Mining Frequent Closed Graphs on Evolving Data Streams

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer
University of Waikato
Hamilton, New Zealand
{abifet,geoff,bernhard}@cs.waikato.ac.nz

Ricard Gavaldà
LARCA Research Group
UPC-Barcelona Tech
Barcelona, Catalonia
gavalda@lsi.upc.edu

ABSTRACT

Graph mining is a challenging task by itself, and even more so when processing data streams which evolve in real-time. Data stream mining faces hard constraints regarding time and space for processing, and also needs to provide for concept drift detection. In this paper we present a framework for studying graph pattern mining on time-varying streams. Three new methods for mining frequent closed subgraphs are presented. All methods work on coresets of closed subgraphs, compressed representations of graph sets, and maintain these sets in a batch-incremental manner, but use different approaches to address potential concept drift. An evaluation study on datasets comprising up to four million graphs explores the strength and limitations of the proposed methods. To the best of our knowledge this is the first work on mining frequent closed subgraphs in non-stationary data streams.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

Data streams, closed mining, graphs, concept drift

1. INTRODUCTION

Graph mining is a challenging task that extracts novel and useful knowledge from graph data [19, 4]. Due to novel applications in social networks, chemical informatics, bioinformatics, communication networks, computer vision, video indexing and the Web [21], more and more large-scale graphs and sets of graphs are becoming available for analysis. Frequent pattern mining on graphs is one of the ways to obtain useful patterns, e.g. for discriminating graphs in classification and clustering tasks.

Conventional graph mining methods assume that the amount of data is limited and that it is therefore possible to store all data in memory or local secondary storage. There is no limitation on processing time, either. In the *Data Stream* model, we have space and time restrictions. Fundamentally, these restrictions imply the use of incremental techniques. Furthermore, as the data source is not necessarily stationary, methods must be able to adapt to changes over time in the data as well.

As the number of possible subgraphs is exponential, we propose to use *coresets* or compressed representations of the graphs based on closed patterns. Closed patterns are powerful representatives of frequent patterns, since they eliminate redundant information. A frequent pattern is *closed* for a dataset if none of its proper superpatterns has the same support as itself. Other possible definitions of a frequent closed pattern are the following:

- a frequent pattern is closed if it is one of the intersections of all transactions that contain it.
- a frequent pattern is closed if no superpattern is contained in exactly the same transactions as itself.

Generally, there are many fewer frequent closed graphs than frequent ones. In fact, we can obtain all frequent subgraphs with their support from the set of frequent closed subgraphs with their support. So, the set of frequent closed subgraphs maintains the same information as the set of all frequent subgraphs.

There are many methods for computing frequent closed itemsets (see [21]), frequent closed sequences [31, 28], and trees [18, 6, 26, 5], but only two for frequent closed graphs [30, 13], and none for frequent closed graphs on data streams.

We propose the first general methodology to identify closed graphs in a data stream. We develop three closed graph algorithms: IncGraphMiner, an incremental closed graph mining algorithm; WinGraphMiner, a sliding window-based closed graph mining algorithm; and finally AdaGraphMiner, an adaptive closed graph mining algorithm. "Adaptive" means that the model maintains at all times the closed graphs that are frequent in the current state of the data stream, that is, since the latest change in distribution was detected.

The rest of the paper is organised as follows. Sections 2 and 3 give background and present the novel weighted frequent closed pattern setting. Section 4 introduces a new coreset structure and its properties. Section 5 details the handling of concept drift and Section 6 introduces the general mining framework. Experimental results are given in Section 7, and conclusions are drawn in Section 8.

1.1 Related Work

There is a large body of work on itemset mining from data streams; see the survey [23] and the references therein. We can divide these data stream methods into two different classes depending on whether they use a landmark window, containing all the examples seen so far, or a sliding window. Only a small fraction of these methods deal with frequent closed mining. *Moment* [17], *CFI-Stream* [24] and *IncMine* [22] are state-of-the-art algorithms for mining frequent closed itemsets over a sliding window. *CFI-Stream* stores only closed itemsets in memory, but maintains all closed itemsets as it does not apply a minimum support threshold, with the corresponding memory penalty. *Moment* stores much more information besides the current frequent closed itemsets, but it has a minimum support threshold to reduce the number of patterns found. *IncMine* proposes a notion of semi-FCIs that increases the minimum support threshold for an itemset as it is retained longer in the window.

For trees, the work in [8] shows a general methodology to identify closed patterns in a data stream, using Galois Lattice Theory. This approach is based on an efficient representation of trees and a low complexity notion of relaxed closed trees, and presents an online strategy and an adaptive sliding window technique for dealing with changes over time. The approach is different to the one presented in this paper, as it does not use coresets, or weighted frequent mining techniques.

In terms of graphs, two main algorithms exist for mining frequent closed graphs:

- CloseGraph [30]: based on gSpan [29], a miner for finding frequent subgraphs, based on depth-first search (DFS)
- MoSS [13]: an extension to MoFa [11] based on breadth-first search (BFS).

Aggarwal et al. [3] present a mining methodology to find frequent and dense patterns in graph streams. Their notion of density is based both on node-occurrence and edge density, and they present an approach based on finding an approximation of the exact method with the use of a min-hash approach.

To the best of our knowledge the work presented here is the first work dealing with mining frequent closed graphs in streaming data that evolve with time.

2. PRELIMINARIES

We are interested in (possibly infinite) sets of graphs, endowed with a partial order relation $\preceq$ among these graphs. The set of all graphs will be denoted by $\mathcal{G}$, but actually all our developments will proceed in some finite subset of $\mathcal{G}$ which will act as our universe of discourse.

