - S_2 \mid}{\mid S_1 \mid}, \qquad (2)$$

Definition 9 *Given a parameter $\epsilon \in [0, 1]$ and $SG(DB, \theta)$. An ϵ-approximation of $SG(DB, \theta)$ is a subset $S \subseteq SG(DB, \theta)$ such that*

$$LossRate(SG, S) \leq \epsilon. \qquad (3)$$

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 $SG(DB, \theta)$.
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 :

$$\mu = E[R]. \quad (4)$$

The cost measure of a partitioning technique is :

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

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

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.

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 Figure 1).

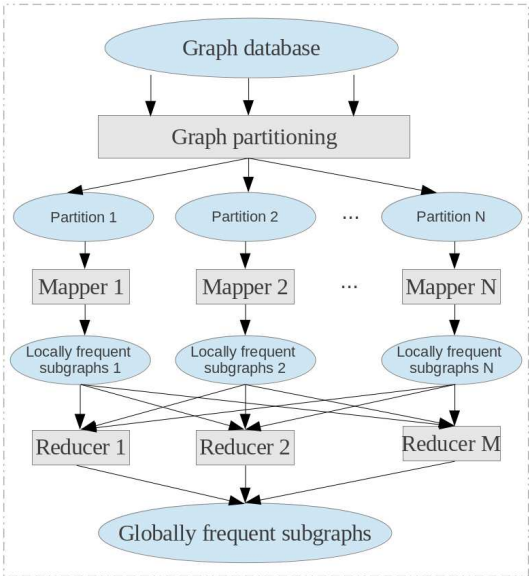


FIGURE 1 – A system overview of our approach.

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 arbitrarily 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. The Algorithms 1 and 2 present our Map and $Reduce$ functions :

Algorithm 1 Map function

Require: A partitioned graph database $DB = \{Part_1(DB), \ldots, Part_N(DB)\}$, 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: 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.

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 can not 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 ϵ ($\epsilon = 0$). 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.

3.2 The Density-based Graph Partitioning method

Considering the fact that the task of frequent subgraph mining depends on the density of graphs [18] [10], 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 2 gives an overview of the proposed partitioning method.

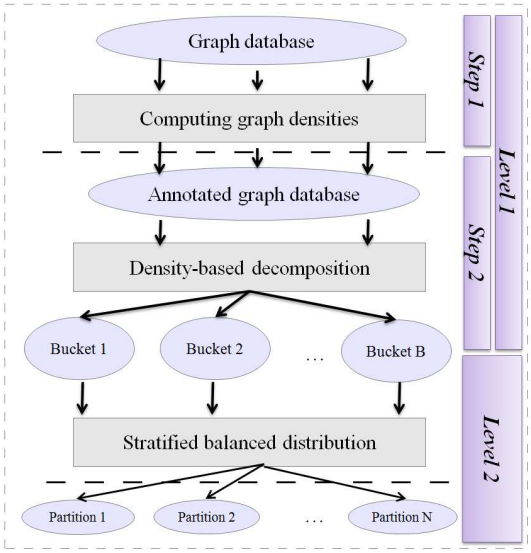


FIGURE 2 – The DGP method.

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 using its Map function. 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. 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 $B_i = \{P_{i1}, \cdots, P_{iN}\}$. Then, we construct the output partitions. Each one is constructed by appending one sub-partition from each bucket.

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

Datasets. The datasets used in our experimental study are described in Table 1.

TABLE 1 – 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 synthetic datasets are generated by the synthetic data generator *GraphGen* provided by Kuramochi and Karypis [13]. 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].

Implementation platform. All the experiments of our approach were carried out using a local hadoop 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

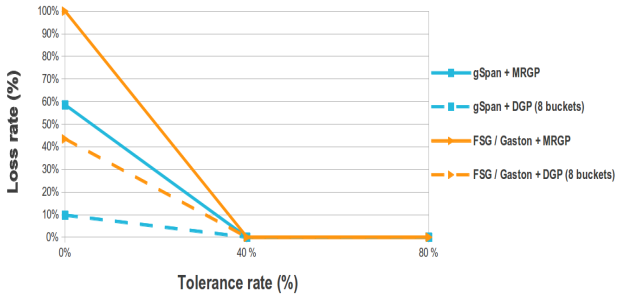
Accuracy and speedup. We mention that we could not conduct our experiment with the sequential algorithms in the case of $DS4$, $DS5$ and $DS6$ due to the lack of memory. However, with the distributed algorithm we were able to handle those datasets.

We illustrate in Figure 3 the effect of the proposed partitioning methods on the rate of lost subgraphs.

