Efficient and Accurate Non-exhaustive Pattern-Based Change Detection in Dynamic Networks

Angelo Impedovo[1(✉)], Michelangelo Ceci[1], and Toon Calders[2]

[1] Department of Computer Science, University of Bari "Aldo Moro", Bari, Italy
{angelo.impedovo,michelangelo.ceci}@uniba.it
[2] Department of Computer Science, University of Antwerp, Antwerp, Belgium
toon.calders@uantwerpen.be

Abstract. Pattern-based change detectors (PBCDs) are non-parametric unsupervised change detection methods that are based on observed changes in sets of frequent patterns over time. In this paper we study PBCDs for dynamic networks; that is, graphs that change over time, represented as a stream of snapshots. Accurate PBCDs rely on exhaustively mining sets of patterns on which a change detection step is performed. Exhaustive mining, however, has worst case exponential time complexity, rendering this class of algorithms inefficient in practice. Therefore, in this paper we propose non-exhaustive PBCDs for dynamic networks. The algorithm we propose prunes the search space following a beam-search approach. The results obtained on real-world and synthetic dynamic networks, show that this approach is surprisingly effective in both increasing the efficiency of the mining step as in achieving higher detection accuracy, compared with state-of-the-art approaches.

Keywords: Change detection · Pattern mining

1 Introduction

Change detection in dynamic networks is the task of finding time points in which the behavior of the observed network begins to change from the ordinary situation. Once identified, the points provide temporal indications on the obsolescence of previously trained models, which should be adapted to new data. The problem, also known as concept drift detection [3], affects both supervised and unsupervised techniques and therefore is one of the most important problems in the analysis of data that are characterized by a temporal component. For example, in the supervised setting the detection is performed by controlling a quality measure of a previously learned model on new data (e.g. using the misclassification rate [3]). Whereas, in the unsupervised setting, it takes into account how new data deviate from the ordinary data distribution [6]. In both cases, if a significant peak on some appropriate measures is observed, then a change is detected and some actions are performed to update the model by considering new data.

P. Kralj Novak et al. (Eds.): DS 2019, LNAI 11828, pp. 396–411, 2019.
https://doi.org/10.1007/978-3-030-33778-0_30

One of the main challenges in change detection is that of detecting changes in an efficient and accurate manner. In the specific case of dynamic networks, existing approaches may fail to address both challenges because of the inherent complexity of the data [1]. In fact, (i) many approaches are designed for time series and categorical data and not for network-based data, and (ii) many network-based approaches are time consuming, and hence not scalable in both network size and number of graph snapshots. Moreover, the lack of a ground truth able to establish which data represents a change favors the adoption of unsupervised change detection methods.

Recently, non-parametric unsupervised change detection methods relying on frequent patterns have been proposed for transactional data [6,8] and dynamic networks [9]. Such methods are *pattern-based change detectors* (PBCDs hereafter) in which the change is sought on a descriptive model of the data, rather than on the data itself. More precisely, the quality measure tracked by PBCDs for detecting changes is the dissimilarity between sets of patterns (e.g. binary Jaccard measure [9], Levenshtein measure [6]) discovered before and after the arrival of new transactions. In PBCDs, the complete set of frequent patterns is mined upon the arrival of new transactions (*mining step*), before measuring the quality measure (*detection step*).

Typically, the mining step relies on exhaustive algorithms leading to *complete* pattern mining. The main intuition for this solution is that *completeness* is desirable to accurately model the data. However, in practical cases, only a small portion of patterns is likely to be relevant for the detection. Furthermore, exhaustive pattern mining may represent the major obstacle for any PBCD that needs to timely react to incoming data. Our main claim is, therefore, that it is possible to relax the completeness property and it is possible to adopt non-exhaustive mining methods in change detection without loosing in accuracy. Simultaneously, this relaxation can improve the efficiency of the mining step and allow the PBCD system to quickly react. To the best of our knowledge, the StreamKRIMP algorithm [8] is the only PBCD adopting a non-exhaustive mining method, which is based on the MDL principle. However, StreamKRIMP is designed for transactional data streams and not for dynamic networks. Moreover, few other attempts can be retraced in pattern-based anomaly detection methods [5] and subgroup discovery [10], where the search space is pruned according to heuristic evaluations. An alternative line of research for reducing the patterns in PBCDs is that of exhaustively mining condensed sets of patterns which are representative of all the possible patterns. In particular, both non-derivable patterns [7] and Δ-closed patterns [11] have been proposed for anomaly detection and change detection, respectively. However, in such approaches, condensed sets of patterns are discovered by exhaustive mining procedures, and hence they do not provide solutions to the computational issues of PBCDs.

By taking into account the aforementioned reasons, this paper extends the PBCD methodology originally proposed in KARMA [9], based on exhaustive frequent connected subgraph mining, with non-exhaustive mining algorithms. In particular, we customize the general architecture inherited by the KARMA

algorithm with improved mining step and detection step, then we perform an extensive study to select the most accurate and the most efficient PBCDs. Experiments show that the proposed approaches improve the efficiency and the detection accuracy with respect to their exhaustive counterpart, on both real world and synthetic dynamic networks.