Given two graphs g and g', we say that g is a *subgraph* of g', or g' is a *super-graph* of g, if $g \preceq g'$. Two graphs g, g' are said to be *comparable* if $g \preceq g'$ or $g' \preceq g$. Otherwise, they are incomparable. Also we write $g \prec g'$ if g is a proper subgraph of g' (that is, $g \preceq g'$ and $g \neq g'$).

The input to our data mining process is a dataset $\mathcal{D}$ of weighted transactions, where each transaction $s \in \mathcal{D}$ consists

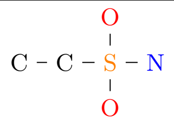
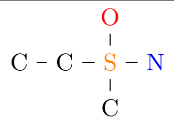
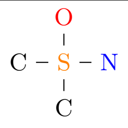
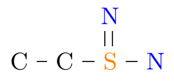
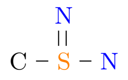
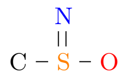
Transaction Id	Graph	Weight
1		1
2		1
3		1
4		1
5		1
6		1

Table 1: Example of graph weighted transaction dataset.

of a transaction identifier *tid*, and a graph. The dataset is a finite set in the standard setting, and a potentially infinite sequence in the data stream setting. Tids are supposed to run sequentially from 1 to the size of $\mathcal{D}$.

Following standard usage, we say that a transaction s *supports* a graph g if g is a subgraph of the graph in transaction s. The number of transactions in the dataset $\mathcal{D}$ that support g is called the *support* of the graph g. A subgraph g is called *frequent* if its support is greater than or equal to a given threshold *min_sup*. The frequent subgraph mining problem is to find all frequent subgraphs in a given dataset. Any subgraph of a frequent graph is also frequent and, therefore, any supergraph of a infrequent graph is also infrequent (the *antimonotonicity* property).

We define a graph g to be *closed* (implicitly, w.r.t. to $\mathcal{D}$) if none of its proper supergraphs has the same support as g has in $\mathcal{D}$.

In this paper we focus our examples and experiments on molecular graphs, but our approach is general enough to be applied to any type of graph.

3. FREQUENT CLOSED WEIGHTED GRAPH MINING

In this section we introduce an extension to the frequent closed graph mining problem which enables the use of compressed graph representations in our adaptive methods: the frequent *weighted* closed graph mining problem. This problem differs from standard frequent closed graph mining in that each input graph has a weight, and this weight is used to compute its weighted support.

The input into the data mining process is a dataset $\mathcal{D}$ of weighted transactions, where each transaction $s \in \mathcal{D}$ consists of a transaction identifier, *tid*, a graph, and a weight value.

Graph	Relative Support	Support
N	-2	6
C – S	-3	6
C – C – S – N	3	3
$\mathrm{C} - \overset{\mathrm{O}}{\overset{\vert}{\mathrm{S}}} - \mathrm{N}$	3	3
$\mathrm{C} - \overset{\mathrm{N}}{\overset{\Vert}{\mathrm{S}}}$	3	3
C – S – O	1	4
C – S – N	-1	5

Table 2: Example of a coreset with minimum support 50% and $\delta = 0$ for the transaction dataset of Table 1.

Table 1 shows an example of a weighted dataset with six molecule graphs [11].

The sum of the weights of transactions in the dataset $\mathcal{D}$ that supports g is called the *weighted support* of the graph g. For brevity, weighted support will be abbreviated to just *support*. A subgraph g is called *frequent* if its weighted support is greater than or equal to a given threshold *min_sup*.

We define a graph g to be *closed* if none of its proper supergraphs has the same weighted support as it has. A graph g is *maximal* if none of its proper supergraphs is frequent. All maximal graphs are closed but not necessarily otherwise. We define a graph g to be *δ-tolerance closed* [16, 25] if none of its proper frequent supergraphs has a weighted support larger than or equal to $(1 - \delta) \cdot support(g)$. Note that a maximal graph is a 1-tolerance closed graph, and a closed graph is a 0-tolerance closed graph. Note that, from [8] we have the following propositions:

Proposition 1. *Adding a transaction with pattern g to a dataset of patterns $\mathcal{D}$ where g is closed does not modify the number of closed patterns for $\mathcal{D}$.*

Proposition 2. *Deleting a pattern transaction that is repeated in a dataset of patterns $\mathcal{D}$ does not modify the number of closed patterns for $\mathcal{D}$.*

Using these propositions we can observe that when adding to or removing from the weights of transactions, we are not modifying the number of closed patterns provided they remain frequent.

4. CORESETS OF CLOSED GRAPHS

A *coreset* [2] of a set P with respect to some problem is a small subset that approximates the original set P, in the sense that solving the problem for the coreset provides an approximate solution for the problem on the original set P. This notion was introduced in computational geometry to denote a small subset of points that can be used to obtain approximate solutions with theoretical guarantees.

For example, for clustering [1], a coreset for a set is a small set, such that for any set of cluster centers the clustering cost of the coreset is an approximation for the clustering cost of the original set with small relative error.

Graph	Relative Support	Support
C – C – S – N	3	3
$\mathrm{C} - \overset{\mathrm{O}}{\overset{\vert}{\mathrm{S}}} - \mathrm{N}$	3	3
$\mathrm{C} - \overset{\mathrm{N}}{\overset{\Vert}{\mathrm{S}}}$	3	3

Table 3: Example of a coreset with minimum support 50% and $\delta = 1$ for the graph dataset of Table 1.

The main advantage of coresets is that we can apply any fast approximation algorithm on the usually much smaller coreset to compute an approximate solution for the original set more efficiently.

