LC-mine: a framework for frequent subgraph mining with local consistency techniques

Brahim Douar, Michel Liquière, Chiraz Latiri, Yahya Slimani

Knowl Inf Syst
DOI 10.1007/s10115-014-0769-4

REGULAR PAPER

LC-mine: a framework for frequent subgraph mining with local consistency techniques

Brahim Douar · Michel Liquiere · Chiraz Latiri · Yahya Slimani

Received: 4 February 2012 / Revised: 14 March 2014 / Accepted: 3 July 2014

Abstract Developing algorithms that discover all frequently occurring subgraphs in a large graph database is computationally extensive, as graph and subgraph isomorphisms play a key role throughout the computations. Since subgraph isomorphism testing is a hard problem, fragment miners are exponential in runtime. To alleviate the complexity issue, we propose to introduce a bias in the projection operator and instead of using the costly subgraph isomorphism projection, one can use a polynomial projection having a semantically valid structural interpretation. In this paper, our purpose is to present LC- MINE, a generic and efficient framework to mine frequent subgraphs by the means of local consistency techniques used in the constraint programming field. Two instances of the framework based on the arc consistency technique are developed and presented in this paper. The first instance follows a breadth-first order, while the second is a pattern-growth approach that follows a depth-first search space exploration strategy. Then, we prove experimentally that we can achieve an important performance gain without or with nonsignificant loss of discovered patterns in terms of quality.

Keywords Relational learning · Graph mining · Projection operator · Graph classification

1 Introduction and motivations

Graphs become increasingly important in modeling complicated structures, such as chemical compounds, protein structures or even social networks. The aim of the graph mining task

B. Douar · M. Liquiere
LIRMM, Montpellier II University, 161 rue Ada, 34392 Montpellier, France

B. Douar (✉) · C. Latiri
LIPAH, Faculty of Sciences of Tunis, Tunis El Manar University, 1060 Tunis, Tunisia
e-mail: b.douar@gmail.com

Y. Slimani
LISI, INSAT, University of Carthage, 1080 Tunis, Tunisia

Published online: 24 July 2014

is to find interesting graph patterns in a big graph or a collection of graphs. Among the various kinds of graph patterns, frequent substructures are very useful for characterizing graph sets, differentiating between different groups of graphs and cluster analysis. In fact, frequent subgraph mining is an important challenge [3], especially in its most important applications areas like chemical informatics, bioinformatics or Web analysis to cite but a few. However, discovering frequent subgraphs is a thriving challenge due to their exponential number. Indeed, based on the APRIORI principle [1], a frequent n-edge graph may contain 2^n frequent subgraphs. In addition to this exponential search space, frequent subgraph miners face the NP-completeness of the subgraph isomorphism projection which is a kernel operator in subgraphs counting and matching process [13,16,24,25].

Many frequent subgraph miners have tried to avoid the NP-completeness of subgraph isomorphism problem by storing all embeddings in embedding lists which consist of a mapping of the vertices and edges of a fragment to the corresponding vertices and edges in the graph it occurs in. It is clear that with this trick, we can avoid excessive subgraph isomorphism tests when counting fragments support and, therefore, avoid exponential runtime.

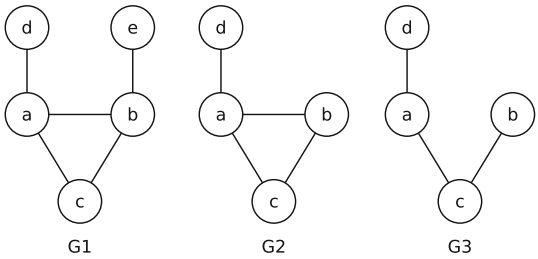
However, these approaches face exponential memory consumption instead. So, we can say that they are only trading time versus storage. This strategy can even cause problem if not enough memory is available or if the memory throughput is not high enough. The authors in [24], after an extensive experimental study of different subgraph miners, conclude that embedding lists do not considerably speed up the search for frequent fragments. Thus, even though GSPAN [25] does not use them, it is almost as competitive as GASTON [16] and FFSM [10], at least with not too big fragments.

To alleviate the complexity issue, we propose to introduce a bias in the projection operator and instead of using the subgraph isomorphism projection, one can use a polynomial projection having a semantically valid structural interpretation. In this paper, our purpose is to present LC- MINE, a generic and efficient framework to mine frequent subgraphs by the means of local consistency techniques used in the constraint programming field [20]. Two instances of the framework are developed and presented in the following. The first instance follows a breadth-first order to find frequent subgraphs and takes advantage of the well-known APRIORI [1] levelwise strategy. The second is a pattern-growth approach that follows a depth-first search space exploration strategy and uses powerful pruning techniques in order to considerably reduce this search space.

This paper is a wide extension of two previous works we carried out [4,5]. The present work defines a generic framework that generalizes the two instances presented in [4] and [5]. We have specified this framework after a thorough analysis of the characteristics of the two approaches. Furthermore, the earlier works have been extended in two ways: First, we have brought theoretical proofs related to the pruning techniques initially introduced in [5]. Then, we have performed an extensive comparative study of the two proposed approaches with much larger graph databases. This aims at validating these approaches and assessing them with regard to the state-of-the-art subgraph mining algorithms.

The remainder of the paper is organized as follows: Section 2 recalls the basic mathematical foundations for frequent subgraph mining. In Sect. 3, we propose, in a generic way, the LC- MINE framework. Then, in Sects. 4 and 5, we present two efficient instances of our framework based on the AC-projection definitions. Finally, we experimentally evaluate the computational efficiency of our proposed instances and study the relevance of the AC-reduced patterns for the supervised graph classification in Sect. 6. The conclusion is then presented in Sect. 7.

Fig. 1 A set of labeled graphs



2 Frequent subgraph mining

Given a database consisting of small graphs, for example, molecular graphs, the problem of mining frequent subgraphs is to find all subgraphs that are subgraph isomorphic with a large number of example graphs in the database. In this section, we recall some preliminary concepts as well as a brief review of literature dedicated to frequent subgraph mining.

2.1 Basic definitions

For the sake of clarity, the rest of this paper will deal with labeled undirected graphs only, although the concepts and methods can be extended in a straightforward way to directed labeled graphs.

Definition 2.1 (*Labeled Graph*) A labeled graph can be represented by a 4-tuple, $G = (V, E, L, l)$, where:

- V is a set of vertices,
- $E \subseteq V \times V$ is a set of edges,
- L is a set of labels,
- $l : V \cup E \rightarrow L$, l is a function assigning labels to the vertices and the edges.

This definition can be generalized to include unlabeled graphs if the label set L is an empty set.

Definition 2.2 (*Induced subgraph*) A subgraph S of a graph G is said to be induced if, for any pair of vertices x and y of S, (x, y) is an edge of S if and only if (x, y) is an edge of G. In other words, S is an induced subgraph of G if it has all the edges that appear in G over the same vertex set.

For example, we can see in Fig. 1 that G_2 is an induced subgraph of G_1 while G_3 is not an induced one.

Definition 2.3 (*Isomorphism, Subgraph Isomorphism*) Given two graphs $G_1(V_1, E_1, L_1, l_1)$ and $G_2(V_2, E_2, L_2, l_2)$, an isomorphism is a bijective function $f : V_1 \rightarrow V_2$, such that $\forall x \in V_1, l_1(x) = l_2(f(x))$, and $\forall (x, y) \in E_1, (f(x), f(y)) \in E_2$ and $l_1(x, y) = l_2(f(x), f(y))$.

A subgraph isomorphism from G_1 to G_2 is an isomorphism from G_1 to a subgraph of G_2.

Definition 2.4 (*Graph projection*) Let G_1 and G_2 be two graphs. A graph projection is the mapping which maps each vertex of G_1 into one or many vertices of G_2. This defines the generalization order between graphs.

This latter projection can have different names according to the field in which it is defined. In Table 1, we expose terminology equivalents in these fields.

Table 1 Two types of graph projection and their equivalents

Graph theory	Category theory	Universal algebra	Logic
Graph homomorphism	Morphism	Homomorphism	θ-subsumption
Subgraph isomorphism	Monomorphism	Injective homomorphism	OI-subsumption

Definition 2.5 (*Frequent Subgraph Mining*) Given a graph dataset, $GS = \{G_i | \, i = 0 \ldots n\}$, and a minimal support (minSup), let

$$\varsigma(g, G) = \begin{cases} 1 & \text{if there is a projection from } g \text{ to } G \\ 0 & \text{otherwise.} \end{cases}$$

$$\sigma\,(g, GS) = \sum_{G_i \in GS} \varsigma(g, G_i)$$

$\sigma\,(g, GS)$ denotes how frequently g occurs in GS, i.e., the support of g in GS. Frequent subgraph mining aims at finding every g graph such that $\sigma\,(g, GS)$ is greater than or equal to minSup.

Known frequent subgraph miners are based on this definition and deal with the special case where the projection operator is a subgraph isomorphism.

2.2 Related work

Algorithms for frequent subgraph mining are based on two pattern discovery paradigms, namely *breadth-first search* and *depth-first search*. Most of these algorithms employ different ways for candidate generation and support counting. An interesting quantitative comparison of the most cited subgraph miners is given in [24].

2.2.1 Breadth-first approaches

The algorithm that follows a breadth-first order to find frequent subgraphs takes advantage of the well-known AprioriI [1] levelwise strategy. In the literature, pioneering and most cited approaches are AGM [11] and FSG [12].

