# A Projection Bias in Frequent Subgraph Mining Can Make a Difference

Brahim Douar, Chiraz Latiri, Michel Liquière, Yahya Slimani

International Journal on Artificial Intelligence Tools


# A PROJECTION BIAS IN FREQUENT SUBGRAPH MINING CAN MAKE A DIFFERENCE

Brahim DOUAR
*LIRMM Research Laboratory, 161 rue Ada 34392 - Montpellier, France*
*douar@lirmm.fr*

Chiraz LATIRI
*LIPAH Research Laboratory, Computer Sciences Department*
*Faculty of Sciences of Tunis,*
*El Manar University, 1068, Tunis, Tunisia*
*chiraz.latiri@gnet.tn*

Michel LIQUIERE
*LIRMM Research Laboratory, 161 rue Ada 34392 - Montpellier, France*
*liquiere@lirmm.fr*

Yahya SLIMANI
*LISI Research Laboratory, INSAT*
*Carthage University, Centre Urbain Nord BP 676, 1080, Tunis, Tunisia*
*yahya.slimani@fst.rnu.tn*

Received (20 May 2012)

The aim of the frequent subgraph mining task is to find frequently occurring subgraphs in a large graph database. However, this task is a thriving challenge, as graph and subgraph isomorphisms play a key role throughout the computations. Since subgraph isomorphism testing is a hard problem, subgraph 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. This paper presents a new projection operator for graphs named AC-projection, which exhibits nice theoretical complexity properties. We study the size of the search space as well as some practical properties of the projection operator. We also introduce a novel breadth-first algorithm for frequent AC-reduced subgraphs mining. Then, we prove experimentally that we can achieve an important performance gain (polynomial complexity projection) without or with non-significant loss of discovered patterns in terms of quality.

*Keywords*: Graph mining; Projection operators; Graph classification.

## 1. Introduction and Motivation

Given an increasing and urgent need to analyze large amount of structured data such as chemical compounds, proteins structures, XML documents, to cite but a

few, graph mining has become a compelling issue in the data mining field [3]. Indeed, discovering frequent subgraphs, i.e., discovering subgraphs which occur frequently enough over the entire set of graphs, 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. This raises a serious problem related to the exponential search space as well as the counting of complete sub-patterns while the kernel of frequent subgraph mining is subgraph isomorphism test. The latter has been proved to be NP-complete [5]. To alleviate the complexity issue, some heuristics based on an incomplete search or some limitations on the class of subgraphs must be introduced.

In this paper, we study an innovative projection operator meant to replace the costly subgraph isomorphism. In the second section, we provide a brief literature review of the subgraph mining field. Then, we present the AC-projection operator initially introduced in [12], as well as its very interesting properties. We propose an efficient graph mining algorithm using the AC-projection operator. Finally, we study the relevance of the AC-reduced patterns for the supervised graph classification.

Notice that, this paper is a wide extension of a previous work we carried out [4].

## 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*

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

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

**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 Figure 1 that G2 is an induced subgraph of G1 while G3 is not an induced one.

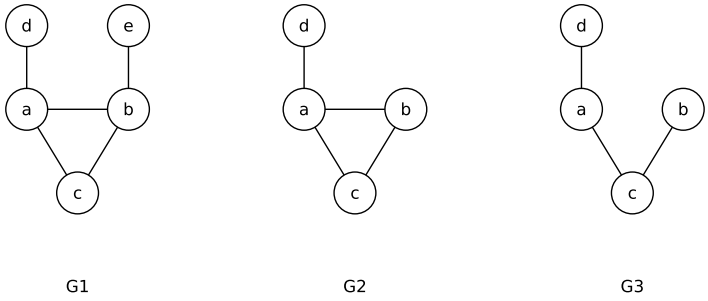


Fig. 1. A set of labeled graphs

**Definition 2.3.** (Isomorphism, Subgraph Isomorphism) Given two graphs $G_1$ and $G_2$, an isomorphism is a bijective function $f : V(G_1) \to V(G_2)$, such that
$\forall x \in V(G_1)$, $l(x) = l(f(x))$, and
$\forall (x, y) \in A(G_1)$, $(f(x), f(y)) \in A(G_2)$ and $l(x, y) = l(f(x), f(y))$.
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) For two graphs, $G_1(V_1, A_1)$ and $G_2(V_2, A_2)$, we call graph projection, the mapping which maps each vertex of $G_1$ into one or many vertices of $G_2$. This defines the generalization order between our graphs.

In this paper, we present a new projection operator (AC-projection) having a polynomial complexity and good properties. It is worth to mention that, given two graphs $G_1$ and $G_2$, if there is no AC-projection between $G_1$ and $G_2$, then there will not be a graph homomorphism between them, then there will also not be a subgraph isomorphism.

**Definition 2.5.** (Frequent Subgraph Mining) Given a graph dataset, GS=$\{G_i |\ i = 0 \dots n\}$, 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 the occurrence frequency of $g$ in $GS$, i.e., the support of $g$ in $GS$. Frequent subgraph mining aims at finding every graph $g$ such that $\sigma(g, GS)$ is greater than or equal to minSup. We note that the value of the support based on AC-projection is an upper bound of the original graph isomorphism. In fact, if

there is no AC-projection between two graphs $G_1$ and $G_2$, then there will be neither graph homomorphism; nor subgraph isomorphism between them.

Known frequent subgraphs 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 patterns 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 [18].