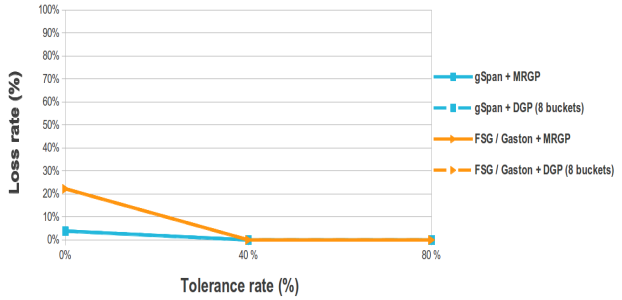
We can easily see in Figure 3 that the density-based graph partitioning allows low values of loss rate especially with low values of tolerance rate. We also notice that FSG and Gaston present a higher loss rate than gSpan in almost all cases.

We note also that the use of the proposed density-based partitioning method significantly improves the performance of

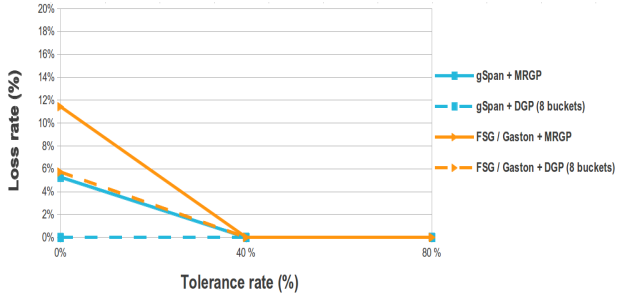
1. http ://dtp.nci.nih.gov/branches/npb/repository.html



(a) DS1



(b) DS2



(c) DS3

FIGURE 3 – Effect of the partitioning method on the rate of lost subgraphs.

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 term of density. Consequently, this balanced distribution of the data provides an effective load balancing scheme for distributed computations over worker nodes. Figure 4 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.

As illustrated in Figures 4, the density-based partitioning method allows a balanced distribution of workload across the distributed worker nodes.

In order to evaluate the capability of the density-based partitioning method to balance the computations over the used nodes, we show in Figure 5 the cost of this partitioning method in comparison with the MapReduce-based partitioning method. For each partitioning method and for each dataset, we present the mean value of the set of runtime values in the

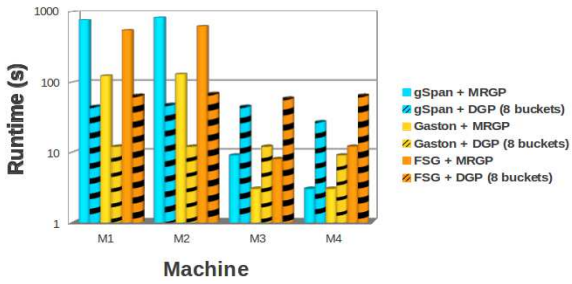


FIGURE 4 – Effect of the partitioning method on the distribution of computations. We used $\theta = 30\%$ and $\tau = 0.3$.

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.

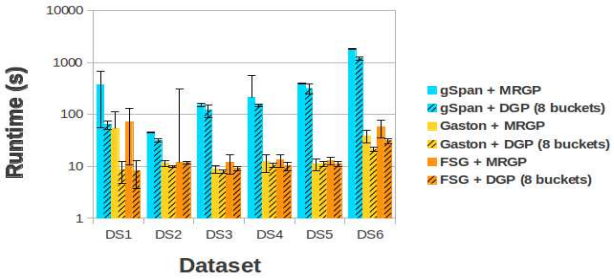


FIGURE 5 – Cost of partitioning methods. We used $\theta = 30\%$ and $\tau = 0.3$.

As shown in Figure 5, the density-based partitioning method allows minimal cost values in almost all datasets and all thresholds setting. This can be explained by the balanced distribution of graphs in the partitions. It is also clear that FSG and Gaston present a smaller runtime than gSpan (see Figure 5).

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 6 the runtime of our approach for each number of mapper machines.

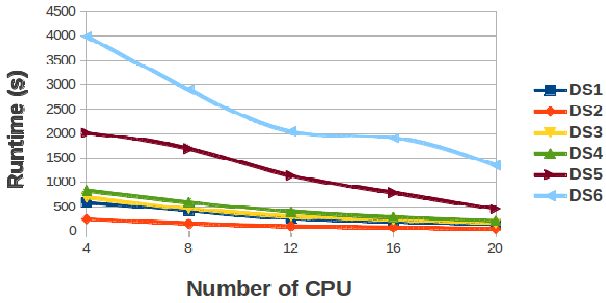


FIGURE 6 – Effect of the number of workers on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

As illustrated in Figure 6, 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. computationally to large datasets .

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.