2 Background

Let N be the set of nodes, L be the set of edge labels, and $I = N \times N \times L$ the alphabet of all the possible labeled edges, on which a lexicographic order $\geq$ is defined. A dynamic network is represented as the time-ordered stream of graph snapshots $D = \langle G_1, G_2, \ldots, G_n \rangle$. Each snapshot $G_i \subseteq I$ is a set of edges denoting a directed graph observed in t_i, which allows self-loops and multiple edges with different labels. G_i is uniquely identified by id i. Let G be a directed graph, a *connected subgraph* $S \subseteq G$ is a directed graph such that for any pair of nodes (in S) there exists a path connecting them. Then, a *subtree* $S \subseteq G$ is a connected subgraph in which every node (in S) is connected to a unique parent node, except for the root node.

The data representation fits with the one adopted in transactional data mining, allowing the mining of frequent patterns by adapting traditional frequent itemset mining algorithms. In this perspective a snapshot $G_{tid} \in D$ is a transaction uniquely identified by *tid*, whose items are labeled edges from I. While a pattern $P \subseteq I$, with *length* $|P|$, can be seen as a word $P = \langle i_1 \ldots i_n \rangle$ of n lexicographic sorted items, with prefix $S = \langle i_1 \ldots i_{n-1} \rangle$ and suffix i_n. The *tidset* of P in the network D is defined as $tidset(P, D) = \{tid \mid \exists G_{tid} \in D \wedge P \subseteq G_{tid}\}$, while the *support* of P in D is $sup(P, D) = \frac{|tidset(P,D)|}{|D|}$. P is *frequent* in D if $sup(P, D) > \alpha$, where $\alpha \in [0, 1]$.

In PBCDs designed for network data, we deem as interesting two types of patterns: (i) *frequent connected subgraphs* (FCSs) and, (ii) *frequent subtrees* (FSs). Both FCSs and FSs are mined from snapshots belonging to time windows. A window $W = [t_i, t_j]$, with $t_i < t_j$, is the sequence of snapshots $\{G_i, \ldots, G_j\} \subseteq D$. Consequently, the width $|W| = j - i + 1$ is equal to the number of snapshots collected in W. For our convenience we term F_W the set of all the FCSs (FSs) in the window W.

2.1 Problem Statement

Let $D = \langle G_1, G_2, \ldots, G_n \rangle$ a dynamic network, $\alpha \in [0, 1]$ be the minimum support threshold, $\beta \in [0, 1]$ the minimum change threshold. Then, pattern-based change detection *finds* pairs of windows $W = [t_b, t_e]$ and $W' = [t'_b, t'_e]$, where $t_b \leq t'_b \leq t_{e+1}$ and $t_e < t'_e$, satisfying $d(F_W, F_{W'}) > \beta$, where (i) F_W and $F_{W'}$ are the sets of patterns discovered on W and W' according to α, and (ii) $d(F_W, F_{W'}) \in [0, 1]$ is a dissimilarity measure between sets of patterns. In this perspective, changes correspond to significant variations in the set of patterns discovered on two windows, which denote *stable* features exhibited by the graph snapshots.

3 Architecture of a PBCD

The aforementioned change detection problem can be solved by various computational solutions. In this section we provide the general architecture of a PBCD for network data, by generalizing the algorithm KARMA proposed in [9].

In general, a PBCD forms a two-step approach in which: (i) a pattern mining algorithm extracts the set of patterns observed from the incoming data, and (ii) the amount of change is quantified by adopting a dissimilarity measure defined between sets of patterns. Practically speaking, a PBCD is an iterative algorithm that consumes data coming from a data source, in our case a dynamic network, and produces quantitative measures of changes. In particular, the KARMA algorithm is a PBCDs based on exhaustive mining of FCSs, whose general workflow can be seen in Fig. 1. The algorithm iteratively consumes blocks Π of graph snapshots coming from D (Step 2) by using two successive landmark windows W and W' (Step 3). This way, it mines the complete sets of FCSs, F_W and $F_{W'}$, necessary to the detection step (Steps 4–5). The window grows ($W = W'$, Step 8) with new graph snapshots, and the associated set of FCSs is kept updated (Step 9) until the Tanimoto coefficient $d(F_W, F_{W'})$ exceeds β and a change is detected. In that case, the algorithm drops the content of the window by retaining only the last block of transactions ($W = \Pi$, Steps 6–7). Then, the analysis restarts. The KARMA algorithm offers a general architecture for building custom PBCD, which is made of 4 components: (i) the *window model* (Fig. 1, Steps 3, 8 and 6), (ii) the *feature space* (FCSs or FSs), (iii) the *mining step* (Fig. 1, Steps 4, 9 and 7), and (iv) the *detection step* (Fig. 1, Step 5). In the following sections we will focus on both the mining step and the detection step, also by commenting their contribution to the efficiency of the PBCD strategy.