The particularity of AGM is that it only mines frequent- induced subgraphs in the graph database, as well as dealing with graphs having self-loops. During the mining process, the algorithm extends subgraphs by adding one vertex at each level. Experiments, reported in [11], show that AGM achieves relatively good performances for synthetic dense datasets. Compared to more recent approaches, and except the fact of self-loops handling, AGM is an obsolete mining approach with a very poor scalability power [11].

So, a more efficient breadth-first approach named FSG [12] has been introduced. The latter algorithm is also based on the same level-by-level expansion exactly as AprioriI did but in the context of labeled graphs. FSG essentially differs from the AGM approach by adding one edge at a time (instead of one vertex for AGM) allowing to efficiently generate candidates.

Various optimizations, based partially on labels, vertex degrees and hierarchical structure of the search space, have been proposed for canonical form computation, candidate generation and counting. These optimizations have allowed, the FSG algorithm, to scale to large graphs. The performance of FSG was relatively worse for graph database with few vertex and edge

labels as its dependency on labels and vertex degrees was strong. This is due to the exponential complexity of the subgraph isomorphism test that badly influenced the overall performance.

Besides, if we have enough vertex and edge labels, then FSG will be able to achieve good performance and to scale linearly with the database size.

In this paper, we are particularly interested in this latter approach. In fact, we propose a frequent subgraph mining approach based on the FSG version and using a novel operator for the support counting process as well as an innovative graph reduction algorithm.

2.2.2 Depth-first approaches

Main depth-first approaches are restricted to finding connected subgraphs and traverse the search lattice in a vertical way. In this respect, several algorithms were proposed in the literature. We give, in the following, a brief description of the two most cited ones, namely the GSPAN [25] and GASTON [16] algorithms.

The GSPAN algorithm [25] is based on a canonical representation for graphs, called *dfs-code*. A dfs-traversal of a graph defines an ordering of the visited edges. The concatenation of edge representation in that order is the graph's dfs-code. In order to prevent isomorphic subgraphs (duplicates) generation, GSPAN computes the canonical (lexicographically smallest) dfs-code from each refinement using a series of permutations. Refinements with non-minimal dfs-code can be pruned. The GSPAN stores occurrence lists for each subgraph. Explicit subgraph isomorphism testing has to be done on all graphs in these occurrence lists.

Instead of storing occurrence lists for each subgraph, the GASTON algorithm [16] stores all embeddings[1] to generate only refinements that actually appear and to achieve faster subgraph isomorphism testing. The main idea behind this algorithm is that there are efficient ways to enumerate paths and trees. By looking for subgraphs that are paths or trees first, and by only dealing with general graphs with cycles at the very end, a large fraction of the work can be done efficiently. Indeed, GASTON will face the NP-completeness of the subgraph isomorphism problem only in the last phase. Duplicate detection is performed in two phases: hashing to pre-sort and an explicit graph isomorphism test for final duplicate detection.

In the following section, we introduce a novel framework named LC- MINE. The core idea behind it is the use of a biased projection operator based on local consistency techniques in order to avoid the NP-completeness of the subgraph isomorphism and to considerably reduce the search space.

3 The LC-MINE framework

This framework introduces the use of local consistency techniques in the graph mining process. Such a framework alleviates the complexity of using the costly subgraph isomorphism projection usually used by frequent subgraph miners.

The idea of using constraint programming in a graph matching process is not new [21,26]. What is novel in this framework is that we are only dealing with polynomial local consistency techniques. There are many different levels of consistency that can be achieved, and these can be used to ensure the projection with a certain desirable precision. Each instance of the framework can use its own local consistency technique. In order to do so, some operators

[1] An embedding is a mapping of the nodes and edges of a subgraph to the corresponding nodes and edges in the graph the subgraph occurs in.

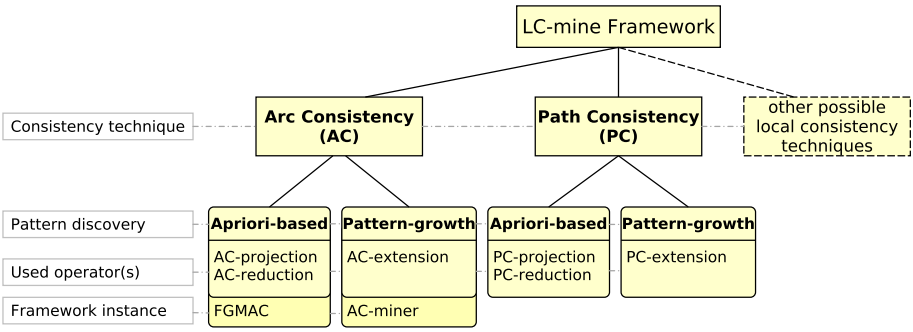


Fig. 2 The LC- MINE framework. The left-most leaves of the tree show the implemented instances of the framework, namely the FGMAC and AC- MINER approaches. The upper level expose the used operators within each approach. Then, the level "Pattern discovery" enumerates the possible search space exploration paradigms. Finally, the top-most level exhibits the multiple possibilities of local consistency techniques that we can use within the framework

must be defined as well as a semantic interpretation of frequent graphs derived by using the chosen local consistency level. An overview of the framework is depicted in Fig. 2.

In the following, we will firstly introduce preliminary concepts concerning the constraint programming field, followed by a generic presentation of the key operators related to the LC- MINE framework.

3.1 Preliminary concepts

A constraint satisfaction problem (CSP) [20] involves the assignment of values to variables subject to a set of constraints. A large variety of problems in Artificial Intelligence and other areas of computer science can be viewed as a special case of the constraint satisfaction problem. A great deal of research in constraint satisfaction has focused on algorithms which, given a constraint network as input, automatically find a solution. This is useful in applications where, once the problem has been formulated as a constraint network, no user interaction is required. Any constraint satisfaction problem involves variables. Each variable can be given a value chosen from a set of possible values called its domain. The constraints impose limitations on the values which may be assigned to a variable or a combination of variables. Together, variables, domains and constraints form a constraint network.

Definition 3.1 A constraint network is a triplet $\mathcal{N} = (X, D, C)$ where:

- $X = \{x_1, ..., x_n\}$ is a finite set of variables;
- D is a function that maps each variable x in X to a finite set of values, written $D(x)$, which it is allowed to take. The set $D(x)$, called the domain of x, is also denoted Dx;
- $C = \{C_1, ..., C_k\}$ is a finite set of constraints. Each constraint restricts the combination of values that a set of variables may take simultaneously. A constraint involving only two variables is called binary.

Within the LC- MINE framework, we are only dealing with normalized binary constraint networks. Such a network must have only binary constraints and must not have two constraints involving exactly the same variables.

3.2 The framework operators

In this section, we will present key operators related to the LC- MINE framework. These operators are presented in a generic fashion and must be precisely defined in each one of the instances within our framework. In the remainder of this paper, we will give a generic name for each operator. These names will have the string "LC" as prefix, namely LC-projection, LC-reduction and LC-extension. The prefix references the local consistency technique used, and the two letters of "LC" will change depending on this technique (i.e., AC for arc consistency, PC for path consistency and so on).

3.2.1 The projection operator (LC-projection)

This is the kernel operator of our framework. This is intended to have polynomial space/time complexity. Indeed, this operator is supposed to replace the costly subgraph isomorphism which suffers from an exponential complexity. It is worth mentioning that, if there is no LC-projection between two graphs G_1 and G_2, then there will be neither graph homomorphism; nor subgraph isomorphism between them.

Given two graphs G_1 and G_2, the purpose of the LC-projection operator is to derive a mapping $\mathcal{I}$ which associates one or more vertices from G_2 to each vertex of G_1. The formal labeling definition is given by the definition below.

Definition 3.2 (*Labeling*) Let $G_1(V_1, E_1, L_1, l_1)$ and $G_2(V_2, E_2, L_2, l_2)$ be two graphs and 2^{V_2} be the set of all subsets of V_2. We call a labeling from G_1 into G_2 a mapping $\mathcal{I} : V_1 \rightarrow 2^{V_2} | \forall x \in V_1, \forall y \in \mathcal{I}(x), l_1(x) = l_2(y)$.

Thus, for a vertex $x \in V_1, \mathcal{I}(x)$ is a set of vertices of G_2 with the same label $l_1(x)$. We can say that $\mathcal{I}(x)$ is the set of "possible images" of the vertex x in G_2.

The core idea behind using local consistency techniques to replace the subgraph isomorphism is the conversion of the projection problem to a CSP materialized by a constraint network.

The problem of the LC-projection of a graph $G_1(V_1, E_1, L_1, l_1)$ into a graph $G_2(V_2, E_2, L_2, l_2)$ is equivalent to the following constraint satisfaction problem. A variable x_i is associated with each vertex $v_i \in V_1$, and each variable having $l_1(x_i)$ as label takes values on domain equal to the set: $\{\bigcup\{v\}, v \in V_2, l_2(v) = l_1(x_i)\}$.