Here we use the set of frequent closed subgraphs as a smaller subset of the frequent subgraphs maintaining the same information but using less space. We define the *relative support* of a closed graph as the support of a graph minus the relative support of its closed supergraphs. We define the relative support in this way to exploit the fact that the sum of the closed supergraphs' relative supports of a graph is equal to its own support.

We define a *(s, δ)-coreset* for the problem of computing closed graphs as a weighted multiset of frequent δ-tolerance closed graphs with minimum support s using their relative support as a weight. Table 2 shows a (50%,0)-coreset for the dataset in Table 1 and Table 3 shows a (50%,1)-coreset for the same dataset.

Note that given a graph dataset D, its $(s, 0)$-coreset is the set of frequent closed graphs, and its $(s, 1)$-coreset contains the frequent maximal graphs. As the set of maximal graphs is contained in the set of closed graphs for a minimum support s, the set of graphs in the $(s, 1)$-coreset is also a subset of the graphs in the $(s, 0)$-coreset. For example, the set of graphs in the (50%,1)-coreset of Table 3 is a subset of the graphs in the (50%,0)-coreset of Table 2.

For two minimum supports s_L and s_H where $s_L < s_H$, the set of graphs in its (s_H, δ)-coreset is a subset of the graphs in the (s_L, δ)-coreset.

A coreset is a "compressed representation" of the original dataset. This compression is two-dimensional:

- Minimum support excludes infrequent graphs
- δ-tolerance excludes supergraphs with very similar support

In this paper we focus only on $(s, 0)$-coresets, i.e., frequent closed graphs, as they are the only coresets that can guarantee the recovery of all frequent and frequent closed graphs with minimum support s. To show that this $(s, 0)$-coreset outputs all the frequent subgraphs with minimum support s, we need the following proposition.

Proposition 3. *Let $\mathcal{D}1$ and $\mathcal{D}2$ be two datasets of graphs. Let $C1$ and $C2$ be the $(s, 0)$-coresets of closed graphs for $\mathcal{D}1$ and $\mathcal{D}2$. The set of closed graphs of $C1 \cup C2$ is exactly the set of closed graphs of $\mathcal{D}1 \cup \mathcal{D}2$.*

First, we note that the support of a graph in $C1$ and $C2$ is the same support of a graph in $\mathcal{D}1$ and $\mathcal{D}2$, due to the fact that the support of a graph in $C1$ and $C2$ is the sum of all relative supports. As the support of a closed graph in $\mathcal{D}1$ and $\mathcal{D}2$ is the same support as in $C1$ and $C2$, we can obtain from them the same closed subgraphs, using relative support instead of support.

PROPOSITION 4. *Let $\mathcal{D}1$ and $\mathcal{D}2$ be two datasets of graphs. Let $C1$ and $C2$ be the $(s, 0)$-coresets of closed graphs for $\mathcal{D}1$ and $\mathcal{D}2$. The set of frequent graphs of $C1 \cup C2$ is exactly the set of frequent graphs of $\mathcal{D}1 \cup \mathcal{D}2$.*

To show that we can also obtain the frequent ones, we see that the frequent non-closed graphs have the support of their closure, so we obtain the same frequent graphs.

PROPOSITION 5. *Let $\mathcal{D}1$, $\mathcal{D}2$ and $\mathcal{D}$ be three datasets of graphs with $\mathcal{D} = \mathcal{D}1 \cup \mathcal{D}2$. Let $C1$, $C2$ and C be the $(s, 0)$-coresets of closed graphs for $\mathcal{D}1$,$\mathcal{D}2$ and $\mathcal{D}$. The set of frequent graphs for $C1 = C \backslash C2 = C \cup \overline{C2}$ obtained using relative support for $C1$ and negative relative support for $C2$, is exactly the set of frequent graphs for $\mathcal{D}1 = \mathcal{D} \backslash \mathcal{D}2$.*

As the support of a graph in $C1$ is the support of the graph in C minus its support in $C2$, we can obtain the closed graphs in $C1$ by adding the closed graphs in $C2$, with negated relative support, to C.

In general terms, using $(s, 0)$-coresets we can obtain all frequent, closed and maximal graphs with minimum support s without losing any information. Using (s, δ)-coresets we get more compact representations of the original dataset, and we can still obtain the maximal graphs, but we cannot completely recover all frequent closed graphs of minimum support s.

5. ESTIMATING FREQUENCIES ADAPTIVELY

To deal with concept drift, our methods have to be able to adapt to changes in the input distribution. Keeping a sliding window of recent elements is an option, however, it has the cost of maintaining it in memory. In this paper, we propose to use `ADWIN` to estimate frequencies of graphs with theoretical guarantees.

`ADWIN` (ADaptive sliding WINdow) [7] is a change detector and estimation algorithm. It solves, in a well-specified way, the problem of tracking the average of a stream of bits or real-valued numbers. `ADWIN` keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window".

More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: one, change is reliably detected whenever the window shrinks; and two, at any time the average over the existing window can be used as a reliable estimate of the current average in the stream (barring a very small or recent change that is not yet statistically significant).

The inputs to `ADWIN` are a confidence value $\delta \in (0, 1)$ and a (possibly infinite) sequence of real values x_1, x_2, x_3, ..., x_t, ... The value of x_t is available only at time t. Each x_t is generated according to some distribution D_t, independently for every t. We denote with μ_t the expected value of x_t when it is drawn according to D_t. We assume that x_t is always in $[0, 1]$; rescaling deals with cases where $a \leq x_t \leq b$. No further assumption is being made about the distribution D_t; in particular, μ_t is unknown for all t.

`ADWIN` is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, inherent to all algorithms dealing with random processes.

It is important to note that `ADWIN` does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique [20]. This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item, rather than the $O(W)$ one expects from a naïve implementation.