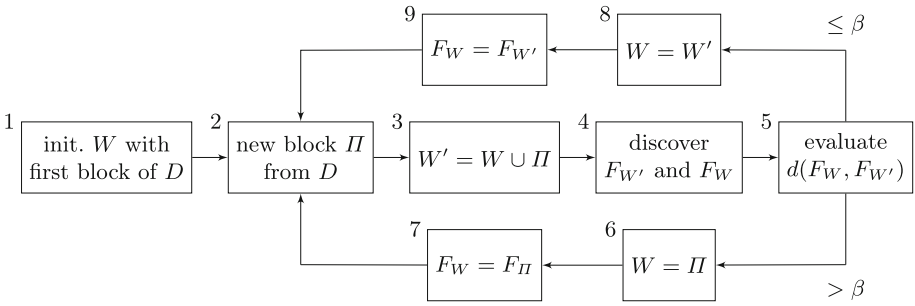

Fig. 1. The KARMA algorithm workflow

Here, we briefly discuss the choice of an appropriate time window model of a PBCD. We deem as interesting 3 models: the *landmark model*, the *sliding model* and the *mixed model*. They differ in the way they consume the incoming block Π of graph snapshot. In its original version, KARMA uses the *landmark model*. Here, Π is added to the window W, forming the successive window $W' = W \cup \Pi$

(Fig. 1, Step 3). In this model, the window grows until a change is detected and when a change is detected, old data are discarded. In the *sliding model*, the detection is performed on two successive windows W and Π of fixed size. The windows are non overlapping and they always slides forward, both in case of change detected and not detected ($W = \Pi$). Therefore, old data are always discarded. In the *mixed model*, the detection is performed on W and Π, as in the sliding model. However, as in the landmark model, Π is added to W, forming $W' = W \cup \Pi$ until a change is detected. In that case, old data are discarded.

4 Exhaustive and Non-exhaustive Mining in PBCDs

The main difference between exhaustive and non-exhaustive PBCDs lies in the *exhaustiveness* of the mining step used to discover the patterns, which is the major bottleneck of any exhaustive PBCD approach. In fact, the discovery of an exponentially large number of patterns affects the efficiency of both the mining and detection step, hence rendering this class of algorithms not efficient in practice. The main objective of this paper is to reduce the computational complexity of exhaustive approaches by adopting non-exhaustive ones. In particular, we propose a mining algorithm able to prune the search space of patterns following a beam-search approach.

Being based on beam-search, the proposed approach relies on a parameter k which controls the *beam size* of the mining step when traversing the search space of patterns, that is a lattice $L = (2^I, \subseteq)$ ordered by the generality relation $\subseteq$, conveniently represented in a SE-Tree data structure. Since an exhaustive search can be achieved with non-exhaustive procedures by setting $k = |I|$, we refer to Algorithm 1 in both cases. In particular, the algorithm implements a pattern-growth approach for mining patterns F_W in a time window W, and it is initially called with empty prefix $\emptyset$. An important remark is that exhaustive PBCDs rely on complete pattern sets, discovered by the exhaustive mining procedure, as the feature sets for the detection problem. On the contrary, non-exhaustive PBCDs rely only on limited pattern sets discovered by the non-exhaustive mining procedure.

4.1 Exhaustive FCSs and FSs Mining

The mining procedure (Algorithm 1) takes 4 input parameters, that is the content of the window W, the minimum support threshold α, the beam-size k, and the pattern prefix (initially equals to $\emptyset$). The algorithm exhaustively traverses the search space of FCSs and FSs by setting $k = |I|$, following a recursive DFS approach. In particular, it is able to *(i)* build patterns with a pattern-growth approach in which items are appended as suffix to a pattern prefix, and *(ii)* evaluate the supports through tidset intersection. The result is the complete set of the frequent patterns F_W in W according to α.

The procedure considers the window W as an i-conditional database of transactions in which every item $j \leq i$ has been removed, as done in [4]. At the beginning of each recursive call, Line 2 initializes the set $F[P]$ of frequent patterns on

Algorithm 1: Mining procedure based on beam search

```
   Output: F[P] the set of frequent patterns having prefix P
1  minePatterns (W, α, k, P)
2      F[P] = ∅, Items = ∅, Beam = ∅
3      for (i, tidset) occurring in W do
4          if |tidset| / |W| ≥ α then
5              if isValid(P ∪ {i}) then
6                  F[P] = F[P] ∪ (P ∪ {i})
7              end
8              Items = Items ∪ {i}
9          end
10     end
11     Beam = topKSortedBySupport(Items, k)
12     for all i occurring in Beam do
13         W^i = ∅
14         for j occurring in Beam | j > i do
15             C = tidset(i, W) ∩ tidset(j, W)
16             W^i = W^i ∪ {(j, C)}
17         end
18         if isValid(P ∪ {i}) then
19             F[P ∪ {i}] = minePatterns(W^i, α, k, P ∪ {i})
20             F[P] = F[P] ∪ F[P ∪ {i}]
21         end
22     end
23     return F[P]
```

prefix P as empty. Then, Lines 3–10 exploit the vertical layout of W, and test the supports against the threshold α. The FCS (FS) $P \cup \{i\}$ is built by appending the item i to the prefix P only when allowed by the predicate $isValid$ (Line 4), which checks whether $P \cup \{i\}$ is a connected subgraph (when mining FCSs) or a subtree (when mining FSs), respectively. Lastly, they are added to the set $F[P]$. The algorithm adds the suffix i of any pattern discovered to $Items$.