This problem can be modelized with a constraint network composed of:

- a set of variables $X = \{X_1, ..., X_{|V_1|}\}$;
- a domain for each variable X_i, $D(X_i) = \{\bigcup\{v\}, v \in V_2, l_2(v) = l_1(X_i)\}$;
- a set of binary constraints $C = \{C_1, ..., C_{|E_1|}\}$;

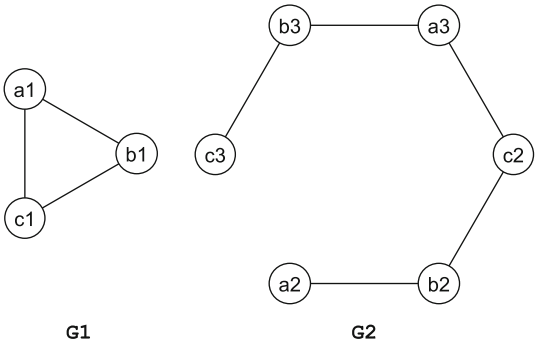
Each constraint involves two variables, $X(C_{mn}) = (X_m, X_n)$ and for each values tuple (v_m, v_n), $v_m \in D(X_m)$ and $v_n \in D(X_n)$, C_{mn} is satisfied if and only if there is an edge $(v_m, v_n) \in E_2$.

Since each vertex in G_1 is associated with all vertices having the same label in G_2, the first labeling is trivial but it will be refined by the constraint propagation mechanism.

Example 3.1 Let us suppose that we are using an arc consistency-based LC-projection operator (*cf.* AC-projection). Considering the two graphs depicted in Fig. 3, we are trying to find an AC-projection from graph G_1 into G_2.

The corresponding constraint network $\mathcal{N}$ associated with this projection problem is depicted in Fig. 4(1). We can see that $\mathcal{N}$ is not arc consistent because there are some values

Fig. 3 Two graphs G_1 and G_2



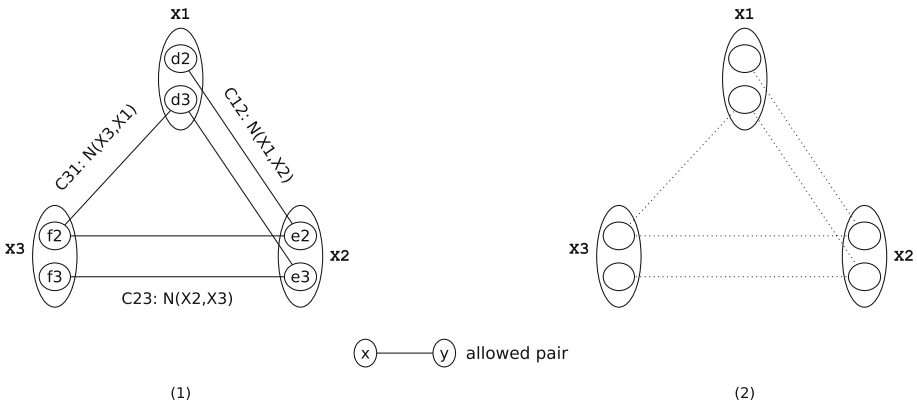


Fig. 4 Constraint network $\mathcal{N}$ of Example 3.1, before arc consistency (**1**) and after (**2**)

(vertices) inconsistent with some neighborhood constraints. Checking constraints C_{12} and C_{23} does not permit to remove any value. But when checking constraint C_{31}, we see that d_2 must be removed from $D(X_1)$ because it has no neighbor vertex in $D(X_3)$. Likewise, f_3 also has to be removed from $D(X_3)$ because it has no neighbor vertex in $D(X_1)$. Removing d_2 from $D(X_1)$ and f_3 from $D(X_3)$ causes in turn, respectively, the removal of e_2 from $D(X_2)$ (because of the C_{12} constraint) and e_3 from $D(X_2)$ (because of the C_{23} constraint). At this level, $D(X_2)$ becomes empty and we can conclude that the constraint network $\mathcal{N}$ is not arc consistent. Therefore, there is no AC-projection from G_1 into G_2. In fact, if we continue the constraint propagation process, we will finally have a constraint network with no values [*cf.* Fig. 4(2)].

However, if we add the edge (d_2, f_3) to E_1, the initial constraint network $\mathcal{N}$ will be arc consistent, and all the values (vertices) will be compatible with all constraints.

3.2.2 The reduction operator (LC-reduction)

Before introducing the reduction operator, it is necessary to present an important property of the projection operator.

Definition 3.3 (*Equivalence*) Let G_1 and G_2 be two graphs, if we have an LC-projection from G_1 into G_2, and an LC-projection from G_2 into G_1, then G_1 and G_2 will be considered as equivalent (w.r.t. to the LC-projection).

This equivalence relation will bring equivalence classes of graphs. For each equivalence class, we can search for a specific graph which will be its unique representative. Given a graph G, the reduction operator is intended to derive a reduced graph G' which must be the minimal graph equivalent to G.

We will give the generic name of "LC-reduced graph" to this minimal-sized graph. It is worth noting that the definition of the graph size depends on the algorithm used and the chosen local consistency technique.

3.2.3 The extension operator (LC-extension)

This operator is the core of the pattern-growth approaches within the LC- MINE framework. Given a graph G, the extension operator is intended to extend G by an extra vertex, edge or path. This extension must be done in such a way that the constraint network $\mathcal{N}'$ associated with the extended graph G' remains consistent w.r.t. the local consistency technique which is used.

3.3 Search strategies

All the operators defined within the LC- MINE framework allow us to use APRIORI-like strategy based on a breadth-first search space exploration. These operators also favor the use of the pattern-growth strategy which usually explores the latter in a depth-first manner. In the following, we will present details about these search strategies.

3.3.1 APRIORI*-like strategy*

Within the LC- MINE framework, approaches adopting this search space exploration paradigm take advantage of the APRIORI [1] levelwise strategy. Indeed, based on this strategy, LC- MINE instances discover all frequent subgraphs in ascending order of the size of the graphs based on the anti-monotonic property of the support threshold. The search for frequent graphs starts with small graphs and proceeds in a bottom-up manner by generating candidates having an extra vertex, edge or path. These approaches explore the full isomorphism-wide search space and extract only LC-reduced frequent subgraphs. In other words, at each level during the breadth-first exploration, all the candidates have to be generated on a classical (isomorphic) way. However, during the support calculation phase, the polynomial LC-projection operator will be used instead of the costly subgraph isomorphism test to verify whether a candidate subgraph appears on a graph transaction or not. Then, if a given subgraph candidate has a sufficient support, its reduced version will be added to the output of the algorithm. As previously said, the reduction will be done using the LC-reduction operator already defined within the LC- MINE framework.

3.3.2 Pattern-growth strategy

In order to allow our framework to scale to bigger graph databases, we propose a depth-first exploration scheme. This search strategy is made possible thanks to the extension operator previously defined. In fact, a generic approach for frequent subgraphs discovery within our framework will start by an empty graph and will try to extend it at each step w.r.t. the support threshold. An extension can be made if the resulting constraint network associated with the extended subgraph remains consistent w.r.t. the local consistency technique used. Thus, when it becomes impossible to further extend the current subgraph, it will be added to the output

of the approach and will be considered as frequent. The challenge is to efficiently find the unique set of extensions, rather than stepping past the same subgraph multiple times.

In the remainder of this paper, we propose two instances of the LC- MINE framework. The first instance, named FGMAC [4], follows a breadth-first order to find frequent subgraphs and uses an APRIORI-like [1] search strategy. The second, named AC-miner [5], is a pattern-growth approach that follows a depth-first search space exploration strategy and uses powerful pruning techniques in order to considerably reduce this search space. These two instances are based on the arc consistency technique. Indeed, these approaches use properties of the AC-projection operator initially introduced in [14].

4 FGMAC: frequent subgraph mining with arc consistency

In this section, we present FGMAC, which is the first instance of our LC- MINE framework. This is an APRIORI-like frequent subgraph mining approach based on the AC-projection operator. Our approach is closely similar to the FSG algorithm [13]. In fact, the essential innovation related to FGMAC is the support counting part. Instead of subgraph isomorphism, the AC-projection is used to verify whether a candidate graph appears in a transaction or not. In the following, we introduce the two operators used within the FGMAC approach, namely AC-projection and AC-reduction. As explained previously, these operators are crucial for every APRIORI-based instances of our framework.

4.1 The AC-projection operator

The approach suggested in [14] advocates a projection operator based on the arc consistency algorithm. This projection method has the required properties: polynomiality, local validation, parallelization, structural interpretation. We note that the name "AC-projection" comes from the classical AC (arc consistency) used in [2].

4.1.1 Mathematical foundations

In this section, we will present a mathematical formulation of the AC-projection operator.

Definition 4.1 (*AC-compatible* $\curvearrowright$) Let $G(V, E, L, l)$ be a graph $V_1 \subseteq V$, $V_2 \subseteq V$
V_1 and V_2 are AC-compatible if and only if

1. $\forall x_k \in V_1 \exists y_p \in V_2 | (x_k, y_p) \in E$
2. $\forall y_q \in V_2 \exists x_m \in V_1 | (x_m, y_q) \in E$.

We denote $V_1 \curvearrowright V_2$ two AC-compatible vertex sets.

Definition 4.2 (*Consistency for one edge*) Let $G_1(V_1, E_1, L_1, l_1)$ and $G_2(V_2, E_2, L_2, l_2)$ be two graphs. We say that a labeling $\mathcal{I} : V_1 \rightarrow 2^{V_2}$ is consistent with an edge $(x, y) \in E_1$, if and only if $\mathcal{I}(x) \curvearrowright \mathcal{I}(y)$.

Definition 4.3 (*AC-labeling*) Let $G_1(V_1, E_1, L_1, l_1)$ and $G_2(V_2, E_2, L_2, l_2)$ be two graphs. A labeling $\mathcal{I} : V_1 \rightarrow 2^{V_2}$ is an AC-labeling if and only if $\mathcal{I}$ is consistent with all the edges $e \in E_1$.