The main technical result in [7] about the performance of `ADWIN` is the following theorem, that provides bounds on the rate of false positives and false negatives:

THEOREM 1. *With ϵ_{cut} defined as*

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4|W|}{\delta}}$$

where for some *partition of W in two parts $W_0 W_1$ (where W_1 contains the most recent items), and m is the harmonic mean of $|W_0|$ and $|W_1|$, at every time step we have:*

1. (False positive rate bound). *If μ_t has remained constant within W, the probability that* `ADWIN` *shrinks the window at this step is at most δ.*

2. (False negative rate bound). *Suppose that for* some *partition of W in two parts $W_0 W_1$ we have $|\mu_{W_0} - \mu_{W_1}| > 2\epsilon_{cut}$. Then with probability $1 - \delta$* `ADWIN` *shrinks W to W_1, or shorter.*

6. FREQUENT GRAPH MINING ON DATA STREAMS

In this section we present new graph mining methods for data streams using the results discussed in previous sections. We start by reviewing non-incremental basic methods, and then present the incremental miner INCGRAPHMINER, the sliding-window based miner WINGRAPHMINER, and the adaptive miner ADAGRAPHMINER.

6.1 Non-Incremental Graph Mining

[29, 30] presents two algorithms for computing frequent and closed graphs from a dataset of graphs, in a non incremental way. They represent the potential subgraphs to be checked for, being frequent and closed on the dataset, in such a way that extending them by one single node, in all possible ways, corresponds to a clear and simple operation on the representation. The completeness of the procedure is assured, that is, all graphs can be obtained in this way. This allows them to avoid extending graphs that have already been found to be infrequent.

The pseudocode of CLOSEGRAPH is presented in Figure 1. Note that the first line of the algorithm is a canonical representative check. Such checks are used frequently in tree and graph mining. The use of the right-most extension approach in CLOSEGRAPH, based on depth-first search, guarantees that all possible graphs are reached, but reduces the generation of duplicate graphs. CLOSEGRAPH selects the minimum DFS code based on a DFS lexicographical order as the canonical representative.

In MoSS [12], Borgelt et al. present a different method to perform frequent closed graph mining using breadth-first search instead of using the right-most extension approach.

```
CloseGraph($g, D, min\_sup, S$)
    Input: A graph $g$, a graph dataset $D$, $min\_sup$.
    Output: The frequent graph set $S$.

 1  if $g \neq$ Canonical_Representative($g$)
 2     then return $S$
 3  $isClosed \leftarrow$ TRUE
 4  $C \leftarrow \emptyset$
 5  for each $g'$ that is a one step right-most
        extension of $g$
 6        do if support($g'$) $\geq min\_sup$
 7              then insert $g'$ into $C$
 8           if support($g'$) $=$ support($g$)
 9              then $isClosed \leftarrow$ FALSE
10  if $isClosed =$ TRUE
11     then insert $g$ into $S$
12  for each $g'$ in $C$
13        do $S \leftarrow$ CloseGraph($g', D, min\_sup, S$)
14  return $S$
```

Figure 1: The CloseGraph algorithm

```
IncGraphMiner($D, min\_sup$)
    Input: A graph dataset $D$, and $min\_sup$.
    Output: The frequent graph set $G$.

1  $G \leftarrow \emptyset$
2  for every batch $b_t$ of graphs in $D$
3        do C $\leftarrow$ Coreset($b_t, min\_sup$)
4           $G \leftarrow$ Coreset($G \cup C, min\_sup$)
5  return $G$

Coreset($b_t, min\_sup$)
    Input: A graph dataset $b_t$, and $min\_sup$.
    Output: The coreset $C$.

1  $C \leftarrow$ CloseGraph($b_t, min\_sup$)
2  $C \leftarrow$ Compute_Relative_Support($C$)
3  return $C$
```

Figure 2: The IncGraphMiner algorithm

6.2 Incremental Mining

The incremental setting is different. Suppose that the data arrives in batches of graphs. We consider every batch of graphs b_t as a small finite dataset $\mathcal{D}_b$ of transactions, where each transaction $s \in \mathcal{D}_b$ consists of a transaction identifier, *tid*, a graph and a weight value. We maintain a set of graphs G and we update this set every time a new batch of graphs $\mathcal{D}_b$ arrives. We compute the coreset of the batch b_t, and then we use it to update the graph set G.

Note that using this methodology we can transform non-incremental methods into incremental ones by extending them to use weights and coresets of graphs. Figure 2 shows the pseudocode of IncGraphMiner.

```
WinGraphMiner($D, W, min\_sup$)
    Input: A graph dataset $D$, a size window $W$ and $min\_sup$.
    Output: The frequent graph set $G$.

1  $G \leftarrow \emptyset$
2  for every batch $b_t$ of graphs in $D$
3        do C $\leftarrow$ Coreset($b_t, min\_sup$)
4           Store $C$ in sliding window
5           if sliding window is full
6              then $\overline{R} \leftarrow$ Oldest $C$ stored in sliding window,
                       negate all support values
7              else $\overline{R} \leftarrow \emptyset$
8           $G \leftarrow$ Coreset($G \cup C \cup \overline{R}, min\_sup$)
9  return $G$
```

Figure 3: The WinGraphMiner algorithm

6.3 Erroneous omission or inclusion bounds

When working on batches of graphs at a time, there is a possibility that the current batch does not contain enough occurrences of a frequent pattern, and it might also contain more than *min_sup* occurrences of an otherwise infrequent pattern. The probabilities for these two types of errors, erroneous omission or inclusion, can be bounded in the following way. Given a fixed batch size n, every *min_sup* value is equivalent to a probability p of finding a specific pattern in any given graph. Then, if the global true probability for some pattern is $p + \Delta$, and if we approximate the binomial distribution with the normal distribution $N(n*(p+\Delta), n*(p+\Delta)*(1-p-\Delta)$, the following bound can be derived:

$$p_{discard}(\Delta, n) \quad \leq \quad \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{-2\Delta\sqrt{n}} e^{-\frac{t^2}{2}} \, \mathrm{d}t \qquad (1)$$

If $\Delta = 0$, then half of the patterns will be missed, but reasonably small values for Δ will already result in acceptable probabilities: if $n = 10000$, then $\Delta = 0.01$ results in less than 5% erroneous omissions or inclusions, and for $\Delta = 0.02$ this percentage is already very close to zero.