Line 11 selects only the most promising subset of k patterns, according to their support. In practice, this line is irrelevant in exhaustive mining as it will always select all the patterns, since $k = |I|$. Then, lines 12–22 build the i-conditional databases on which to perform recursive calls. In particular, the algorithm iterates over each item i in $Beam$, and Line 13 initialize the associated i-conditional database W^i as empty. Lines 14–17, iterate on items j from $Beam$ such that $j > i$. This way, Line 15 computes the tidset C as the set intersection between the tidsets of i and j in the database W, respectively. Then, C is the tidset of j in the newly created i-conditional database W^i. The mining procedure is recursively called at Line 19 for mining the set $F[P \cup \{i\}]$ of FCSs (or FSs) with valid prefix $P \cup \{i\}$, according to the pattern language. This way, subgraphs which are not connected (or do not represent trees) are pruned at Line 18. Finally, the patterns in $F[P \cup \{i\}]$ are added to $F[P]$, which is returned as the final result.

The exhaustive mining of FCSs and FSs requires time proportional to $O(2^{|I|})$ in the worst case scenario, in line with that of traditional frequent itemset mining. Moreover, due to the constraints imposed by the pattern language, the number of FCSs and FSs is in practice much lower than the number of itemsets,

FSs $<$ FCSs $< 2^{|I|}$. However, the mining time is still exponential in the number of edges $|I|$, thus resulting inefficient.

4.2 Non-exhaustive FCSs and FSs Mining

The non-exhaustive mining is achieved by pruning the search space of FCSs and FSs according to some heuristic criteria. In particular, when calling Procedure 1 with $k < |I|$, the intermediate selection step (Line 11) selects only the most promising subset of k patterns to further advance the process (Lines 12–22). As in traditional beam search-based algorithms, frequent patterns in a recursive call are evaluated by means of a heuristic evaluation function and sorted in increasing order. Then, only the first top-k of them are further considered.

Many heuristic evaluation functions can be used to select the most promising subset of patterns, among them we adopt the support of patterns. In particular, Line 11 sorts the patterns in $F[P]$ according to their support, then only k of them with the greatest support are kept to further advance the process, while the remaining ones are ignored. However, since the sorting is performed between patterns having *same length*, the evaluation based on the support is consistent with the evaluation based on the *area*.

Proof. Let S be a pattern, and W be a window. Then, the support $sup(S, W) = \frac{|tidset(S,W)|}{|W|}$ and the area $area(S, W) = |S| \cdot |tidset(S, W)|$ are linearly proportional to $|tidset(S, W)|$ with two constant factors $\frac{1}{|W|}$ and $|S|$.

The area of the FCS (or FS) S in the window W, $area(S, W)$, is an *interestingness* measure adopted in *tile mining* [4]. In our case, it is used to restrict the search space by considering only the most *interesting* patterns at each recursion step. This simple, yet effective approach allows us to significantly prune the search space when mining the limited sets F_W and $F_{W'}$. In particular, the mining is more focused towards patterns covering large portions of the window, *tiles* from a transactional database point of view [4], and hence more interesting.

The non-exhaustive mining procedure is more efficient than the exhaustive one, requiring time proportional to $O(2^k)$ in the worst-case scenario. In fact, the algorithm restricts the attention on only k items from I in the base recursion step, with $k \ll |I|$.

5 Detecting Changes on Pattern Sets

Once the complete or limited pattern sets, F_W and $F_{W'}$, have been discovered by either the exhaustive or non-exhaustive procedure, respectively, the detection step can be executed and the dissimilarity score $\beta = d(F_W, F_{W'})$ computed. We recall that $d(F_W, F_{W'})$ is a binary dissimilarity measure defined on sets of patterns. For our convenience we define it as operating on the vector encoding $\mathbf{w}$ and $\mathbf{w}'$ of F_W and $F_{W'}$, respectively.

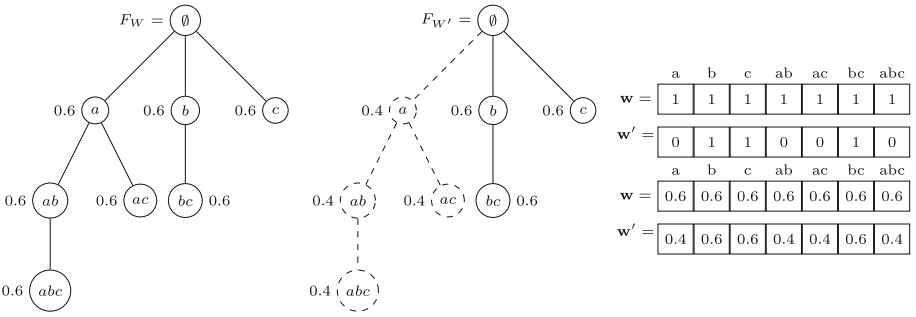

Fig. 2. Example of binary (top) and real-valued (bottom) vector encoding of F_W and $F_{W'}$. Dashed circles denote infrequent patterns with $\alpha = 0.5$.

5.1 Detecting Changes on Complete Pattern Sets

When detecting changes on complete pattern sets, the encoding is built by enumerating the patterns in $F_W \cup F_{W'}$. More specifically, $\mathbf{w}$ ($\mathbf{w}'$) is a vector of size $n = |F_W \cup F_{W'}|$, where the i-th element is a weight associated to the i-th pattern in the enumeration of $F_W \cup F_{W'}$ with respect to W (or W', respectively). Then, a change is detected if the dissimilarity score exceeds the minimum threshold β, that is when $d(F_W, F_{W'}) > \beta$.