#### 2.2.1. *Breadth-first Approaches*

Algorithms, that follow a breadth-first order to find frequent subgraphs, take advantage of the well-known Apriori [1] levelwise strategy. In the literature, pioneer and most cited approaches are Agm [9] and Fsg [10].

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 [9], show that Agm achieves relatively good performances for synthetic dense datasets. Compared to more recent approaches, with the exception of handling self-loops, Agm is an obsolete mining approach with very poor scalability power.

Thus, a more efficient breadth-first approach named Fsg [10] has been introduced. The latter algorithm is also based on the same level-by-level expansion exactly as did Apriori 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 the candidates.

Various optimizations, partially based 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 it to scale to large graphs. The result of the strong dependency of Fsg to labels and vertex degrees etc., makes the performance of Fsg to be relatively worse for graph database with few vertex and edge labels. This is due to the exponential complexity of the subgraph isomorphism test that badly influenced the overall performance.

Besides, if we have sufficiently 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 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 find 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 gSpan [19] and Gaston [14] algorithms.

The gSpan algorithm [19] is based on a canonical representation for graphs, called *dfs-code*. A dfs-traversal of a graph defines an order in which the edges are visited. 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 appearance lists for each subgraphs. Explicit subgraph isomorphism testing has to be done on all graphs in these appearance lists.

Instead of storing appearance lists for each subgraphs, the Gaston algorithm [14] stores all embeddings [a] 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.

### 2.3. *Critical Discussion*

Developing algorithms that discover all frequently occurring subgraphs in a large graph database is computationally extensive, since 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. Many frequent subgraphs 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 [18], after an extensive experimental study of different subgraph miners, conclude that embedding lists do not

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

considerably speed up the search for frequent fragments. Thus, even though gSpan [19] does not use them, it is almost as competitive as Gaston [14] and Ffsm [8], 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 [12], the author introduced an interesting projection operator named AC-projection which seems to have good properties and ensure polynomial time and memory consumption. The forthcoming sections present this operator with its many interesting properties and show an optimized algorithm for computing it.

## 3. The AC-projection Operator

The projection problem (including conceptual graph projection, homomorphism, injective morphism, Θ-subsumption and OI-subsumption) is crucial to the efficiency of relational learning systems. The approach suggested in [12] advocates a projection operator based on the arc consistency algorithm. This projection method has the required properties: polynomiality, local validation, parallelization and structural interpretation. We note that the name "AC-projection" comes from the classical AC (arc consistency) used in [2].

### 3.1. *AC-projection And Arc Consistency*

We first state the formal definitions related to AC-projection and arc consistency and used in the remainder of the paper.

**Definition 3.1.** (Labeling) Let $G_1$ and $G_2$ be two graphs. We call a labeling from $G_1$ into $G_2$ a mapping $\mathcal{I} : V(G_1) \rightarrow 2^{V(G_2)} | \forall x \in V(G_1), \forall y \in \mathcal{I}(x), l(x) = l(y)$.

Thus, for a vertex $x \in V(G_1).\mathcal{I}(x)$ is a set of vertices of $G_2$ with the same label l(x). We can say that $\mathcal{I}(x)$ is the set of "possible images" of the vertex $x$ in $G_2$.

This first labeling is trivial but can be refined using the neighborhood relations between vertices.

**Definition 3.2.** (AC-compatible $\curvearrowright$ ) Let $G$ be a graph $V_1 \subseteq V(G), V_2 \subseteq V(G)$

$V_1$ is AC-compatible with $V_2$ iff

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

We denote it $V_1 \curvearrowright V_2$

In the definition above we give a direct relation between two sets of vertices $V_1$ and $V_2$. So, for each vertex $x_k$ of $V_1$ there is at least one vertex $y_p$ of $V_2$ which is a neighbor of $x_k$ and, for each vertices $y_q$ of $V_2$, there is at least one vertex $x_m$ of $V_1$ which is a neighbor of $y_q$.

**Definition 3.3.** (Consistency for one arc) Let $G_1$ and $G_2$ be two graphs. We say that a labeling $\mathcal{I} : V(G_1) \to 2^{V(G_2)}$ is consistent with an arc $(x, y) \in A(G_1)$, iff $\mathcal{I}(x) \curvearrowright \mathcal{I}(y)$.

**Definition 3.4.** (AC-labeling) Let $G_1$ and $G_2$ be two graphs. A labeling $\mathcal{I}$ from $G_1$ into $G_2$ is an AC-labeling iff $\mathcal{I}$ is consistent with all the arcs $e \in A(G_1)$.

**Definition 3.5.** (AC-projection $\rightharpoonup$ ) Let $G_1$ and $G_2$ be two graphs. An AC-labeling $\mathcal{I} : V(G_1) \to 2^{V(G_2)}$ is an AC-projection iff $\forall$ AC-labeling $\mathcal{I}' : V(G_1) \to 2^{V(G_2)}$ and $\forall x \in V(G_1)$, $\mathcal{I}'(x) \subseteq \mathcal{I}(x)$. We denote it $G_1 \rightharpoonup G_2$

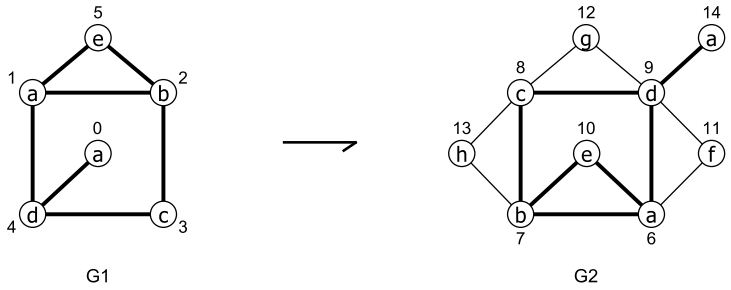


Fig. 2. An AC-projection example ($G_1 \rightharpoonup G_2$)

As depicted in Figure 2, considering 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 labeling consistent with all arcs of $G_1$ and this labeling is maximal.