Definition 4.4 (*AC-projection* $\rightharpoonup$) Let $G_1(V_1, E_1, L_1, l_1)$ and $G_2(V_2, E_2, L_2, l_2)$ be two graphs. An AC-labeling $\mathcal{I} : V_1 \rightarrow 2^{V_2}$ is an AC-projection if and only if $\mathcal{I}$ is *maximal*. $\mathcal{I}$ is said to be maximal if and only if $\forall$ AC-labeling $\mathcal{I}' : V_1 \rightarrow 2^{V_2}$ and $\forall x \in V_1$, $\mathcal{I}'(x) \subseteq \mathcal{I}(x)$. We denote it $G_1 \rightharpoonup G_2$.

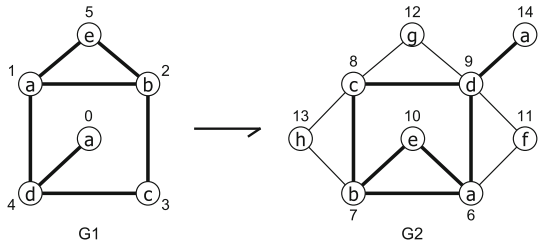


Fig. 5 An AC-projection example ($G_1 \rightharpoonup G_2$)

Example 4.1 Let G_1 and G_2 be two graphs. In Fig. 5, we consider the labeling $\mathcal{I}$: $\mathcal{I}(a0) = \{a6, a14\}$, $\mathcal{I}(a1) = \{a6\}$, $\mathcal{I}(b2) = \{b7\}$, $\mathcal{I}(c3) = \{c8\}$, $\mathcal{I}(d4) = \{d9\}$, $\mathcal{I}(e5) = \{e10\}$. We verify that $\mathcal{I}(a0) \curvearrowright \mathcal{I}(d4)$, $\mathcal{I}(d4) \curvearrowright \mathcal{I}(c3)$, $\mathcal{I}(c3) \curvearrowright \mathcal{I}(b2)$, $\mathcal{I}(b2) \curvearrowright \mathcal{I}(e5)$, $\mathcal{I}(e5) \curvearrowright \mathcal{I}(a1)$, $\mathcal{I}(a1) \curvearrowright \mathcal{I}(d4)$, $\mathcal{I}(b2) \curvearrowright \mathcal{I}(a1)$. Then $\mathcal{I}$ is an AC-projection from G_1 into G_2, since $\mathcal{I}$ is a maximal labeling which is consistent with all edges of G_1.

4.1.2 The AC-projection algorithm outline

In [4], we have introduced an improved AC-projection algorithm for graphs (based on the AC3 algorithm [15]). The AC-projection algorithm takes two graphs G_1 and G_2 and tests whether there is an AC-projection from G_1 into G_2. Like the AC3 algorithm, the actual AC-projection algorithm has a worst-case time complexity of $O(e \times d^3)$ and space complexity of $O(e)$ where e is the number of arcs and d is the size of the largest domain. In our case, the size of the largest domain is the size of the largest subset of nodes with the same label.

4.2 The AC-reduction operator

The AC-reduction is a key operator used by the FGMAC algorithm. Given a graph G, the AC-reduction operator is intended to derive a reduced graph G' which must be the minimal graph equivalent to G.

4.2.1 The AC-equivalence relation

The following definition introduces an equivalence relation between graphs w.r.t. AC-projection.

Definition 4.5 (*AC-equivalent graphs*) Two graphs G_1 and G_2 are AC-equivalent if and only if both $G_1 \rightharpoonup G_2$ and $G_2 \rightharpoonup G_1$ are fulfilled.

Two AC-equivalent graphs G_1 and G_2 are denoted by $G_1 \leftrightharpoons G_2$.

We have an equivalence relation between graphs using the AC-projection. For each equivalence class there is a minimal-sized graph for which we give the name of "AC-reduced graph."

4.2.2 The AC-reduction algorithm outline

The AC-reduction algorithm [4] is able to construct the AC-reduced graph considering any graph G. To do this, the algorithm does an auto AC-projection $G \rightharpoonup G$ and then make the necessary merges. Thus, this algorithm is very simple and has a polynomial complexity, since the AC-projection's complexity is polynomial. We note that, in [4], we have proved that the AC-reduced graph is minimal.

4.3 Related work and discussion

New graph matching approaches have recently been proposed [6–8]. These approaches redefine the graph isomorphism problem by relaxing the hard constraints imposed by this type of exact matching in order to allow dealing with real-world problems. In [8], the authors define a new type of homomorphism [9] denoted *p-hom*. Briefly, this homomorphism allows the matching of one edge from the pattern graph to several edges in the target graph. It introduces a new concept of similarity between vertices. In [7], Fan et al. introduce new classes of graphs as well as several new algorithms for computing the isomorphism between graphs of these new classes with cubic complexity. Besides, in [6], authors introduce the notion of *incremental graph matching*. This allows, in case of changes occurred in two already paired graphs, to avoid recalculating the full matching based on the former calculated matching.

The AC-projection operator [14] differs from these redefinitions of graph and subgraph isomorphism [7,8] regarding the following aspects. First, the AC-projection is not based on a graph distance measure or a threshold during the matching process. This lessen the burden of additional parameters specification. Then, the AC-projection operator is granted with a panoply of attractive properties. These properties enable us, for example, to compute the minimal graph representative of an equivalence class of graphs as presented in previous sections.

4.4 The FGMAC algorithm outline

The FGMAC algorithm [4] initially enumerates all the frequent single and double edge graphs. Then, based on those two sets, it starts the main computational loop. During each iteration, it first generates candidate subgraphs whose size is greater than the previous frequent ones by one edge. Next, it counts the frequency for each of these candidates and prunes subgraphs that do not satisfy the support constraint. Discovered frequent subgraphs fulfill the downward closure property of the support condition, which enables an efficient pruning of the lattice of frequent subgraphs.

The FGMAC's particularity is to output only frequent AC-reduced graphs which is a subset of the whole frequent isomorphic pattern set. The reader can find a complete description of the algorithm in [4].

5 AC-miner: a pattern-growth graph mining approach with a polynomial time projection

In this section, we will present, AC-miner [5], a basic pattern-growth instance of the LC- MINE framework for frequent AC-reduced subgraphs mining. This approach is based on three key concepts: the AC-extension operator (member of the LC- MINE framework), the increasing forbidden labels inheritance and the decreasing allowed vertices inheritance. These concepts are presented in the following section.

5.1 The AC-extension operator

The AC-extension is the core operator of the AC-miner algorithm. Given a graph G, the AC-extension operator is intended to extend G by an extra edge such that the constraint network $\mathcal{N}'$ associated with the extended graph G' remains arc consistent.

5.1.1 Mathematical foundations

Definition 5.1 (*Graph database*) A graph database $\mathscr{D}$ is a set of labeled graphs.

Definition 5.2 (*Vertex group and most general vertex group*) Given a graph database $\mathscr{D}$, a vertex group $\mathcal{V}^l$ is a set of vertices that belong to graphs in $\mathscr{D}$ with the same vertice label l. The most general vertex group $\widehat{\mathcal{V}^l}$ is the maximal vertex group of a given label l.

Definition 5.3 (*Vertex group support*) Let $\mathcal{V}^l$ be a vertex group in a graph database $\mathscr{D}$. The support of $\mathcal{V}^l$, which is denoted by $sup(\mathcal{V}^l)$, is the set of graphs in $\mathscr{D}$ such that each graph contains at least one vertex $v \in \mathcal{V}^l$.

We note that we are using a minimal support parameter σ as a bias which limits the search space.

Definition 5.4 ($^{\sigma}$*AC-compatible* $\overset{\sigma}{\frown}$) Let $\mathcal{V}^a$ and $\mathcal{V}^b$ be two vertex groups in a graph database $\mathscr{D}$.

$\mathcal{V}^a$ is $^{\sigma}$AC-compatible with $\mathcal{V}^b$ if and only if:

- for all $x_k \in \mathcal{V}^a$, there exists $y_p \in \mathcal{V}^b$ such that x_k and y_p are neighbors.
- for all $y_q \in \mathcal{V}^b$, there exists $x_m \in \mathcal{V}^a$ such that x_m and y_q are neighbors.
- $|sup(\mathcal{V}^a)| \geq \sigma \times |\mathscr{D}|$.

The $^{\sigma}$AC-compatibility between $\mathcal{V}^a$ and $\mathcal{V}^b$ is denoted by $\mathcal{V}^a \overset{\sigma}{\frown} \mathcal{V}^b$.

Let $\mathcal{V}^a$ and $\mathcal{V}^b$ be two vertex groups that are labeled a and b, respectively. The AC-extension consists in finding the two maximal subsets of $\mathcal{V}^a$ and $\mathcal{V}^b$ that are $^{\sigma}$AC-compatible. This can be formalized in the definition below.

Definition 5.5 (*AC-extension* $\overset{\sigma}{\rightsquigarrow}$) Let $\mathcal{V}^a$ and $\mathcal{V}^b$ be two non-empty vertex groups of a graph database $\mathscr{D}$. The AC-extension from $\mathcal{V}^a$ to $\mathcal{V}^b$ consists in deriving the two maximal subsets $\mathcal{V}^a_{child}$ et $\mathcal{V}^b_{child}$ such that:

- $\mathcal{V}^a_{child} \overset{\sigma}{\frown} \mathcal{V}^b_{child}$.
- $\nexists \mathcal{V}^{a\prime}_{child} \subset \mathcal{V}^a$, $\mathcal{V}^{b\prime}_{child} \subset \mathcal{V}^b$ with $\mathcal{V}^{a\prime}_{child} \supset \mathcal{V}^a_{child}$ or $\mathcal{V}^{b\prime}_{child} \supset \mathcal{V}^b_{child}$ such that $\mathcal{V}^{a\prime}_{child} \overset{\sigma}{\frown} \mathcal{V}^{b\prime}_{child}$.