In the case of KARMA, as shown in Fig. 2, $\mathbf{w}$ and $\mathbf{w}'$ are binary vectors indicating whether each FCS from the enumeration is frequent or not in W and W', respectively. Then the algorithm computes the *Tanimoto coefficient* $d(F_W, F_{W'}) = 1 - \frac{\mathbf{w} \cdot \mathbf{w}'}{\|\mathbf{w}\|^2 + \|\mathbf{w}'\|^2 - \mathbf{w} \cdot \mathbf{w}'}$. By doing so, KARMA quantifies the fraction of FCSs which have crossed the minimum support threshold, thus indicating a relevant change in the underlying graph data distribution. However, this solution does not take into account the FCSs not crossing the minimum support threshold, although exhibiting a potentially significant support spread.

To overcome this limitation, an alternative approach also shown in Fig. 2 is to build the vector encoding as real-valued vectors of supports in W and W', respectively. Then, it is possible to compute the *weighted Jaccard dissimilarity* $d(F_W, F_{W'}) = 1 - \frac{\sum_i min(\mathbf{w}_i, \mathbf{w'}_i)}{\sum_i max(\mathbf{w}_i, \mathbf{w'}_i)}$. We deem this measure as relevant because relates the dissimilarity to the *absolute growth-rate* [2] of each pattern S, defined as $GR(S, W, W') = |sup(S, W) - sup(S, W')|$. The *absolute growth-rate* is a contrast measure adopted in *emerging pattern mining* to discover contrast patterns between two datasets [2].

Proof. Given the analytic formulations for $max(a, b) = \frac{1}{2}(a + b + |a - b|)$ and $min(a, b) = \frac{1}{2}(a + b - |a - b|)$, and the vector encoding $\mathbf{w}$ and $\mathbf{w}'$ of F_W and $F_{W'}$. The weighted Jaccard dissimilarity can be rewritten as $d(F_W, F_{W'}) = 1 - \frac{\sum_i min(\mathbf{w}_i, \mathbf{w'}_i)}{\sum_i max(\mathbf{w}_i, \mathbf{w'}_i)} = 1 - \frac{\sum_i \mathbf{w}_i + \mathbf{w'}_i - GR(S_i, W, W')}{\sum_i \mathbf{w}_i + \mathbf{w'}_i + GR(S_i, W, W')}$.

In exhaustive PBCDs the number of patterns grows exponentially in the number of items. Therefore, regardless from the measure $d(F_W, F_{W'})$ adopted, the

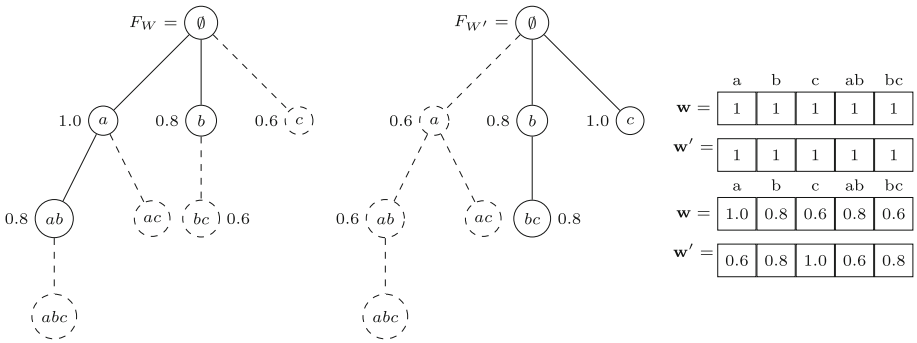

Fig. 3. Example of limited sets of frequent patterns F_W (left) and $F_{W'}$ (right) discovered with $k = 2$ and $\alpha = 0.1$. Dashed circles denote pruned non-interesting patterns.

detection step on complete pattern sets requires an amount of time proportional to the number of patterns in the enumeration.

5.2 Detecting Changes on Limited Pattern Sets

Non-exhaustive PBCDs detect changes in the same way exhaustive PBCDs do, that is by computing the score $d(F_W, F_{W'})$ in terms of the *Tanimoto coefficient* or the *unweighted Jaccard dissimilarity*, and testing it against the minimum change threshold β. Although the detection approach remains the same, a subtle difference in the meaning of the detection is present. In fact, while the dissimilarity measures adopted on complete pattern sets quantify how much the supports of FCSs (FSs) change between W and W', they do not consider the *interestingness* of patterns on W and W', respectively, as intended by the non-exhaustive mining algorithm.

Non-exhaustive mining based on the interestingness prunes the search space of patterns in two different ways for W and W', respectively. Thus restricting the search only to the most interesting FCSs and FSs, while discarding the less interesting ones (Fig. 3). By doing this, the detection relies on a considerably low number of patterns, hence resulting more efficient, while losing information associated to patterns which have been pruned. This affects the construction of the vector encoding $\mathbf{w}$ and $\mathbf{w}'$, which is built according to the enumeration $F_W \cup F_{W'}$ consisting of a reduced number of patterns. The example reported in Fig. 3 depicts a scenario in which every pattern is frequent in both W and W', although with different supports, thus determining different interestingness. Any information related to the patterns "*ac*" and "*abc*" is lost, as they are not present in the enumeration of $F_W \cup F_{W'}$, and therefore they do not contribute to the change.