This type of argument is a heuristic that can be made formal (at the expense of worse figures) by the use of rigorous Hoeffding bounds, which underlie e.g. the `ADWIN` algorithm.

6.4 Mining with a Sliding Window

We present WinGraphMiner as a learner that maintains a sliding window. Its main important parameter is the size of the window W. The difference to IncGraphMiner lies in the management of the items in the sliding window. We update the recent frequent closed graphs, using the coresets of the new batches that arrive. When the window is full, we delete the oldest batch on the sliding window using negative relative support (Proposition 5).

Figure 3 shows the pseudocode of WinGraphMiner. When a new batch b_t arrives, the method performs as IncGraph-Miner, while the sliding window is not full. Once the window is full, the oldest coreset C is removed. The new coreset for G is computed in one step from the coresets of G and b_t, and C with negated relative support.

```
AdaGraphMiner($D, Mode, min\_sup$)
      Input: A graph dataset $D$, mode $Mode$ and $min\_sup$.
      Output: The frequent graph set $G$.

 1  $G \leftarrow \emptyset$
 2  Init ADWIN
 3  for every batch $b_t$ of graphs in $D$
 4        do $C \leftarrow$ Coreset($b_t, min\_sup$)
 5           $\overline{R} \leftarrow \emptyset$
 6           if Mode is Sliding Window
 7              then Store $C$ in sliding window
 8                   if ADWIN detected change
 9                      then $\overline{R} \leftarrow$ Batches to remove
                             in sliding window
                             with negative support
10           $G \leftarrow$ Coreset($G \cup C \cup \overline{R}, min\_sup$)
11           if Mode is Sliding Window
12              then Insert # closed graphs into ADWIN
13              else for every $g$ in $G$ update $g$'s ADWIN
14  return $G$
```

Figure 4: The AdaGraphMiner algorithm

6.5 Adaptive Mining

Finally, we present AdaGraphMiner, an extension to the previous methods that is able to adapt to changes on the stream, maintaining only the *currently* frequent closed graphs.

We present two versions of this method. One uses an adaptive sliding window to maintain the batches of graphs. The other uses a separate ADWIN instance for managing the support of every single frequent closed subgraph.

Figure 4 shows the pseudocode of AdaGraphMiner, where the *Sliding Window* test distinguishes between the two methods.

Assume a scenario where at each time t the t-th element of the stream is a graph taken independently from a distribution D_t over a fixed universe $\mathcal{G}$ of graphs. Then, changes in the D_t's over time t are modelling the evolution in the stream. In particular, we say that "the stream is stationary between times $t1$ and $t2$" (for $t1 < t2$) if

$$D_{t1} = D_{t1+1} = \cdots = D_{t2-1} = \cdots = D_{t2}.$$

For a graph g, let $f_t(g)$ be the probability of g under distribution D_t, and $\hat{f}_t(g)$ the observed probability of g at time t (the number of times g has appeared in the window at time t divided by the length of the window). Since $\hat{f}_t(g)$ is our estimate of the current support of t, we would like it to be close enough to $f_t(g)$ to reliably decide whether g is frequent.

We can prove the following theorem:

Theorem 2. *Suppose the* ADWIN*'s in Algorithm* AdaGraphMiner *are initialized with parameter δ. For every time step $t1$, $t2$ such that $t1 < t2$, if the stream is stationary between $t1$ and $t2$, then for every $g \in \mathcal{G}$ we have with probability at least $1 - 2\delta$*

$$|f_{t2}(g) - \hat{f}_{t2}(g)| \leq \sqrt{\frac{1}{t2 - t1} \ln \frac{4W}{\delta}}$$

That is, if the stream remains stationary for T steps, the estimates of graph frequencies tend to the their value at a rate of roughly $O(1/\sqrt{T})$, which tends to 0 as $T \to \infty$. Observe that this is the best convergence rate obtainable by sampling even on a totally stationary stream. That is, the algorithm is able to effectively forget previous history, and converge at an optimal rate on stationary periods. Note that, if instead of ADWIN, we keep a simple sliding window of fixed size W the approximation will not tend to zero, but remain at about $1/\sqrt{W}$. The dependence on $\ln(W)$ in the theorem is to a large extent an artifact of the proof of ADWIN's guarantees, and in practice, slightly increasing the leading constant in the bound suffices for accurate results.

Proof. (Note: this proof uses both the definition of the ADWIN algorithm as well as Theorem 1 stating false positive and false negative ratios; see [7]).

Fix $t1 < t2$. We will show the following statement: For every fixed graph $g \in \mathcal{G}$, with probability at least $1 - 2\delta$ we have

$$|f_{t2}(g) - \hat{f}_{t2}(g)| \leq \sqrt{\frac{1}{t2 - t1} \ln \frac{4W}{\delta}}. \qquad (2)$$

Now also fix a graph $g \in \mathcal{G}$. If g is not present in the set G kept by the algorithm, then $\hat{f}_{t2}(g) = 0$ and an application of the Hoeffding bound shows that (2) holds for g. Otherwise, g is in the set G, so it has an associated instance of ADWIN, A. Say that, at time $t2$, A keeps a window of size W. Because the stream is stationary between $t1$ and $t2$, by Theorem 1, part (1), we know that $W \geq t2 - t1$ (i.e., the element gathered at $t1$ has not been cut out from A's window) with probability at least $1 - \delta$.