The possibility of such an AC-extension is denoted by $\mathcal{V}^a \overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$, while the impossibility of such relation is denoted by $\mathcal{V}^a \not\overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$.

5.1.2 Increasing forbidden labels inheritance

Proposition 5.1 *Let $\mathcal{V}^a$ and $\mathcal{V}^b$ be two non-empty vertex groups of a graph database $\mathscr{D}$. If $\mathcal{V}^a \not\overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$ then $\forall \mathcal{V}^{a\prime} \subseteq \mathcal{V}^a$, we have $\mathcal{V}^{a\prime} \not\overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$.*

Proof According to Definition 5.5, in order to have $\mathcal{V}^{a\prime} \overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$, we must have $\mathcal{V}^{a\prime}_{child} \subseteq \mathcal{V}^{a\prime}$ and $\mathcal{V}^b_{child} \subseteq \mathcal{V}^b$ such that $\mathcal{V}^{a\prime}_{child} \overset{\sigma}{\frown} \mathcal{V}^b_{child}$. But, since $\mathcal{V}^a \not\overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$ then, according to Definition 5.5, we have:

$$\forall \mathcal{V}^a_{child} \subseteq \mathcal{V}^a \text{ and } \forall \mathcal{V}^b_{child} \subseteq \mathcal{V}^b, \text{ we have } \mathcal{V}^a_{child} \not\overset{\sigma}{\frown} \mathcal{V}^b_{child}. \quad (1)$$

As, by transitivity, we have $\mathcal{V}^{a\prime}_{child} \subseteq \mathcal{V}^a$ (since $\mathcal{V}^{a\prime}_{child} \subseteq \mathcal{V}^{a\prime}$ and $\mathcal{V}^{a\prime} \subseteq \mathcal{V}^a$) then, according to (1), there is no vertex group $\mathcal{V}^{a\prime}_{child}$ which is included in $\mathcal{V}^{a\prime}$ that can be $^{\sigma}$AC-compatible with any other vertex group $\mathcal{V}^b_{child}$ which is included in $\mathcal{V}^b$. Hence, $\mathcal{V}^{a\prime} \not\overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$. □

According to Proposition 5.1, every vertex group $\mathcal{V}$ can transmit its forbidden labels list to every child $\mathcal{V}_{child} \subset \mathcal{V}$. Therefore, the AC-miner algorithm will not try an AC-extension of $\mathcal{V}$ children's with forbidden labels. The size of this list will increase from child to child.

We have experimentally proved that, using the forbidden labels inheritance, AC-miner avoids on average up to 20 % of non-useful AC-extensions trials (see Sect. 5.2.1).

5.1.3 Decreasing allowed vertices inheritance

Proposition 5.2 *Let $\mathcal{V}^a$ and $\mathcal{V}^b$ be two vertex groups of a graph database $\mathscr{D}$. Let us suppose that $\mathcal{V}^a \overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$ and that the two vertex groups $\mathcal{V}^a_{child}$ and $\mathcal{V}^b_{child}$ are the maximal subsets of $\mathcal{V}^a$ and $\mathcal{V}^b$, respectively, such that $\mathcal{V}^a_{child} \overset{\sigma}{\frown} \mathcal{V}^b_{child}$. For all $\mathcal{V}^{b\prime}_{child} \supset \mathcal{V}^b_{child}$, there does not exists any vertex group $\mathcal{V}^{a\prime}_{child} \subseteq \mathcal{V}^a_{child}$ such that $\mathcal{V}^{a\prime}_{child} \overset{\sigma}{\frown} \mathcal{V}^{b\prime}_{child}$. This says that the vertex group $\mathcal{V}^{a\prime}_{child}$ cannot be $^{\sigma}$AC-compatible with any superset of $\mathcal{V}^b_{child}$.*

Proof Let us verify whether there exists a vertex group $\mathcal{V}^{a\prime}_{child} \overset{\sigma}{\frown} \mathcal{V}^{b\prime}_{child}$ with $\mathcal{V}^{b\prime}_{child} \supset \mathcal{V}^b_{child}$. We have $\mathcal{V}^a \overset{\sigma}{\rightsquigarrow} \mathcal{V}^b$ then, according to Definition 5.5, there is no vertex group $\mathcal{V}^{a''}_{child} \supset \mathcal{V}^a_{child}$ or $\mathcal{V}^{b''}_{child} \supset \mathcal{V}^b_{child}$ with $\mathcal{V}^{a''}_{child} \subseteq \mathcal{V}^a$ and $\mathcal{V}^{b''}_{child} \subseteq \mathcal{V}^b$, respectively, such that $\mathcal{V}^{a''}_{child} \overset{\sigma}{\frown} \mathcal{V}^{b''}_{child}$. Hence, $\mathcal{V}^{a\prime}_{child} \overset{\sigma}{\not\frown} \mathcal{V}^{b\prime}_{child}$. □

According to Proposition 5.2, every vertex group $\mathcal{V}$ can transmit its allowed vertices list to every child $\mathcal{V}_{child} \subset \mathcal{V}$. Therefore, the AC-miner algorithm will only try an AC-extension of $\mathcal{V}$ children's with a subset of $\widehat{\mathcal{V}^l}$. The size of this list will decrease from child to child.

With allowed vertices inheritance, AC-miner will not avoid AC-extensions test, but will speed up this test from child to child as $\mathcal{V}^l$ gets smaller. This fact is experimentally proved in Sect. 5.2.2.

5.2 Experimental evaluation of the AC-extension pruning techniques

In this section, we experimentally evaluate the impact of pruning techniques, which were previously presented, on the performance of the algorithm AC-miner. First, we proceed to the evaluation of gains generated by using the technique of forbidden labels inheritance. This evaluation is about computing the number of extensions performed by AC-miner algorithm as well as the number of AC-extensions that were avoided due to our pruning technique. Then, we evaluate the average number of vertices that were handled during AC-extensions operations. The evaluation concerns both settings where the allowed vertices inheritance technique is used or not.

5.2.1 The forbidden labels inheritance

To experimentally evaluate the forbidden labels inheritance technique adopted by the AC-miner algorithm, we computed the number of AC-extensions that were performed or avoided during the process of frequent subgraphs mining. In our first series of experiments, we set the minimal support to 50 %. The results of these experiments are shown in Table 2. The first column of this table lists the graph databases that are used. Then, for each of these graph databases, we give the number of AC-extensions performed, the number of forbidden AC-extensions that were avoided and the gain percentage. We note that the gains range between 10 and 33 % with an average of 20 %. This is true for both the large graph database named

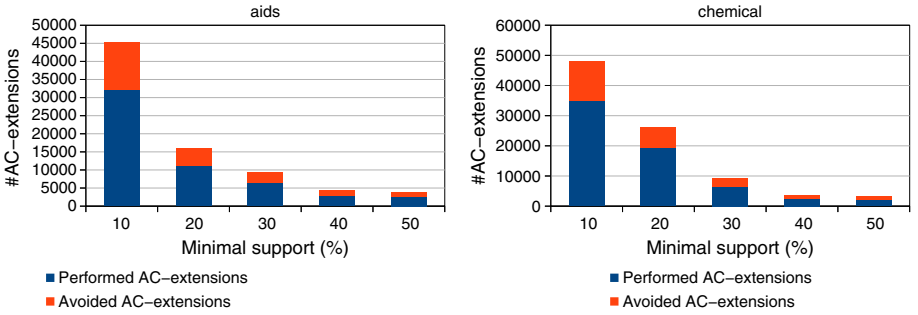


Fig. 6 Comparison of the number of forbidden AC-extensions operations avoided with the number of AC-extensions operations performed by the AC-miner algorithm using different minimal supports over *aids* and *chemical* datasets

aids and the small graph database named *chemical* with gains that can reach up to 33 % (i.e., one third of the total number of AC-extensions operations) for a minimal support equal to 50 %. The gain is around 10 % for the *nci* graph databases group.

Comparison of the number of forbidden AC-extensions operations avoided with the number of AC-extensions operations performed by the AC-miner algorithm using different minimal supports over *aids* and *chemical* datasets.

Afterward, we concluded our study by a second series of experiments that allow us to assess the achieved gains with different values of minimal support. To do this, we selected a large and a small dataset : aids and chemical. In this series of experiments, and for each dataset, we computed the number of AC-extensions performed during the graph mining process by varying, at each time, the minimal support between 10 and 50 %. The results are depicted in Fig. 6. This figure shows two histograms, one for each dataset. Each histogram shows the proportion of forbidden AC-extensions that were avoided within to AC-extensions that were performed. The results of this series of experiments show that this pruning method can achieve a gain with all values of minimal support. This confirms the results obtained in the previous series of experiments and shows experimentally that, on average, the AC-miner algorithm allows avoiding 20 % of the total number of useless AC-extensions operations with the forbidden labels inheritance pruning technique.