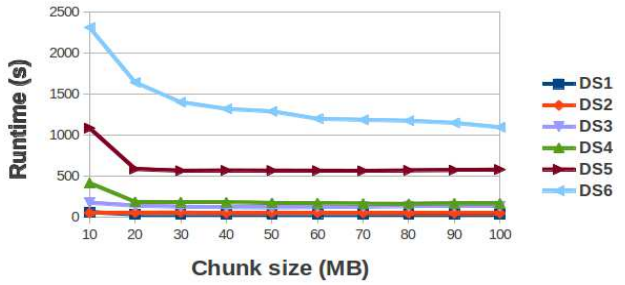


FIGURE 7 – Effect of chunk size on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

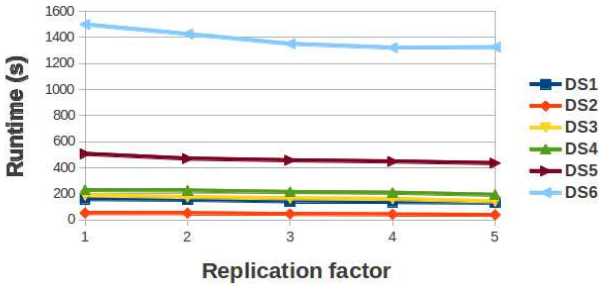


FIGURE 8 – Effect of replication factor on the runtime. We used DGP as a partitioning method, gSpan as a subgraph extractor, $\theta = 30\%$ and $\tau = 0.3$.

The experimentations presented in Figure 7 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 Figure 8, 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 [1]. AGM [12] and FSG [13] are two frequent substructures mining algorithms that use the Apriori approach. Non-Apriori-based or pattern growth algorithms such as gSpan [20], MoFa [4], FFSM [11], Gaston [16] and ORIGAMI [8] have been developed to avoid the overheads of the Apriori-based algorithms. All these algorithms adopt the pattern growth methodology [7], which intends to extend patterns from a single pattern directly.

The use of parallel and/or distributed algorithms for 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 [3] [14] [19] [15] [17] [5].

In [19], 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 [19]. 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 [14], 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 [15], 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 [9] 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 [3], 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 [17], 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 (see [2]). The performance and scalability of our approach are satisfying for large-scale databases.

In the extended version of this paper [2], we studied the effect of the number of buckets on the partitioning step. Also, 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 on the execution time of our approach.

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.

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 the anonymous reviewers for their useful comments.

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB '94*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[2] S. Aridhi, L. d'Orazio, M. Maddouri, and E. Mephu Nguifo. Density-based data partitioning strategy to approximate large-scale subgraph mining. *Information Systems, Elsevier, in press*.

[3] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. volume 5, pages 454–465. VLDB Endowment, Jan. 2012.

[4] C. Borgelt and M. R. Berthold. Mining Molecular Fragments : Finding Relevant Substructures of Molecules. In *IEEE ICDM '02*, pages 51–58, 2002.

[5] G. Buehrer, S. Parthasarathy, and Y.-K. Chen. Adaptive parallel graph mining for cmp architectures. In *ICDM*, pages 97–106. IEEE Computer Society, 2006.

[6] J. Dean and S. Ghemawat. Mapreduce : Simplified data processing on large clusters. *Commun. ACM*, 51(1) :107–113, Jan. 2008.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. volume 29, pages 1–12, New York, NY, USA, May 2000. ACM.

[8] M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki. Origami : Mining representative orthogonal graph patterns. In *IEEE ICDM'07*, pages 153–162. IEEE Computer Society, 2007.

[9] S. Hill, B. Srichandan, and R. Sunderraman. An iterative mapreduce approach to frequent subgraph mining in biological datasets. In *ACM BCB'12*, pages 661–666. ACM, 2012.

[10] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *IEEE ICDM '03*, pages 549–, 2003.

[11] J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. In *IEEE ICDM '03*, pages 549–552. IEEE Computer Society, 2003.

[12] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *ECML-PKDD'00*, pages 13–23. Springer-Verlag, 2000.

[13] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *IEEE ICDM '01*, pages 313–320, Washington, DC, USA, 2001.

[14] Y. Liu, X. Jiang, H. Chen, J. Ma, and X. Zhang. Mapreduce-based pattern finding algorithm applied in motif detection for prescription compatibility network. In *APPT '09*, pages 341–355. Springer-Verlag, 2009.

[15] Y. Luo, J. Guan, and S. Zhou. Towards efficient subgraph search in cloud computing environments. In *DASFAA'11*, pages 2–13, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *ACM KDD'04*, pages 647–652, New York, NY, USA, 2004. ACM.

[17] A. Silva, W. Meira, Jr., and M. J. Zaki. Mining attribute-structure correlated patterns in large attributed graphs. volume 5, pages 466–477. VLDB Endowment, Jan. 2012.

[18] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In *ECML-PKDD'05*, pages 392–403, Berlin, Heidelberg, 2005. Springer-Verlag.

[19] B. Wu and Y. Bai. An efficient distributed subgraph mining algorithm in extreme large graphs. In *AICI'10*, pages 107–115. Springer-Verlag, 2010.

[20] X. Yan and J. Han. gSpan : Graph-Based Substructure Pattern Mining. In *IEEE ICDM'02*, pages 721–724, 2002.