Let $t0$ be the oldest element present in the window, so that $t0 = t2 - W \leq t1$. Now let $\hat{\mu}_0$ (resp., $\hat{\mu}_1$) be the observed frequence of g in the window $[t0..t1-1]$ (resp., in the window $[t1..t2]$); observe that $E[\mu_1] = f_{t2}(g)$. Observe also that by the definition of ADWIN, if the interval $[t0..t1-1]$ has *not* been cut out from the window, we must have $|\hat{\mu}_0 - \hat{\mu}_1| \leq \epsilon_{cut}$, where

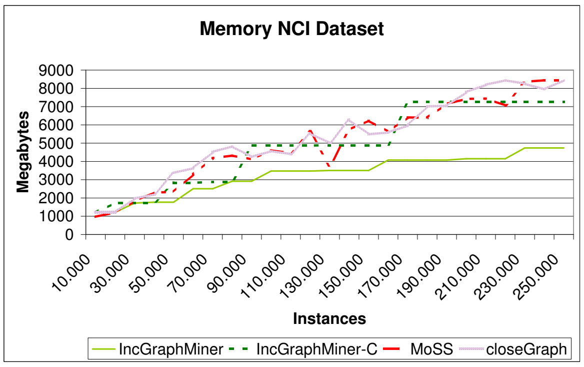
$$\epsilon_{cut} = \sqrt{\frac{1}{2}\left(\frac{1}{t2 - t1} + \frac{1}{t1 - t0}\right) \ln \frac{4W}{\delta}}.$$

The estimate produced by A at time $t2$ is the average of the elements in the window of length W, which is

$$\begin{aligned}
\hat{f}_{t2}(g) &= \hat{\mu}_1 \cdot \frac{t2 - t1}{W} + \hat{\mu}_0 \cdot \frac{t1 - t0}{W} \\
&= \hat{\mu}_1 \cdot \frac{t2 - t1}{W} + (\hat{\mu}_1 \pm \epsilon_{cut}) \cdot \frac{t1 - t0}{W} \\
&= \hat{\mu}_1 \pm \epsilon_{cut} \cdot \frac{t1 - t0}{W}.
\end{aligned}$$

Now using $W = t2 - t0$ and some simple algebra one can see

$$\begin{aligned}
\epsilon_{cut} \cdot \frac{t1 - t0}{W} &\leq \sqrt{\frac{t1 - t0}{2(t2 - t1)(t2 - t0)} \ln \frac{4W}{\delta}} \\
&\leq \sqrt{\frac{1}{2(t2 - t1)} \ln \frac{4W}{\delta}}.
\end{aligned}$$



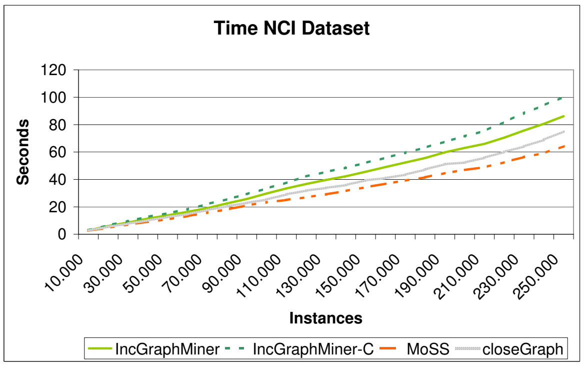
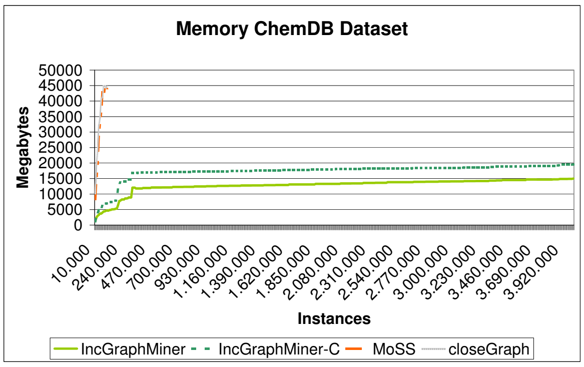


Figure 5: Memory and time used on Open NCI database with minimal support of 40%.



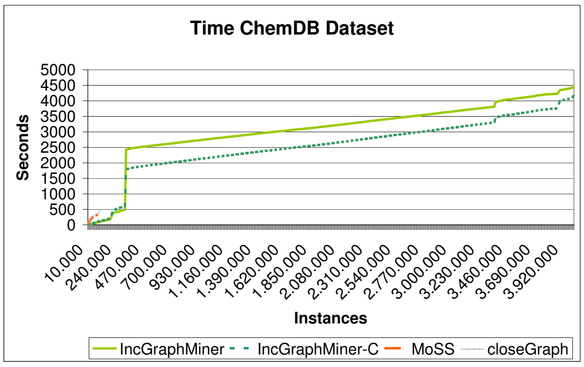


Figure 6: Memory and time used on ChemDB dataset with minimal support of 40%.

and by the Hoeffding bound and $E[\mu_1] = f_{t2}(g)$ we have

$$\Pr\left[|\hat{\mu}_1 - f_{t2}(g)| \le \sqrt{\frac{1}{2(t2-t1)} \ln \frac{2}{\delta}}\right] \le \delta.$$