### 3.2. *AC-projection: Improved Algorithm*

We introduce an improved AC-projection algorithm for graphs (based on the AC3 algorithm [13]). The AC-projection algorithm takes two graphs $G_1$ and $G_2$ and tests whether there is an AC-projection from $G_1$ into $G_2$ (*cf.* Algorithm 1). It begins by the creation of a first rough labeling $\mathcal{I}$ and reduces, for each vertex $x$, the given lists $\mathcal{I}(x)$ to consistent lists using the function `ReviseArc`. The consistency check fails if some $\mathcal{I}(x)$ becomes empty, otherwise the consistency check succeeds and the algorithm gives the labeling $\mathcal{I}$ which is an AC projection $G_1 \rightharpoonup G_2$. 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.

**Algorithm 1:** AC-projection

**Input** : Two graphs $G_1$ and $G_2$
**Output**: An AC-projection $\mathcal{I}$ from $G_1$ into $G_2$ if there is, otherwise an empty set

**foreach** $x \in V(G_1)$ **do**
  $\mathcal{I}$(x)=$\{y \in V(G_2) | l(x) = l(y)\}$
$S \leftarrow A(G_1)$;
$P \leftarrow \emptyset$;
**while** $S \neq \emptyset$ **do**
  Choose an arc $(x, y)$ from S;
  $\mathcal{I}'$:=`ReviseArc` $((x, y), \mathcal{I}, G_2)$;
  **if** $(\mathcal{I}'(x) = \emptyset)$ *or* $(\mathcal{I}'(y) = \emptyset)$ **then**
    **return** $\emptyset$;
  **if** $\mathcal{I}(x) \neq \mathcal{I}'(x)$ **then**
    $R \leftarrow \{(x', y') \in P | x' = x \text{ or } y' = x\}$;
    $S \leftarrow S \cup R$;
    $P \leftarrow P \setminus R$;
  **if** $\mathcal{I}(y) \neq \mathcal{I}'(y)$ **then**
    $R \leftarrow \{(x', y') \in P | x' = y \text{ or } y' = y\}$;
    $S \leftarrow S \cup R$;
    $P \leftarrow P \setminus R$;
  $S \leftarrow S \setminus \{x, y\}$;
  $P \leftarrow P \cup \{x, y\}$;
  $\mathcal{I} \leftarrow \mathcal{I}'$;
**return** $\mathcal{I}$;

**Function** ReviseArc

**Input** : A graph $G_2$, A labeling $\mathcal{I}$ from $G_1$ into $G_2$, An arc $(x, y) \in V(G_1)$
**Output**: A new labeling $\mathcal{I}'$ from $G_1$ into $G_2$

$\mathcal{I}' \leftarrow \mathcal{I}$;
$\mathcal{I}'(x) \leftarrow \mathcal{I}(x) \setminus \{x' \in V(G_2) | \nexists\ y' \in \mathcal{I}(y) \text{ with } (x', y') \in A(G_2)\}$;
$\mathcal{I}'(y) \leftarrow \mathcal{I}(y) \setminus \{y' \in V(G_2) | \nexists\ x' \in \mathcal{I}(x) \text{ with } (x', y') \in A(G_2)\}$;
**return** $\mathcal{I}'$;

### 3.3. *AC-projection And Reduction*

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

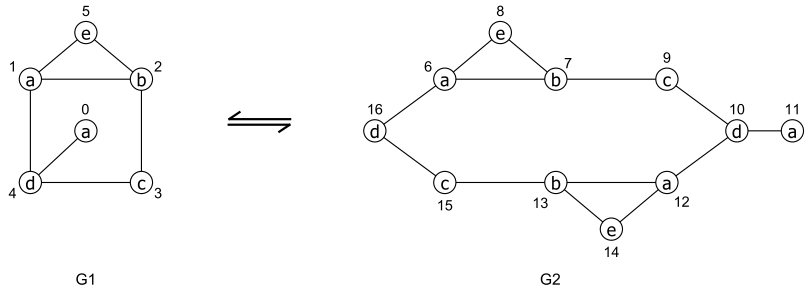


Fig. 3. Two AC-equivalent graphs ($G_1 \leftrightharpoons G_2$)

**Definition 3.6.** (AC-equivalent graphs)
Two graphs $G_1$ and $G_2$ are AC-equivalent iff both $G_1 \rightharpoonup G_2$ and $G_2 \rightharpoonup G_1$ are fulfilled.
We denote it $G_1 \leftrightharpoons G_2$.