5.2.2 The allowed vertices inheritance

Unlike the pruning technique based on forbidden labels inheritance, whose aim is to reduce the number of AC-extensions that have to be performed during the graph mining process, the pruning technique based on allowed vertices inheritance allows to reducing the number of vertices handled during each AC-extension operation. To evaluate the effectiveness of this technique, we followed the same experimental scheme as that used for the evaluation of forbidden labels inheritance technique. However, for this series of experiments, our metric is based not on the number of AC-extensions performed but on the average number of vertices handled by the AC-extension operations. The results of this series of experiments are reported in the second part of Table 2. This part presents the average number of allowed vertices, the average total number of handled vertices per AC-extension operation and the gain as a percentage. The results show that the average number of allowed vertices is smaller than the average total number of vertices by a factor of three. Gains reach over 70 % for all datasets that we considered. This finding is also supported by the second series of experiments

Table 2 Comparison of the number of forbidden AC-extensions operations avoided with the number of AC-extensions operations performed by the AC-miner algorithm and comparison of the average number of allowed vertices with the average total number of vertices handled within one AC-extension operation

Dataset	#AC-extensions			#Vertices/AC-extension		
	Performed	Avoided	Gain (%)	Allowed	Total	Gain (%)
aids	2,499	1,221	32.82	1,817	6,753	73.09
chemical	2,120	1,048	33.08	17	61	71.65
nci1	104,658	11,982	10.27	622	2,379	73.88
nci145	112,430	12,067	9.69	602	2,290	73.73
nci33	97,984	11,216	10.27	526	2,029	74.09

The associated gains are presented in %. The minimal support has been fixed to 50 %

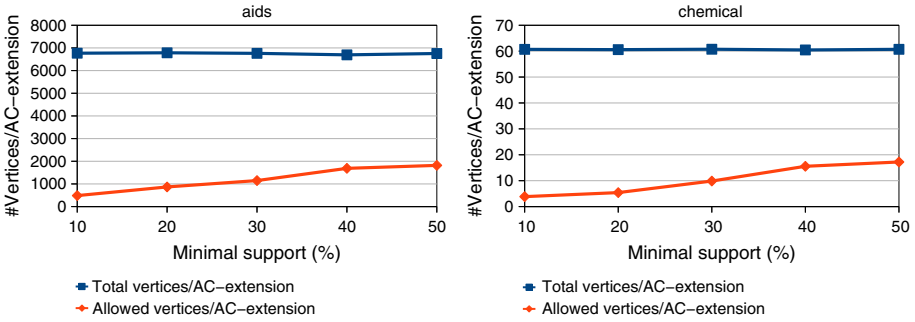


Fig. 7 Comparison of the average number of allowed vertices with the average total number of vertices handled within one AC-extension operation using different minimal supports over *aids* and *chemical* datasets

whose results are reported in Fig. 7. In this series of experiments, we have, for each database graph, varied the minimal support from 10 to 50 % with recording, at each time, the average number of allowed vertices and the average total number of vertices handled per AC-extension operation. These results are represented by two graphs: one graph for each dataset. Each graph shows two curves: the first translates the evolution of the total number of vertices that are handled by the AC-extension operation, while the second shows the evolution of the average number of the allowed vertices that are handled by the AC-extension. This is done for each minimal support value that is considered in these settings.

These results confirm our observations reported in Table 2 and allow us to experimentally prove that our pruning technique based on allowed vertices inheritance achieves a significant gain of more than 70 %. This achievement takes place regardless of the value of minimal support adopted or the dataset considered.

5.3 The AC-miner algorithm outline

The AC-miner algorithm [5] starts by adding to each vertex label in a graph database $\mathscr{D}$ its associated most general vertex group $\mathcal{V}^l$ in a list. This list contains the remaining vertex group to be extended. Then, the algorithm tries to extend each vertex group till it has no further possible extension. So, we can say that the resulting frequent patterns are said to be closed.[2]

[2] A frequent graph pattern is said to be closed, if there is no super frequent graph pattern with the same support.

Table 3 Datasets statistics

Dataset	Graphs	Vertex/Graph		Edge/Graph		Labels		Classes
		avg	max	avg	max	vertex	edge	
nci1	11,272	27	113	29	119	54	3	2
nci33	9,201	27	113	29	119	52	3	2
nci41	8,404	28	113	30	119	34	3	2
nci47	11,165	27	111	29	119	53	3	2
nci81	13,017	27	111	29	119	56	3	2
nci109	11,351	27	111	29	119	54	3	2
nci145	10,719	27	110	29	116	53	3	2
nci330	12,506	23	120	24	132	57	3	2
aids	42,285	26	222	28	247	62	4	3
dd	1,178	284	5,748	716	14,267	82	1	2

The reader can find a complete description of the AC-miner algorithm as well as a running example in [5].

6 Experiments and comparative study

In order to prove the usefulness of the LC- MINE framework for frequent subgraph mining, we present in the following an experimental study of the two previously described instances, namely the FGMAC and AC-miner algorithms. We insist that the set of frequent AC-reduced graphs discovered by these approaches is not exhaustive w.r.t. isomorphic patterns. So, in the following, we present a quantitative study of our instances performance followed by a qualitative evaluation of the AC-reduced patterns which consists in a computation of their discriminative power within a supervised graph classification process.

6.1 Datasets

In order to evaluate the computational and the qualitative aspects of the FGMAC and AC-miner instances, as well as the AC-projection operator, we carried out classification experiments on 10 real-world datasets group widely cited in the literature (*cf.* Table 3):

- The anti-cancer screen datasets (nci) consist in eight datasets collected from the PubChem Web site as in [22].
- The AIDS antiviral screen data (aids) contain the activity test information of 42,285 chemical compounds. Each chemical compound is labeled as either active, moderately active or inactive with respect to the HIV virus.
- The Dobson and Doig molecule dataset (dd) contain 1,178 proteins. Each protein is labeled as either enzyme or non-enzyme. These proteins are represented by larger and more densely connected graphs than the other datasets.

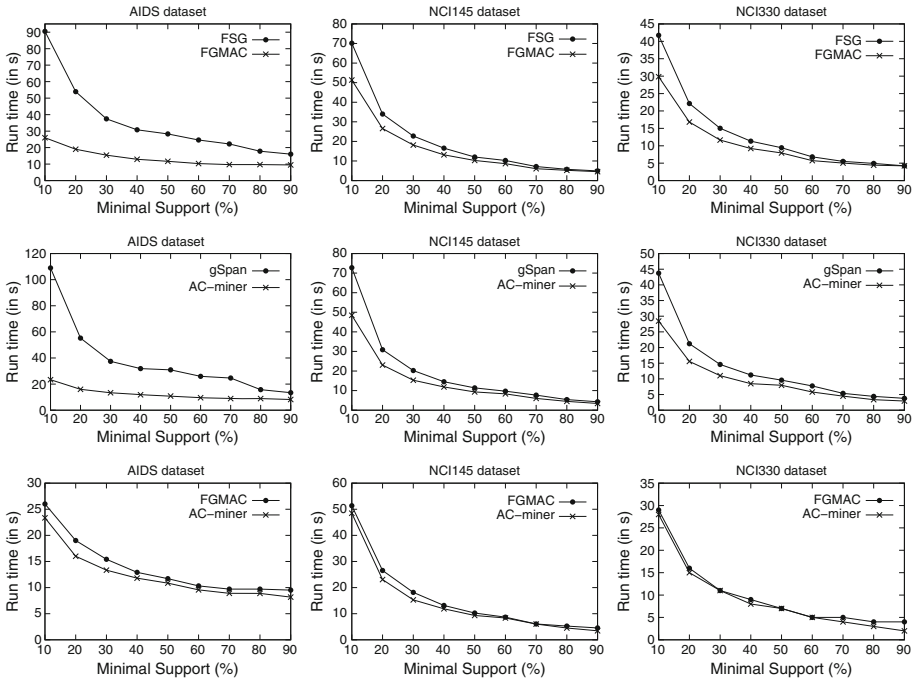


Fig. 8 Runtime comparison of FGMAC versus FSG (*line 1*), AC-miner versus gSpan (*line 2*) and FGMAC vs AC-miner (*line 3*), with the three datasets AIDS, NCI145 and NCI330 (*columns*)

6.2 Performance point of view

In order to have an idea about the very interesting computational power of each one of the two developed instances, we compare their performances with a state-of-the-art mining pattern-growth algorithm named gSpan [25] as well as the popular Apriori-like algorithm named FSG [13]. These approaches are placed among the most efficient graph miners in their respective categories [13,24]. In this subsection, we present a quantitative study of the computational performance of the FGMAC algorithm compared to FSG, AC-miner compared to gSpan and, finally, FGMAC compared to AC-miner.

The key parameter of the graph mining algorithms is the minimal support used to discover the frequent substructures. To evaluate the performance of our framework instances with regard to this parameter, we performed a set of experiments in which we varied the minimal support from 10 to 90 % in 10 % increments.

Results depicted in Fig. 8 clearly show that, AC- miner and FGMAC, respectively, outperform gSpan and FSG regarding the runtime for all minimal supports selected and validate the theoretical results about the polynomiality of the AC-projection operator compared to the exponential complexity of the subgraph isomorphism adopted by gSpan and FSG.