Therefore, the detection step becomes non-exhaustive itself, by focusing the detection only on the most interesting frequent patterns. In particular, as the number of patterns discovered by the non-exhaustive mining procedure grows

exponentially with the parameter $k \ll |I|$, the detection step requires in practice much smaller time than that required on complete pattern sets.

6 Computational Complexity

In this section we study the computational complexity of PBCDs in the worst-case scenario. In particular, the analysis takes into account the influence of the *feature space*, the *mining step*, the *detection step* and *the window model*. Given the dynamic network $D = \langle G_1, G_2, \ldots, G_n \rangle$ where I denotes the possible labeled edges observed over the time, and $|\Pi|$ the size of blocks, then every PBCD built according to the architecture in Fig. 1 consumes exactly $e = \frac{n}{|\Pi|}$ blocks of transactions, thus requiring $O(e)$ iterations. The time complexity $O(a + b)$ required during every iteration depends on the cost a of the mining step, and the cost b of the detection step.

The mining step requires time complexity $a = O(2^c) \cdot d$ in the worst case scenario. $O(2^c)$ denotes the number of patterns discovered according to the *feature space* and to the *exhaustiveness* of the mining step. In the exhaustive setting, all the edges ($c = |I|$) are considered to discover $O(2^{|I|})$ patterns. Since $FSs < FCSs < 2^{|I|}$, we refer to $O(2^{|I|})$ as the maximum number of patterns discovered in the worst-case scenario. However, it reduces to $O(2^k)$ in the non-exhaustive setting ($c = k$), where $k \ll |I|$. The term d denotes a multiplicative factor describing the amount of work spent by the algorithm in tidset intersections, which depends on the *time window model* adopted. In the case of landmark and mixed model it is $O(|W| + |\Pi|)$, while in the case of the sliding model it is $O(|\Pi|)$. As for the detection step, the computation of the $d(F_W, F_{W'})$ requires time complexity $O(b)$ proportional to the enumeration of patterns $|F_W \cup F_{W'}|$, which is $O(2^{|I|})$ in exhaustive setting, and $O(2^k)$ in the non-exhaustive one.

Then, the computational complexity in the worst case scenario of exhaustive PBCDs is $O(e \cdot (d2^{|I|} + 2^{|I|}))$, while for non-exhaustive PBCDs is $O(e \cdot (d2^k + 2^k))$. Therefore, it is exponential in $|I|$ and k, with $k \ll |I|$, respectively.

7 Experimental Results

The experiments are organized alongside different perspectives concerning both synthetic and real-world dynamic networks. In particular, we answer the following research question: **(Q1)** What is the best PBCD in terms of *efficiency and accuracy* when tuning the minimum change threshold β on synthetic networks? **(Q2)** How much the parameter k affects the *efficiency* and the *accuracy* of non-exhaustive PBCD on synthetic networks? **(Q3)** How much the parameter k affects the *efficiency* of non-exhaustive PBCD on real-world networks?

For experiments on synthetic networks, we generated 40 networks, 20 with frequent drifts and 20 with rare drifts. Every network consists of 200 hourly blocks made of 120 graph snapshots, one observed every 30 s, for a total amount of 24000 snapshots. Each hourly block is built by randomly choosing with

Table 1. Most accurate (top) and most efficient (bottom) PBCD when tuning β.

PBCD component	Most accurate PBCD @ β											
	0.10		0.20		0.30		0.40		0.50		0.60	
	choice	p-val.	choice	p-val.	choice	p-val.	choice	p-val.	choice	p-val.	choice	p-val.
Features (fcs/fs)	fs	0.0025	fs	0.0001	fs	0.0001	fs	0.0317	fs	0.2357	fs	0.6816
Mining (ex/nex)	nex	0.0001	nex	0.0001	nex	0.0001	nex	0.0828	nex	0.4657	nex	0.7154
Detection (tan/wj)	tan	0.3618	wj	0.0001	wj	0.0001	wj	0.0001	wj	0.0001	wj	0.1004
Windows (lan/sli/mix)	lan	0.0001	mix	0.0001	mix	0.0001	mix	0.0001	mix	0.0001	mix	0.0001
PBCD component	Most efficient PBCD @ β											
	0.10		0.20		0.30		0.40		0.50		0.60	
	choice	p-val.	choice	p-val.	choice	p-val.	choice	p-val.	choice	p-val.	choice	p-val.
Features (fcs/fs)	fs	0.0001	fs	0.0001	fs	0.0001	fs	0.0001	fs	0.0001	fs	0.0001
Mining (ex/nex)	nex	0.0001	nex	0.0001	nex	0.0001	nex	0.0001	nex	0.0001	nex	0.0001
Detection (tan/wj)	tan	0.0001	tan	0.0001	tan	0.0001	tan	0.0001	tan	0.0079	tan	0.3618
Windows (lan/sli/mix)	sli	0.0001	sli	0.0001	sli	0.0001	sli	0.0001	sli	0.0001	sli	0.0001

replacement (i) one out of 10 different generative models in the case of frequent drifts, and (ii) one out of 2 different generative models in the case of rare drifts. As a consequence, it is more likely that two consecutive hourly blocks are built according to different generative models, thus denoting a change, in the dataset with frequent drifts than in the one with rare drifts. Every generative model builds a first snapshot made of 50 nodes by adopting a random scale-free network generator, which is then replicated for the remaining snapshots of the block. Every graph snapshot of a block is then perturbed by adding new edges and removing existing ones with a probability equals to 2%. A random perturbation is required to test the false alarm rate of the two approaches.