For example in Figure 3 we have:

- $G_1 \rightharpoonup G_2$ with the labeling $\mathcal{I} : \mathcal{I}(a0) = \{a6, a11, a12\}$, $\mathcal{I}(a1) = \{a6, a12\}$, $\mathcal{I}(b2)$=$\{b7, b13\}$, $\mathcal{I}(c3) = \{c9, c15\}$, $\mathcal{I}(d4) = \{d16, d10\}$, $\mathcal{I}(e5) = \{e8, e14\}$
- $G_2 \rightharpoonup G_1$ with the labeling $\mathcal{I}' : \mathcal{I}'(a11) = \{a0, a1\}$, $\mathcal{I}'(a6|a12) = \{a1\}$, $\mathcal{I}'(b7|b13) = \{b2\}$, $\mathcal{I}'(c9|c15) = \{c3\}$, $\mathcal{I}'(d16|d10) = \{c4\}$, $\mathcal{I}'(e8|e14) = \{e5\}$

We have an equivalence relation between graphs using the AC-projection. In this paragraph we study the properties of this operation and search for a reduced element in an equivalence class of graphs. This element will be the unique representative of this equivalence class, and for which we give then the name of "AC-reduced graph". The Figure 4 shows a set of graphs belonging to the same equivalence class as well as the AC-reduced one in the extreme right.

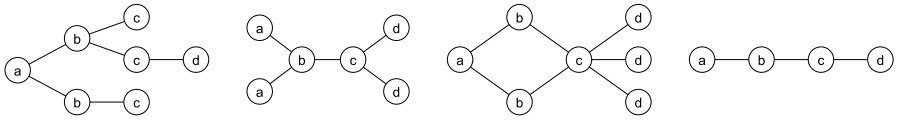

Fig. 4. AC-equivalent graphs and the associated AC-reduced one (extreme right)

3.3.1. *Auto AC-projection And AC-reduction*

We examine the auto AC-projection operation ($G \rightharpoonup G$), which we will use to find the minimal graph of an equivalence class of graphs and we will prove in the following that the obtained graph is minimal.

**Proposition 3.7.** *Given an AC-projection $\mathcal{I} : G \rightarrow G'$, $x' \in \mathcal{I}(x)$ iff for each tree $\mathcal{T}(V_\mathcal{T}, A_\mathcal{T})$, (with $V_\mathcal{T}$ is the set of vertices of $\mathcal{T}$, and $A_\mathcal{T}$ its set of arcs) and each $t \in V_\mathcal{T}$ we have:*

*If there is a morphism from $\mathcal{T}$ to $G$ which associates $t$ to $x$ then there is a morphism from $\mathcal{T}$ to $G'$ which associates $t$ to $x'$.* [6]

**Proposition 3.8.** *(Order relation on $\mathcal{I}$)*

*For an AC-projection $\mathcal{I} : G \rightarrow G$, if $x_i \in \mathcal{I}(x)$ then $\mathcal{I}(x_i) \subseteq \mathcal{I}(x)$*

*Proof.* If we have $x_i \in \mathcal{I}(x)$, it means that for all trees $\mathcal{T}$ having a morphism in $G$ and which associates $t$ to $x$, then there is a morphism from $t$ in $G$ which associates $t$ to $x_i$ (*cf.* Proposition 3.7). We call $\mathcal{T}^{(x,t)}$ this set of trees.

Let us see now if we can have $x_j \in \mathcal{I}(x_i)$ and $x_j \notin \mathcal{I}(x)$.

For $x_j \in \mathcal{I}(x_i)$, according to Proposition 4.3, we see that all trees from $\mathcal{T}^{(x,t)}$ associates to t the vertex $x_i$. Since $x_j \in \mathcal{I}(x_i)$, it will be the same for it, so $x_j \in \mathcal{I}(x)$.

We conclude that we can not have $x_j \notin \mathcal{I}(x)$ since $x_j \in \mathcal{I}(x_i)$, so: $\mathcal{I}(x_i) \subseteq \mathcal{I}(x)$. □

**Proposition 3.9.** *Given a graph $G$ and an AC-projection $\mathcal{I} : G \rightarrow G$ and given a vertex $x \in V(G)$ with $|\mathcal{I}(x)| > 1$.*
*If we have $x_i \in \mathcal{I}(x)$, the graph $G'$ formed by the merging of $x$ and $x_i$ is AC-equivalent to $G$.*

*Proof.* To prove that $G \rightleftharpoons G'$ we have to prove that $G \rightarrow G'$ and $G' \rightarrow G$.
Since $G' \rightarrow G$ by construction we have only to prove that $G \rightarrow G'$:

We construct this AC-projection by replacing $x$ with $x_i$ in the auto AC-projection $G \rightarrow G$. Since $\mathcal{I}(x_i) \subseteq \mathcal{I}(x)$, so there is really an AC-projection. We conclude that $G \rightarrow G'$. □

Now, we are interested in finding the smallest element of the equivalence class of graphs. For two AC-equivalent graphs $G$ and $G'$, we will consider that $G < G'$ iff $|V(G)| < |V(G')|$.

**Proposition 3.10.** *(Minimality)*

*A graph $G$ is minimal in the equivalence class iff for the AC-projection $\mathcal{I} : G \rightarrow G$, $\forall x \in V(G)$, $\mathcal{I}(x) = \{x\}$.*

*Proof.* According to Proposition 4.3, it is clear that if there was a vertex $x$ such that $|\mathcal{I}(x)| > 1$, then we will be able to do another reduction.
Now, the question is: can a graph $G' = G \setminus x$ be AC-equivalent to $G$ ?
If this is true, then we must have an AC-projection from $G$ to $G'$. It would say that $x$ in $G$ has another image $x'$ in $G'$. So, $x'$ must be in $\mathcal{I}(x)$ which is contradictory to the initial hypothesis. □

**Algorithm 2:** AC-reduce

**Input** : A graph $G$
**Output**: $G'$ =AC-reduced G

```
1  G' ← G;
2  I ←AC-projection (G, G);
3  Q ← V(G);
4  Sort Q such as x comes before y if |I(x)| < |I(y)|;
5  foreach v in Q do
6      foreach i in I(v) do
7          if (i ≠ v) then
8              N(v) ← N(v) ∪ N(i); //if v and i are neighbors, then we would
               have a reflexive arc
9              Q ← Q ∖ i;
10             V(G') ← V(G') ∖ i;
11 return G';
```

3.3.2. *AC-reduce Algorithm*

The AC-reduce algorithm (*cf.* Algorithm 2) is based on the properties given on the section above, these properties allow to construct the AC-reduced graph considering any graph $G$. To do this, we simply have to do an auto AC-projection $G \rightharpoonup G$ and then make the necessary merges. So this algorithm is very simple and have a polynomial complexity, since the AC-projection's complexity is polynomial.