However, looking at Fig. 9, we can see that for the dd dataset, FGMAC has similar runtime with FSG for every minimal support. After thorough analysis of these results, we can conclude that, with dense datasets having too much labels (82 labels for the dd dataset) subgraph isomorphism becomes relatively more efficient (because of the increased number of vertex invariants [13,19] that considerably reduce backtracks). Consequently, the performance of the AC-projection will be almost the same as the subgraph isomorphism. Therefore, frequent

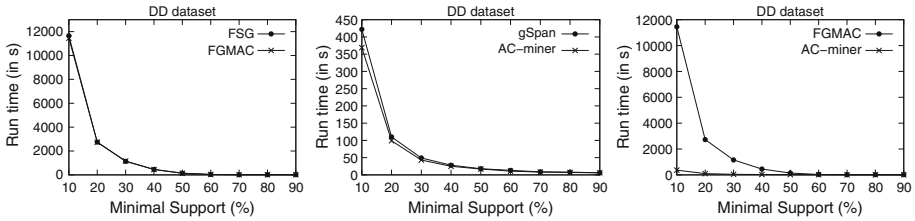


Fig. 9 Runtime comparison of FGMAC versus FSG (*left*), AC-miner versus GSPAN (*center*) and FGMAC vs AC-miner (*right*) with the dd dataset

subgraph mining time for classical approaches will be reduced. Nevertheless, AC-miner is a bit faster than GSPAN on the same dataset, because of its reduced search space compared to the large isomorphism-wide GSPAN's one.

When comparing AC-miner and FGMAC results over the dd datasets (*cf.* Fig. 9), we can see that AC-miner is by far more efficient with this kind of dense datasets. For example, we can see that, for the 10 % minimal support, AC-miner is a thousand times efficient as FGMAC. This is principally due to the costly candidate generation phase adopted by FGMAC which faces combinatorial problem and generates millions of false-positive candidates. This is not the case of AC-miner, which explores the search space in a depth-first manner and without candidate generation. We can conclude that AC-miner scales well for big dense datasets compared to FGMAC.

In the following section, we present a study in a qualitative point of view of frequent AC-reduced patterns.

6.3 Qualitative point of view: graph classification

Graph classification is a supervised learning problem in which the goal is to categorize an entire graph as a positive or negative instance of a concept. We are particularly interested in feature mining on graphs as it uses frequent graph patterns in the classification process. Feature mining on graphs is usually performed to find all frequent or informative substructures in the graph instances. These substructures are used for transforming the graph data into data which is represented by a single table, and then traditional classifiers are used for classifying the instances.

In this paper, the aim of using graph classification is the evaluation of the quality and discriminative power of frequent AC-reduced subgraph patterns and its comparison with isomorphic frequent subgraphs.

Using accuracy to judge a classifier would be incorrect as the size of the positive class is significantly smaller than the negative class. To get a better understanding of the classifier performance for different cost settings, we obtain the ROC curve [17] for each classifier. ROC curve plots the false-positive rate (X-axis) versus the true-positive rate (Y-axis) of a classifier; it displays the performance of the classifier regardless of class distribution or error cost. Two classifiers are evaluated by comparing the area under their respective ROC curves, a larger area under ROC curve indicating better performance. The area under the ROC curve will be referred to by the parameter AUC. In the following experimentations, it is worth mentioning that we are not comparing different classifiers but different feature sets (i.e., isomorphic and AC-reduced) using only one classifier.

We carried out classification experiments on five biological activity datasets and measured the AUC classification value using the known decision trees classifier, namely C4.5 [18]. The

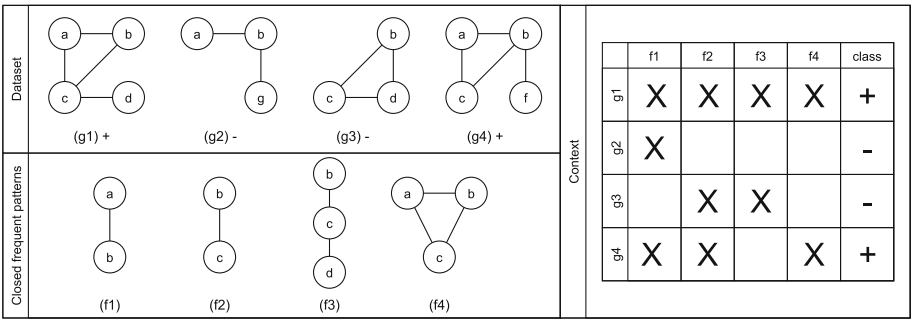


Fig. 10 Graph classification process

classification methods are described in more detail in the following subsections, along with the associated results.

6.3.1 Methods

We evaluated the classification AUC using four different feature sets. The first two sets of features (isomorphic, closed isomorphic), respectively, consist of all frequent isomorphic subgraphs and all closed frequent isomorphic ones. Those subgraphs are mined using the FSG software [13] with different minimal supports. Each chemical compound is represented by a binary vector with a length equal to the number of mined subgraphs. Each subgraph is mapped to a specific vector index, and if a chemical compound contains a subgraph, then the bit at the corresponding index is set to one, otherwise it is set to zero.

The third and fourth feature sets (AC-reduced, closed AC-reduced), respectively, contain the FGMAC's output consisting of only AC-reduced frequent subgraphs and the AC- MINER's output which consists of only closed AC-reduced frequent subgraphs. We have used the method described above with the only difference of using the AC-projection instead of the subgraph isomorphism as the projection operator.

The graph classification preprocessing over an example is depicted in Fig. 10. The final matrix (the context) will be used by the C4.5 [18] decision tree algorithm.

6.3.2 Results

All classifications have been done with the Weka data-mining software package [23], and we have reported results of the prediction AUC value over the 10 cross-validation trials. In the following, we are analyzing the AC-reduced patterns from quantitative and qualitative points of view.

(Patterns count) Results presented in Fig. 11 show that for all datasets, we have very few AC-reduced frequent patterns compared to the isomorphic ones. On average, we have 35 % fewer patterns. This ratio is bigger for lower supports and can reach 60 % for the AIDS dataset with a minimal support of 10 %. These experimental results highlight the fact that the search space for extracting AC-reduced patterns is smaller than the one for classical isomorphic subgraphs. Thus, having an algorithm which looks for all AC-reduced frequent subgraphs would benefit from the polynomiality of the projection operation as well as a smaller search space (*i.e.* fewer AC-projection tests).

Table 4 Comparison of the classification AUC value of different feature sets for all the ten datasets with a minimal support of 50 %

Datasets	Isomorphic		AC-reduced		Closed Isomorphic		Closed AC-reduced	
	AUC	Std	AUC	Std	AUC	Std	AUC	Std
aids	0.688	0.012	0.683	0.012	0.618	0.019	0.618	0.019
dd	0.815	0.048	0.815	0.052	0.812	0.043	0.814	0.044
nci109	0.752	0.017	0.743	0.014	0.723	0.020	0.712	0.022
nci145	0.755	0.016	0.745	0.017	0.716	0.021	0.707	0.021
nci1	0.758	0.020	0.747	0.020	0.718	0.017	0.711	0.013
nci330	0.734	0.022	0.736	0.023	0.700	0.020	0.685	0.019
nci33	0.755	0.025	0.746	0.026	0.723	0.022	0.713	0.022
nci41	0.743	0.023	0.737	0.023	0.717	0.031	0.712	0.031
nci47	0.751	0.017	0.735	0.016	0.725	0.021	0.710	0.020
nci81	0.735	0.017	0.728	0.015	0.702	0.014	0.691	0.009

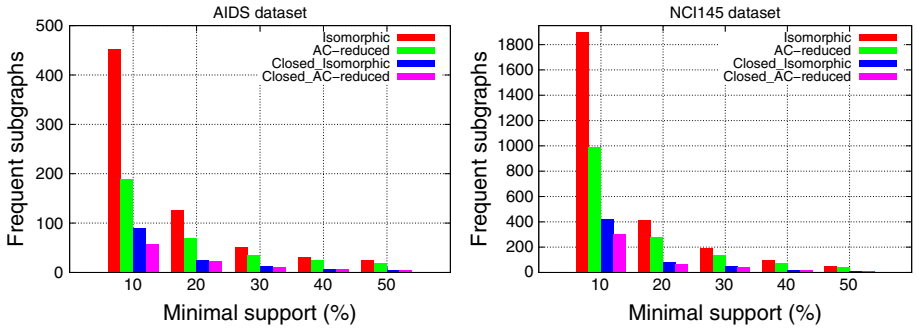


Fig. 11 Comparison of the number of patterns of different feature set for the AIDS and NCI145 datasets

(Classification relevance) When we see that the number of frequent subgraph patterns has drastically decreased after the AC-reduction process, we are surely curious to know about the relevance of these few patterns for supervised graph classification. That is why we have conducted experiments to evaluate the accuracy of classification. These experiments use (closed) AC-reduced and (closed) isomorphic patterns to compare them.

As shown in Fig. 12, when comparing frequent isomorphic patterns to frequent AC-reduced ones and closed frequent isomorphic patterns to closed frequent AC-reduced ones, we can see that, the AUC value is almost the same for all minimal support and all datasets. Table 4 confirms our claim that, despite the fact that frequent AC-reduced patterns are clearly fewer than isomorphic ones, their quality and discriminative power remain almost the same.

Taking a more in-depth look at the results, we notice that, for some datasets and minimal support values, we have even better AUC for AC-reduced and closed AC-reduced feature sets. This is eventually due to the better generalization power of the AC-reduction process, which helped supervised classifiers avoid the over-fitting learning problem.