7.1 Q1: The Most Accurate and Most Efficient PBCD When Tuning β

In this paper, we discussed various components of PBCDs, that is (i) 2 mining steps (exhaustive and non-exhaustive), (ii) 2 feature spaces (FCSs and FSs), (iii) 2 detection steps (with the Tanimoto dissimilarity score and the weighted Jaccard score), and (iv) 3 time window models (landmark, sliding and mixed). These components can be combined to form 24 possible PBCDs, and hence determining variants of the original KARMA algorithm. Here, we evaluate which one performs statistically better, by measuring the efficiency (*running times*) and the change detection accuracy (*Accuracy*). We executed the 24 variants on 40 randomly generated synthetic networks, by tuning β 6 times (resulting in 5760

Table 2. Running times of PBCD-1 and PBCD-2, when tuning k, against KARMA and StreamKRIMP on synthetic data ($\alpha = 0.5$, $\beta = 0.20$, $|\Pi| = 15$).

Dataset	Running times (s) @ k													
	PBCD-1						PBCD-2						KARMA	KRIMP
	5	10	15	20	25	30	5	10	15	20	25	30		
freq-drifts-1	6.016	8.536	12.913	16.156	19.234	23.745	3.881	5.612	6.194	7.870	9.091	10.372	60.763	86.130
freq-drifts-2	6.022	9.518	12.284	15.758	19,270	23.084	4.147	5.083	6.522	7.826	9.141	10.182	55.982	77.138
freq-drifts-3	6.689	8.882	12.603	15.710	19.653	22.988	3.195	4.497	6.463	7.667	9.818	10.599	58.137	76.750
freq-drifts-4	6.107	9.739	12.792	17.029	20.976	23.980	3.778	4.529	6.712	8.129	9.800	10.419	78.240	23.213
rare-drifts-1	7.438	12.578	18.358	24.582	29.271	35.250	3.341	4.709	5.854	7.556	9.005	10.034	1775.234	109.816
rare-drifts-2	7.302	12.326	17.549	23.911	29.702	33.339	4.182	5.156	6.435	7.563	8.928	10.365	1971.625	112.303
rare-drifts-3	7.181	12.696	18.259	25.245	29.642	35.035	3.933	5.132	6.228	7.236	8.608	10.102	1971.367	109.395
rare-drifts-4	7.273	12.159	17.901	23.821	28.416	31.260	4.059	4.306	6.452	7.488	9.072	10.559	2026.794	116.141

executions) and fixing the value of k to 20 items. Then, we selected the most accurate and the most efficient PBCD (Table 1) by using (i) a Wilcoxon post-hoc test when deciding about the feature space, the mining strategy and the detection step, and (ii) a Nemenyi-Friedman post-hoc test when deciding about the best time-windows model, both at significance level $\alpha = 0.05$.

As for the accuracy, the results show that FSs are more appropriate features than FCSs. Furthermore, the PBCDs equipped with non-exhaustive mining step always outperforms exhaustive ones. The Tanimoto measure, as originally used by the KARMA algorithm, outperforms the weighted Jaccard measure for low values of β. From this set of experiments it is clear that the factors that impact the most on the PBCDs accuracy are the change detection measure and the time windows model. In particular, it is strongly evident that a mixed model outperforms the landmark model, which is preferred only when $\beta = 0.10$.

As for the efficiency, the very low p-values indicate a strong evidence that non-exhaustive PBCD based on FSs and the Tanimoto distance in the sliding model, outperforms every other PBCD approach. In particular, this is an expected result since: (i) the mining of FSs is less time consuming than FCSs, (ii) a non-exhaustive mining strategy is more efficient than exhaustive one.

An aspect worth to be considered is that the original KARMA algorithm (*FCSs + EX + Tanimoto + Landmark*) is never selected as the best PBCD. In this perspective the adoption of a new feature set, the FSs, jointly with a non-exhaustive mining step generally improves the detection accuracy and the efficiency. However, the test suggests that the landmark model adopted by the KARMA algorithm is a bad choice leading to poor accuracy and efficiency. While the Tanimoto coefficient leads to poor detection accuracy.