## 4. Search space with AC-projection

In this section, we examine the size of the search space using the AC-projection. We present some properties of the AC-projection. While using these properties we can find an upper bound of the search space. We present this result for one labeled graph G and can be easily extended for n graphs (one for each example). In this case, G is the disjoint union of the graphs describing the examples.
Notation: For a labeled graph G:(V,E,L) we note $\mathcal{P}_l$(V), the power set of vertices, in V, with label l ∈ L.

**Definition 4.1.** (AC-graph ) For a labeled graph G:(V,E,L) and a set P of element $\in \bigcup \mathcal{P}_l$(V) with l ∈ L.
We construct a graph G':(V',E',L') with:
- a vertex $v$ for each element in P. We note p($v$) ∈ P the associated element.
- The label of a vertex v in the label of the element in p($v$)
- ($V_1$,$V_2$) ∈ V' iff p($V_1$)$\curvearrowright$p($V_2$)
G' is an AC-graph of G.

So, an AC-graph is built from a list of set of vertices from a graph G.
Now, we look at some links between the AC-graph and the AC-projection.

**Proposition 4.2.** *For each AC-projection between two graphs G', G there is an associated AC-graph.*

*Proof.* Since an AC-projection $\mathcal{I}$, gives, for each vertex of x of G', a set of vertex of G. The AC-graph built from an AC-projection is the one build from the set of I(x), x ∈ V'. □

**Proposition 4.3.** *For each AC-graph G' of a graph G we have G' $\rightharpoonup$ G.*

*Proof.* The labeling $\mathcal{I}$ with, for each V ∈ G', $\mathcal{I}$(V) = p(V) is an AC-labeling from G' into G by construction. □

Now for a graph G we can define a specific AC-graph built from the power set of vertices of G.

**Definition 4.4.** (Max-AC-graph ) For a graph G:(V,E,L) the Max-AC-graph of G is the AC-graph built from the set P of all element ∈ $\bigcup \mathcal{P}_l$(V) with label l ∈ L. We denote this graph Max-AC-graph(G).

All subgraphs of Max-AC-graph(G) have an AC-projection into G. Since the Max-AC-graph is the biggest AC-graph, it will represent the search space. The complexity of the construction of the Max-AC-graph is O($2^{n^2}$) where $n$ is the number of vertices in G. This complexity is high but for many structural descriptions (graph with homorphism projection) the size of the search space is bigger by an order of magnitude.

## 5. $^{\mathcal{B}}$FGMAC: Breadth-first Frequent subGraph Mining with Arc Consistency

In this section, we present $^{\mathcal{B}}$FGMAC, a modified version of the FSG algorithm [10] based on the AC-projection operator. In fact, in this version we have changed 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.

### 5.1. *The Algorithm*

The $^{\mathcal{B}}$FGMAC algorithm 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 (*cf.* Algorithm 3, line 5). Next, it counts the frequency for each of these candidates, and prunes subgraphs that do no satisfy the support constraint (*cf.* Algorithm 3, lines 6-11). Discovered frequent subgraphs

| | | Distinct labels | | Edges / Transaction | | Vertices / Transaction | |
|---|---|---|---|---|---|---|---|
| Datasets | Transactions | Edge | Vertices | Average | Max | Average | Max |
| HIA | 86 | 3 | 8 | 24 | 44 | 22 | 40 |
| PTC-FM | 349 | 3 | 19 | 26 | 108 | 25 | 109 |
| PTC-FR | 351 | 3 | 20 | 27 | 108 | 26 | 109 |
| PTC-MM | 336 | 3 | 21 | 25 | 108 | 25 | 109 |
| PTC-MR | 344 | 3 | 19 | 26 | 108 | 26 | 109 |

fulfill the downward closure property of the support condition, which allows us to effectively prune the lattice of frequent subgraphs.

The $^{\mathcal{B}}$FGMAC's particularity is to output only frequent AC-reduced graphs (Algorithm 3, line 11) which is a subset of the whole frequent isomorphic pattern set.

In the following, we present the key steps of the $^{\mathcal{B}}$FGMAC main process.

#### 5.1.1. *Candidate Generation*

This step is ensured by the `fsg-gen` function (*cf.* Algorithm 3, line 5) used in the FSG algorithm. This function uses a precise joining operator (`fsg-join`), which generates $(k+1) - edges$ subgraphs by joining two frequent $k - edges$ subgraphs. In order to consider two such frequent $k - edges$ subgraph as eligible for joining they must contain the same $(k-1) - edges$ subgraph named core. The complete description of these functions as well as their detailed algorithms are given in [11].