Finally, comparing AC-reduced and closed AC-reduced feature sets, we can see that frequent AC-reduced patterns can characterize graphs better than closed ones. This is due to the

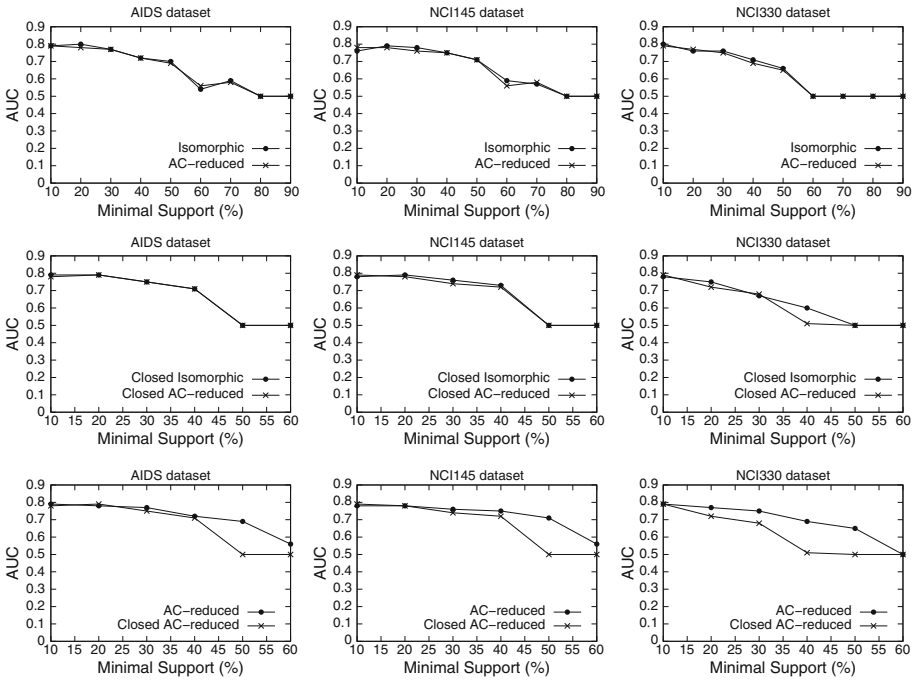


Fig. 12 Comparison of the classification AUC value of different feature sets for AIDS, NCI145 and NCI330 datasets

clearly insufficient number of closed AC-reduced patterns especially for high minimal supports (*cf.* Fig. 11). Such a reduced number of features does not permit to correctly characterize graphs in the learning process. However, it is possible to change the AC-miner algorithm to derive all frequent AC-reduced patterns and not only closed ones with no coding overhead. Therefore, we will have a very efficient mining approach with a more exhaustive frequent patterns output.

7 Conclusion

In this paper, we proposed a generic and efficient framework for frequent subgraph mining with local consistency techniques, namely LC- MINE. The core idea behind our framework is the introduction of a bias in the graph projection operator. Our approach can be easily adapted to any level of local consistencies (arc consistency, path consistency, etc.). We propose two arc consistency-based instances of the LC- MINE framework. The first instance, named FGMAC, follows a breadth-first order to find frequent subgraphs and uses an APRIORI-like search strategy. The second, named AC-miner, is a pattern-growth approach that follows a depth-first search space exploration strategy and uses powerful pruning techniques in order to considerably reduce the search space. We evaluated their performance in terms of the required computation time and the discovered frequent patterns quality for some real-world datasets. The computational efficiency of our approaches was confirmed, and our framework instances outperform state-of-the-art approaches. It is worth mentioning that the number of discovered frequent AC-reduced subgraphs is clearly smaller than isomorphic ones but they have a very comparable quality and discriminative power.

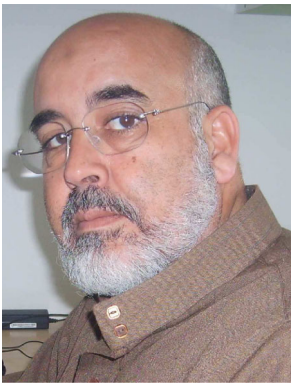
References

1. Agrawal R, Skirant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large databases. Santiago, Chile, pp 478–499
2. Bessière C, Régin JC (1996) Mac and combined heuristics: two reasons to forsake fc (and cbj?) on hard problems. In 'CP', pp 61–75
3. Cook JD, Holder LB (2006) Mining graph data. Wiley, London
4. Douar B, Liquiere M, Latiri C, Slimani Y (2011a), FGMAC: Frequent subgraph mining with Arc Consistency. In: Proceedings of the IEEE symposium on computational intelligence and data mining, CIDM 2011, part of the IEEE symposium series on computational intelligence. IEEE Computer Society, Paris, pp 112–119
5. Douar B, Liquiere M, Latiri C, Slimani Y (2011b) Graph-based relational learning with a polynomial time projection algorithm. In: Proceedings of the 21st international conference on inductive logic programming, ILP 2011, vol 7207 of LNAI. Springer, Windsor Great Park, pp 96–112
6. Fan W, Li J, Luo J, Tan Z, Wang X, Wu Y (2011) Incremental graph pattern matching. In: Proceedings of the 2011 international conference on Management of data, SIGMOD '11. ACM, New York, pp 925–936
7. Fan W, Li J, Ma S, Tang N, Wu Y, Wu Y (2010) Graph pattern matching: from intractable to polynomial time. Proc. VLDB Endow. 3(1–2):264–275
8. Fan W, Li J, Ma S, Wang H, Wu Y (2010) Graph homomorphism revisited for graph matching. Proc. VLDB Endow. 3(1–2):1161–1172
9. Hell P, Nesetril J (2004) Graphs and homomorphism, vol 28. Oxford University Press, Oxford
10. Huan J, Wang W, Prins J (2003) Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceedings of the 3rd IEEE international conference on data mining, ICDM '03, IEEE computer society, Washington p 549
11. Inokuchi A, Washio T, Motoda H (2003) Complete mining of frequent patterns from graphs: mining graph data. Mach. Learn. 50(3):321–354
12. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Cercone N, Lin TY, Wu X (eds) International conference on data mining, IEEE computer society, pp 313–320
13. Kuramochi M, Karypis G (2004) An efficient algorithm for discovering frequent subgraphs. IEEE Trans Knowl Data Eng 16:1038–1051
14. Liquiere M. (2007) Arc consistency projection: a new generalization relation for graphs. In: ICCS, pp 333–346
15. Mackworth AK (1977) Consistency in networks of relations. Artif. Intell. 8(1):99–118
16. Nijssen S, Kok JN (2004) The gaston tool for frequent subgraph mining. In: International workshop on graph-based tools (Grabats). Electronic notes in theoretical computer science, pp 77–87
17. Provost FJ, Fawcett T (2001) Robust classification for imprecise environments. Mach. Learn. 42(3):203–231
18. Quinlan JR (1993) C4.5: programs for machine learning, 1st edn. Morgan Kaufmann, Burlington
19. Read RC, Corneil DG (1977) The graph isomorphism disease. J. Graph Theory 1(1):339–363
20. Rossi F, van Beek P, Walsh T (eds) (2006) Handbook of constraint programming. Elsevier, Amsterdam
21. Solnon C (2010) Alldifferent-based filtering for subgraph isomorphism. Artif. Intell. 174:850–864
22. Thoma M, Cheng H, Gretton A, Han J, Kriegel HP, Smola A, Song L, Yu PS, Yan X, Borgwardt KM (2010) Discriminative frequent subgraph mining with optimality guarantees. Stat. Anal. Data Min. 3(5):302–318
23. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, second edition (Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, San Francisco
24. Wörlein M, Meinl T, Fischer I, Philippsen M (2005) A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In: European conference on machine learning and principles and practice of knowledge discovery in databases, vol 3721 of LNCS, Springer, Berlin pp 392–403
25. Yan X, Han J (2002) gspan: Graph-based substructure pattern mining. In: International conference on data mining, IEEE computer society, pp 721–724
26. Zampelli S, Deville Y, Solnon C (2010) Solving subgraph isomorphism problems with constraint programming. J Constraints 15:327–353

Brahim Douar received a Computer Science Ph.D. from the University of Montpellier II and the University of Tunis El Manar in 2012. He also received B.S. and M.S. degrees in Computer Science from the Tunis Higher Institute of Management, in 2005 and 2007, respectively. Actually, he is a researcher in the LIPAH research Lab at the Faculty of sciences of Tunis. His research interests include business intelligence, relational learning, graph mining and social network analysis.

Michel Liquiere is a Senior Lecturer in Computer Science at the LIRMM (University Montepllier II and IUT Beziers), Since 1990, his research interest focuses on developing novel algorithms in machine learning, formal concept analysis and graph mining.

Chiraz Latiri is Professor in Computer Science at Manouba University (Tunisia). She is a head of the Text mining team within the LIPAH research Lab at the Faculty of sciences of Tunis. His research focuses on mining of large multilingual document collections for information access (mainly mining interesting patterns form text and information retrieval) and is at the intersection of three main research domains: text mining, information retrieval and machine translation. She particularly interested in modeling and mining large textual collections and in proposing new approaches that take into account the properties of such collections such as multilingualism. She is also interested in extracting and modeling text corpora from social networks.



Yahya Slimani studied at the Computer Science Institute of Algers (Algeria) from 1968 to 1973. He received the B.Sc. (Eng.), Dr Eng and PhD degrees from the Computer Science Institute of Algers (Algeria), University of Lille (French) and University of Oran (Algeria), in 1973, 1986 and 1993, respectively. He is currently Full Professor at the High Institute of Art and Multimedia of University of Manouba (Tunisia). His research activities concern knowledge extraction, data mining, parallelism, distributed systems, Grid and Cloud computing. Pr. Slimani has published more than 120 papers from 1986 to 2013. He is a distinguished editorial board member and reviewer for many scientific journals and international conferences.