Putting all of this together we conclude that, with probability at least $1 - 2\delta$,

$$\begin{aligned} |f_{t2}(g) - \hat{f}_{t2}(g)| &\le \sqrt{\frac{1}{2(t2-t1)} \ln \frac{4W}{\delta}} + \\ & \sqrt{\frac{1}{2(t2-t1)} \ln \frac{2}{\delta}} \\ &\le \sqrt{\frac{1}{t2-t1} \ln \frac{4W}{\delta}}. \end{aligned}$$

as was to be proved. $\square$

7. EXPERIMENTAL EVALUATION

We tested our algorithms on synthetic and real datasets. First, we compare our new incremental strategy with *closeGraph* and *MoSS* as non incremental methods to show the benefits in memory and time. Second, we perform experiments on data streams with synthetic concept drift to show the performance of our adaptive strategy.

We run our experiments extending MOA [10] using MoSS [12]. All experiments were performed on a 2.66 GHz Core 2 Duo machine with 64 GB of memory main memory, running CentOS 5.5.

Massive **O**nline **A**nalysis (MOA) [10] is a framework for online learning from continuous supplies of examples, such as data streams. It comprises online classification and clustering methods as well as tools for both offline and online evaluation.

MoSS is a batch framework for finding frequent molecular substructures and discriminative fragments in a database of molecule descriptions. MoSS is not restricted to molecular data sets, it can mine arbitrary data sets of attributed graphs. Apart from the default MoSS/MoFa algorithm, this framework also contains the gSpan [29] and CloseGraph [30] algorithms as special processing modes.

For our experiments we use the following real datasets:

ChemDB dataset ChemDB [14, 15] is a public dataset of approximately 4 million molecules built using the digital catalogs of over a hundred vendors and other public

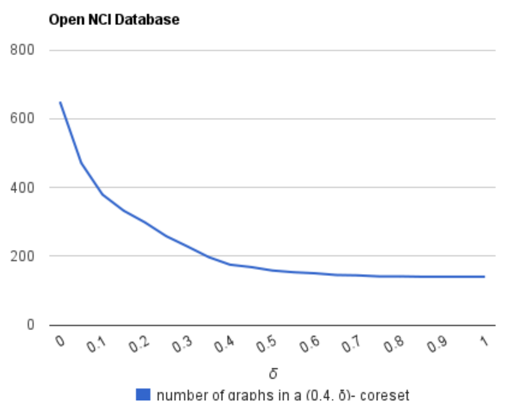


Figure 7: Number of graphs in $(40\%, \delta)$ for NCI.

sources. It is annotated with information derived from these sources as well as from computational methods, such as predicted solubility and 3D structure. It supports multiple molecular formats and is periodically updated, automatically whenever possible. It is maintained by the Institute for Genomics and Bioinformatics at the University of California, Irvine.

Open NCI Database The open National Cancer Institute (NCI) dataset [27] consists of approximately 250,000 structures. It is based on a large NCI database, built using samples from organic synthesis submitted to NCI for testing. While about half of the NCI database is not accessible, the other half of the structural data is free of any disclosure and usage restrictions and therefore termed "open". This data is public domain and often referred to as the "Open NCI Database".

First, we compare the runtime and memory usage of the incremental method IncGraphMiner to the non-incremental ones. As all methods are implemented in Java, the memory shown is the memory allocated by the Java Virtual Machine.

The results of our experiments using batches of $10,000$ instances, minimum support of 40%, $\delta = 0$, and 50 Gigabytes of maximum memory are shown in Figures 5 and 6. We test two incremental methods, IncGraphMiner based on MoSS, and IncGraphMiner-C based on closeGraph as batch learners. For the Open NCI database, which is the smaller dataset, we see that our new incremental methods need more time but less memory. However, for the larger ChemDB dataset, the non-incremental methods can only process about 110,000 instances before running out of memory (50 Gigabytes). Surprisingly, the incremental methods also outperform the non-incremental ones both in terms of runtime and memory usage for smaller subsets of ChemDB. Note that in Figure 6 there is a sudden bump in time used, since the graphs in this part of the stream are much larger.

Figure 7 shows the size of the coreset $(40\%, \delta)$ on the Open NCI database. For $\delta = 0$ there are 649 frequent closed graphs, and for $\delta = 1$ there are 140 frequent maximal graphs.

The next experiment shows how the new methods are able to adapt to changes in the distribution of graphs in an evolving stream scenario. Figure 8 shows the number of closed graphs detected using the incremental method IncGraphMiner, and the two adaptive methods. AdaGraphMiner-Window uses a sliding window monitored only by one `ADWIN` instance, and AdaGraphMiner uses one `ADWIN` instance for each graph to monitor its support. In this experiment, the data stream exhibits artificial concept drift, as it is a concatenation of parts of two different streams. One is the Open NCI database and the other is an artificial data stream with exactly 15 frequent closed graphs generated artificially. Concept drift occurs three times, after 250,000, 500,000, and 750,000 examples, respectively. We follow the methodology in [9] to combine two data streams into one in order to create artificial drift. We observe that the incremental method is the slowest to adapt, as it does not have any forgetting mechanism. Comparing the adaptive ones, we see that AdaGraphMiner-Window, which only maintains one global `ADWIN` instance for change detection, shows a slower response to concept change. AdaGraphMiner, on the hand, can detect and respond to change faster, as it maintains a separate `ADWIN` instance for each graph.