#### 5.1.2. *Support Computation*

The key operator leading this step is the AC-projection previously described. In fact, to check whether a pattern appears in a transaction, $^{\mathcal{B}}$FGMAC computes in polynomial time if there is an AC-projection of the pattern in each one of the transactions. In order to optimize this support computation phase, the algorithm associates to each graph $g$ of size $k$ the set $E(g)$ of transactions such as for each graph $G \in E(g)$, $g \rightharpoonup G$. Having the graph $g_1 \cup g_2$ representing the union of the two graphs $g_1$ and $g_2$, the intersection of the two sets $E(g_1) \cap E(g_2)$ is then calculated.

As $E(g_1 \cup g_2) \subseteq E(g_1) \cap E(g_2)$, it is possible to eliminate the graph $g_1 \cup g_2$ if the transaction's count $E(g_1) \cap E(g_2)$ is sufficiently low to make $g_1 \cup g_2$ infrequent. On the other hand, the existence of a subgraph which has an AC-projection in $g_1 \cup g_2$ can be only looked for in transactions $E(g_1) \cap E(g_2)$.

#### 5.1.3. *Frequents AC-reduction*

This step is essential at the end of each iteration of the main loop of the algorithm. It is intended to avoid the extraction of non AC-reduced frequent graphs, which describe representative elements of graphs equivalence classes w.r.t. AC-equivalence.

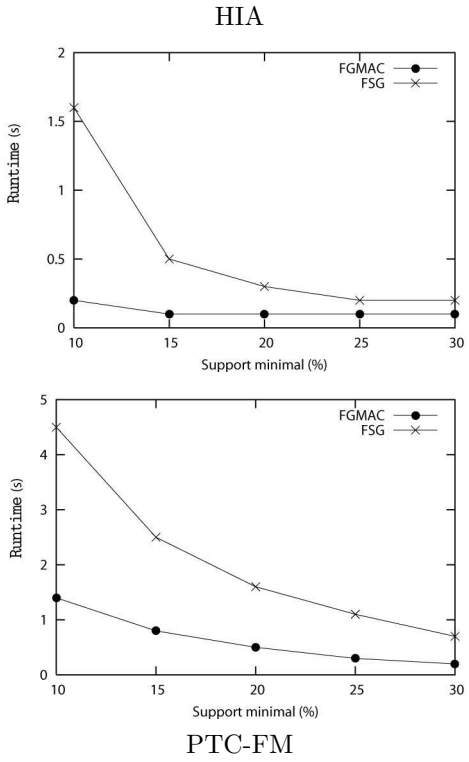


Fig. 5. Runtime comparison of $^{\mathcal{B}}$FGMAC versus FSG with the two datasets HIA and PTC-FM

This process is based on the `AC-reduce` function described previously. We note that this step takes advantage of the polynomial complexity of the AC-reduction algorithm.

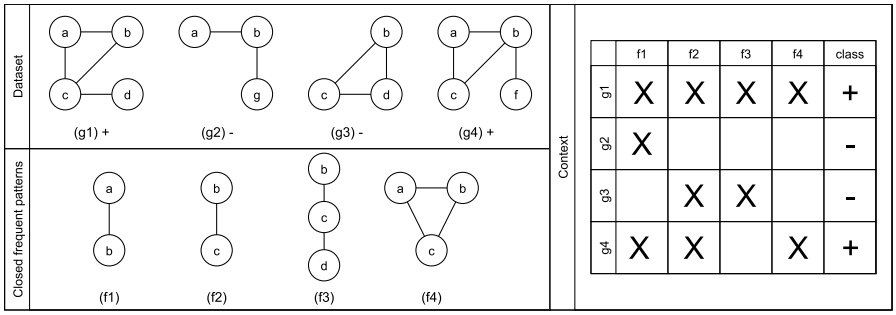


Fig. 6. Graph classification process

**Algorithm 3:** $^{\mathcal{B}}$FGMAC

**Input** : A graph dataset $D$, Minimal support $\sigma$
**Output**: The set $F$ of frequent subgraphs

```
1  F^1 ←detect frequent (1-edge)-subgraphs in D;
2  F^2 ←detect frequent (2-edges)-subgraphs in D;
3  k ← 3;
4  while F^{k-1} ≠ ∅ do
5      C^k ← fsg-gen (F^{k-1})
6      foreach candidate g^k ∈ C^k do
7          g^k.count ← 0;
8          foreach transaction t ∈ D do
9              if g^k ⊸ t then
10                 g^k.count ← g^k.count + 1;
11     F^k ← {AC-reduce(g^k ∈ C^k)|g^k.count ≥ σ|D|};
12     k ← k + 1;
13 return F;
```

## 6. Experiments and Comparative Study

In order to prove the usefulness of the AC-projection for graph mining, we present in the following an experimental study of the $^{\mathcal{B}}$FGMAC algorithm. We insist that the set of frequent AC-reduced graphs found by $^{\mathcal{B}}$FGMAC is not exhaustive w.r.t. isomorphic patterns. In fact, the set of frequent subgraphs found by the means of the AC-projection is a subset of the classical frequent isomorphic subgraphs. In the following, we present a quantitative study of the $^{\mathcal{B}}$FGMAC 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*

We carried out performance and classification experiments on five biological activity datasets widely cited in the literature. These datasets can be divided in two groups:

- The Predictive Toxicology Challenge (PTC) [7] that contains a set of chemical compounds classified according to their toxicity in male rats (PTC-MR), female rats (PTC-FR), male mice (PTC-MM), and female mice (PTC-FM).
- The Human Intestinal Absorption (HIA) [16] that contains chemical compounds classified by intestinal absorption activity.

### 6.2. *Performance Point Of View*

In this subsection we present a quantitative study of the computational performance of $^{\mathcal{B}}$FGMAC compared to FSG. Results depicted in Figure 5 clearly show that $^{\mathcal{B}}$FGMAC outperform FSG in the runtime point of view for all minimal supports selected and confirm the theoretical results about the polynomiality of the AC-projection operator compared to the exponential complexity of the subgraph isomorphism adopted by FSG.

In the following, we present a qualitative-oriented study 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 by 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 represented as a single table, and then traditional classifiers are used for classifying the instances.

The goal of using graph classification in this paper is the evaluation of the quality and discriminative power of frequent AC-reduced subgraph patterns, and to compare it with isomorphic frequent subgraphs.

We carried out classification experiments on five biological activity datasets, and measured classifier prediction accuracy using the known decision trees classifier named C4.5 [15]. The classification methods are described in more detail in the following subsections, along with the associated results.

#### 6.3.1. *Methods*

We evaluated the classification accuracy using two different feature sets. The first set of features (Frequent) consists of all frequent isomorphic subgraphs. Those subgraphs are mined using the Fsg software [10] with different minimal supports. Each chemical compound is represented by a binary vector whose length is 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 second feature set (Closed) is simply a subset of the first set. In fact, it consists of only closed frequent subgraphs. Those subgraphs are also mined using Fsg with the special parameter (-x) to hold only closed frequent subgraphs.

The third feature set (AC-reduced) contains the $^{\mathcal{B}}$FGMAC's output which consists of only AC-reduced frequent subgraphs. We have represented each chemical compound by a binary vector whose length is equal to the number of AC-reduced mined subgraphs. Each AC-reduced subgraph is mapped to a specific vector index,

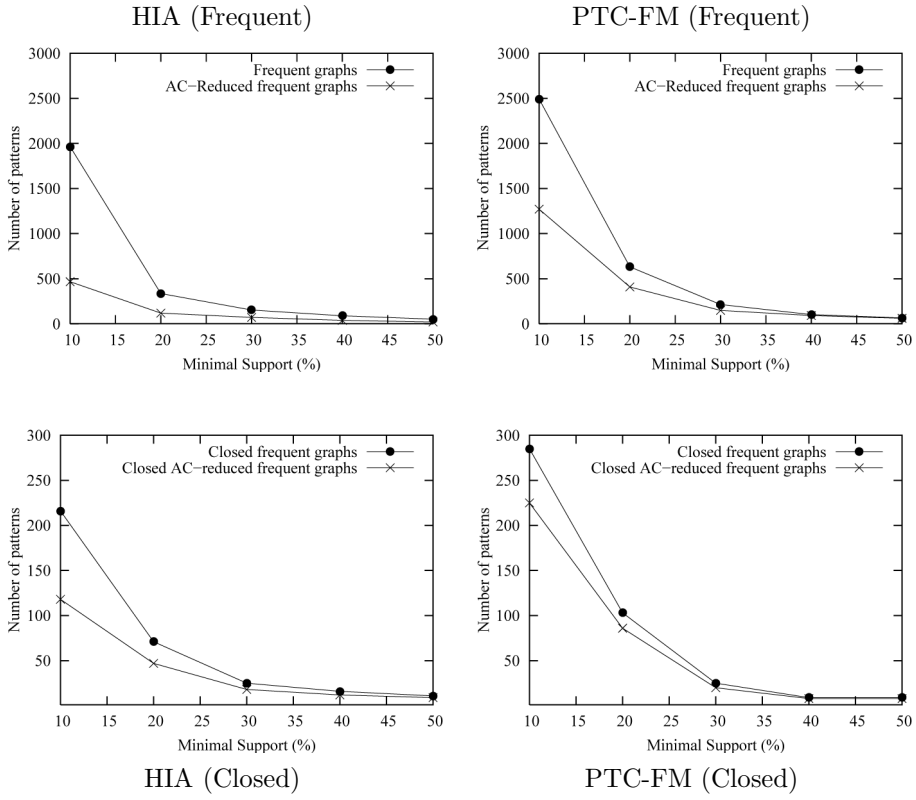


Fig. 7. Comparison of the number of patterns of different feature set for PTC-FM and HIA datasets

and if there is an AC-projection from the AC-reduced subgraph to the chemical compound then the bit at the corresponding index is set to one, otherwise it is set to zero.

Finally, the fourth feature set (Closed AC-reduced) is similar to the third one, the difference is that we only consider closed AC-reduced frequent subgraphs with a special parameter passed to $^{\mathcal{B}}$FGMAC.

The graph classification preprocessing over an example is depicted by Figure 6. The final matrix (the context) will be used by the C4.5[15] decision tree algorithm.

### 6.3.2. *Results*

All classifications have been done with the Weka data-mining software package [17], and we have reported results of the prediction accuracy over the 10 cross-validation trials. In the following, we analyze the AC-reduced patterns from quantitative and qualitative points of view.