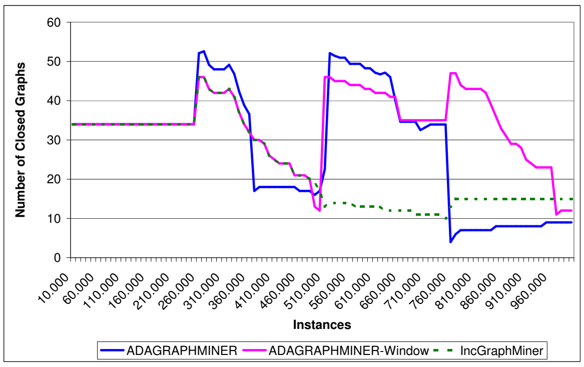


Figure 8: Number of frequent closed graphs on Open NCI database with minimal support of 40% with artificial concept drift.

8. CONCLUSIONS

We have presented the first efficient algorithms for mining frequent closed graphs on evolving data streams.

If the distribution of the graph dataset is stationary, the best method to use is IncGraphMiner, as no past transactions need to be forgotten. If the distribution evolves, then a sliding window method is more appropriate. If the right size of the sliding window is known in advance, then WinGraphMineris the method of choice, otherwise AdaGraphMiner is the preferred option.

Future work will analyze the impact of the size of batches and apply these new adaptive frequent mining techniques to discriminative graph feature generation for classification and clustering. Another ambitious direction for future work would be going beyond batch-incremental methods of graph mining, to develop methods that are fully instance-incremental. It is currently unclear how this could be done efficiently.

9. ACKNOWLEDGMENTS

UPC research is partially supported by the Spanish Ministry of Science and Technology TIN-2008-06582-C03-01 (SESAAME), by the Generalitat de Catalunya 2009-SGR-1428 (LARCA), and by the EU PASCAL2 Network of Excellence (FP7-ICT-216886).

The authors would like to thank Christian Borgelt for making his MoSS graph software available and for helpful discussions.

10. REFERENCES

[1] M. R. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot. StreamKM++: A Clustering Algorithm for Data Streams. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX '10)*, 2010.

[2] P. K. Agarwal, S. Har-Peled, and K. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and computational geometry*, pages 1–30. Cambridge University Press., 2005.

[3] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin. On dense pattern mining in graph streams. *PVLDB*, 3(1):975–984, 2010.

[4] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[5] H. Arimura and T. Uno. An output-polynomial time algorithm for mining frequent closed attribute trees. In *ILP*, pages 1–19, 2005.

[6] J. L. Balcázar, A. Bifet, and A. Lozano. Mining frequent closed rooted trees. *Machine Learning*, 78(1-2):1–33, 2010.

[7] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.

[8] A. Bifet and R. Gavaldà. Mining adaptively frequent closed unlabeled rooted trees in data streams. In *14th ACM SIGKDD*, pages 34–42, 2008.

[9] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *15th ACM SIGKDD*, pages 139–148, 2009.

[10] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering. *Journal of Machine Learning Research - Proceedings Track*, 11:44–50, 2010.

[11] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In ICDM '02, pages 51–, Washington, DC, USA, 2002. IEEE Computer Society.

[12] C. Borgelt, T. Meinl, and M. Berthold. Moss: a program for molecular substructure mining. In OSDM '05, pages 6–15, New York, NY, USA, 2005. ACM.

[13] C. Borgelt, T. Meinl, and M. R. Berthold. Advanced pruning strategies to speed up mining closed molecular fragments. In *IEEE Conf. on Systems, Man and Cybernetics (SMC 2004)*. IEEE Press, 2004.

[14] J. Chen, S. J. J. Swamidass, Y. Dou, J. Bruand, and P. Baldi. ChemDB: a public database of small molecules and related chemoinformatics resources. *Bioinformatics*, September 2005.

[15] J. H. Chen, E. Linstead, S. J. Swamidass, D. Wang, and P. Baldi. ChemDB update full-text search and virtual chemical space. *Bioinformatics*, 23(17):2348–2351, September 2007.

[16] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD Conference*, pages 857–872, 2007.

[17] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM'04)*, November 2004.

[18] Y. Chi, Y. Xia, Y. Yang, and R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *Fundamenta Informaticae*, XXI:1001–1038, 2001.

[19] D. J. Cook and L. B. Holder. *Mining Graph Data*. Wiley-Interscience, 2006.

[20] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 14(1):27–45, 2002.

[21] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15:55–86, 2007.

[22] Y. K. James Cheng and W. Ng. Maintaining frequent closed itemsets over a sliding window. *Journal of Intelligent Information Systems*, 2007.

[23] Y. K. James Cheng and W. Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems*, 2007.

[24] N. Jiang and L. Gruenwald. Cfi-stream: mining closed frequent itemsets in data streams. In *KDD '06*, pages 592–597, New York, NY, USA, 2006. ACM.

[25] I. Takigawa and H. Mamitsuka. Efficiently mining δ-tolerance closed frequent subgraphs. *Machine Learning*, 82(2):95–121, 2011.

[26] A. Termier, M.-C. Rousset, M. Sebag, K. Ohara, T. Washio, and H. Motoda. DryadeParent, an efficient and robust closed attribute tree mining algorithm. *IEEE Trans. Knowl. Data Eng.*, 20(3):300–320, 2008.

[27] J. H. Voigt, B. Bienfait, S. Wang, and M. C. Nicklaus. Comparison of the nci open database with seven large chemical structural databases. *Journal of Chemical Information and Computer Sciences*, 41(3):702–712, 2001.

[28] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *ICDE*, pages 79–90. IEEE Computer Society, 2004.

[29] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM '02*, page 721, Washington, DC, USA, 2002. IEEE Computer Society.

[30] X. Yan and J. Han. CloseGraph: mining closed frequent graph patterns. In *KDD '03*, pages 286–295, 2003.

[31] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In *SDM*, 2003.