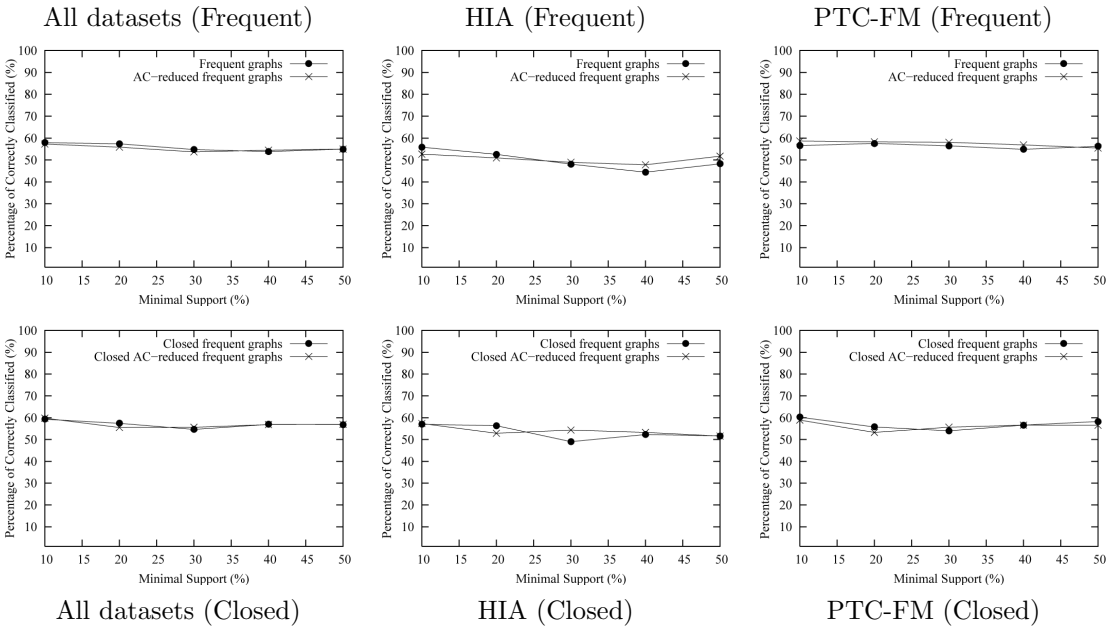


Fig. 8. Comparison of the classification accuracy (PCC) of different feature sets for All datasets(Average), PTC-FM and HIA datasets

**Patterns Count** According to results shown in Figure 7, we see that for all datasets we have very few AC-reduced frequent patterns compared to the isomorphic ones. We have on average 35 patterns. This ratio is bigger for lower supports and can reach up to 70 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 for the polynomiality of the projection operation as well as a smaller search space (*i.e.* fewer AC-projection tests).

**Classification Relevance** When we see that the number of frequent subgraph patterns has drastically decreased after the AC-reduction process, we surely wonder about the relevance of these few patterns for supervised graph classification. That's why we have conducted classification's accuracy experiments using AC-reduced and isomorphic patterns to compare them.

As shown in Figure 8, we see that for the all datasets and all classifiers average, the percentage of correctly classified (PCC) instances is almost the same for all minimal supports, as well as for the other datasets individually.

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

## 7. Conclusions And Future Work

In this paper, we have studied the use of a new polynomial projection operator named AC-Projection initially introduced in [12] and based on a key technique of constraint programming namely Arc Consistency (AC). We have shown that using the AC-projection and its properties has permitted us to have less patterns than all frequent or closed subgraphs but with a very comparable quality and discriminative power. AC-projection is intended to replace the use of the exponential subgraph isomorphism, as well as reducing the search space when seeking for frequent subgraphs. Currently, we are studying a depth-first frequent subgraph mining approach based on the AC-projection operator. Given a graph dataset, this novel approach will be able to look for all frequent AC-reduced patterns with a reduced search space.

## References

1. R. Agrawal and R. Skirant. Fast algorithms for mining association rules. In *proceedings of the* $20^{th}$ *International Conference on Very Large Databases, Santiago, Chile*, pages 478–499, June 1994.
2. C. Bessière and J. Régin. Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems. In E. C. Freuder, editor, *CP*, volume 1118 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1996.
3. D. J. Cook and L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.
4. B. Douar, M. Liquiere, C. Latiri, and Y. Slimani. 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*, pages 112–119, Paris, France, April 2011. IEEE Computer Society.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
6. P. Hell and J. Nesetril. *Graphs and homomorphism*, volume 28. Oxford University Press, Oxford, 2004.
7. C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 20002001. *Bioinformatics*, 17(1):107–108, 2001.
8. J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *International Conference on Data Mining*, page 549. IEEE Computer Society, 2003.
9. A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.
10. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In Nick Cercone, Tsau Young Lin, and Xindong Wu, editors, *International Conference on Data Mining*, pages 313–320. IEEE Computer Society, 2001.
11. M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16:1038–1051, 2004.
12. M. Liquiere. Arc consistency projection: A new generalization relation for graphs. In U. Priss, S. Polovina, and R. Hill, editors, *ICCS*, volume 4604 of *LNCS*, pages 333–346. Springer, 2007.
13. A. K. Mackworth. Consistency in networks of relations. *Artif. Intell.*, 8(1):99–118, 1977.
14. S. Nijssen and J. N. Kok. The gaston tool for frequent subgraph mining. In *Inter-*

*national Workshop on Graph-Based Tools (Grabats). Electronic Notes in Theoretical Computer Science*, pages 77–87, 2004.
15. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1 edition, January 1993.
16. M. D. Wessel, P. C. Jurs, J. W. Tolan, and Steven M. Muskal. Prediction of human intestinal absorption of drug compounds from molecular structure. *Journal of Chemical Information and Computer Sciences*, 38(4):726–735, 1998.
17. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
18. M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. 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*, volume 3721 of *LNCS*, pages 392–403. Springer, 2005.
19. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *International Conference on Data Mining*, pages 721–724. IEEE Computer Society, 2002.