7.2 Q2: Efficiency and Accuracy of Non-exhaustive PBCDs on Synthetic Networks

We report the results of a comparative evaluation in which we compare the running time (Table 2), the accuracy and the false alarm rate (Table 3) of two non-exhaustive PBCDs against the KARMA [9] and the StreamKRIMP [8] state-of-the-art PBCDs. In particular, StreamKRIMP treats the network as a data stream of labeled edges, and adopts a compression-based mining step. We select two non-exhaustive PBCDs emerged in the last section and test their performances on 8 synthetic networks, when tuning the parameter k. In particular, we chose the most efficient PBCD and the most accurate PBCD when $\beta = 0.2$. We denote them as PBCD-1 (*FSs + NEX + Weighted Jaccard + Mixed*), and PBCD-2 (*FSs + NEX + Tanimoto + Sliding*), respectively. The results in Tables 2 and 3 shows increasing efficiency and accuracy for decreasing values of k.

Results in Table 2 show an improved efficiency for both PBCD-1 and PBCD-2 with respect to KARMA and StreamKRIMP. This is an expected result, also confirmed by the statistical significance test in Sect. 7.1, because non-exhaustive mining of FSs is more efficient than (i) the exhaustive mining of FCSs performed by the KARMA algorithm, and (ii) the non-exhaustive mining of itemsets performed by StreamKRIMP. In particular, the running times of both PBCD-1 and PBCD-2 increases with k, as high values of k lead to the discovery of an increasing number of patterns. Moreover, the results show that PBCD-2 is more

Table 3. Accuracy and false alarm rate of PBCD-1 and PBCD-2, when tuning k, against KARMA and StreamKRIMP on synthetic data ($\alpha = 0.5$, $\beta = 0.20$, $|\Pi| = 15$).

Dataset	Accuracy @ k													
	PBCD-1						PBCD-2						KARMA	KRIMP
	5	10	15	20	25	30	5	10	15	20	25	30		
freq-drifts-1	1.0	1.0	0.987	0.987	0.967	0.957	1.0	1.0	0.918	0.891	0.824	0.751	0.8041	0.9299
freq-drifts-2	1.0	1.0	0.991	0.991	0.965	0.952	1.0	1.0	0.916	0.889	0.819	0.735	0.7985	0.9105
freq-drifts-3	1.0	1.0	0.988	0.988	0.958	0.948	1.0	1.0	0.918	0.888	0.819	0.755	0.7960	0.9155
freq-drifts-4	1.0	1.0	0.987	0.987	0.963	0.952	1.0	1.0	0.936	0.907	0.831	0.757	0.7860	0.923
rare-drifts-1	1.0	1.0	1.0	1.0	1.0	0.971	1.0	1.0	0.816	0.816	0.691	0.598	0.9362	1.0
rare-drifts-2	1.0	1.0	1.0	1.0	1.0	0.967	1.0	1.0	0.799	0.799	0.674	0.571	0.9399	1.0
rare-drifts-3	1.0	1.0	1.0	1.0	1.0	0.969	1.0	1.0	0.816	0.816	0.691	0.599	0.9368	1.0
rare-drifts-4	1.0	1.0	1.0	1.0	1.0	0.965	1.0	1.0	0.799	0.799	0.674	0.576	0.9293	1.0
Dataset	False alarm rate @ k													
	PBCD-1						PBCD-2						KARMA	KRIMP
	5	10	15	20	25	30	5	10	15	20	25	30		
freq-drifts-1	0	0	0.014	0.014	0.036	0.048	0	0	0.092	0.123	0.197	0.279	0.1099	0.0399
freq-drifts-2	0	0	0.011	0.0105	0.039	0.053	0	0	0.095	0.125	0.204	0.297	0.1131	0.0513
freq-drifts-3	0	0	0.013	0.0134	0.047	0.058	0	0	0.092	0.126	0.203	0.275	0.1146	0.0478
freq-drifts-4	0	0	0.015	0.0148	0.042	0.054	0	0	0.072	0.105	0.190	0.273	0.1205	0.0437
rare-drifts-1	0	0	0	0	0	0.0312	0	0	0.195	0.195	0.328	0.427	0.0073	0
rare-drifts-2	0	0	0	0	0	0.032	0	0	0.214	0.214	0.347	0.457	0	0
rare-drifts-3	0	0	0	0	0	0.0326	0	0	0.196	0.196	0.329	0.427	0.0027	0
rare-drifts-4	0	0	0	0	0	0.0377	0	0	0.216	0.216	0.351	0.456	0.0013	0

efficient than PBCD-1, as the sliding window model leads to increasing efficiency with respect to the mixed model. This is explained by the forgetful nature of the sliding window, in which old graph snapshots are immediately discarded with the arrival of a new block of snapshots. In this way, the mining step requires reduced computational efforts, as patterns are mined from reduced sets of transactions. Thus intersecting small tidsets when computing the support of each pattern.

As for the accuracy (Table 3), both PBCD-1 and PBCD-2 are optimal change detection solutions, for the considered synthetic networks, for low values of k ($k = 5$ and $k = 10$, respectively). Moreover, the results show a decreasing tendency in the accuracy of both PBCD-1 and PBCD-2. In particular, PBCD-1 always outperforms KARMA and StreamKRIMP (except for $k = 30$), while this is not the case of PBCD-2. These are expected results, again confirmed by the significance test in Sect. 7.1, as the non-exhaustive mining of FSs with a detection step based on the weighted Jaccard dissimilarity takes into account the absolute growth-rate and the interestingness of patterns. This is not the case of PBCD-2, in which the Tanimoto coefficient computed in the sliding window setting, on large sets of patterns, exhibits higher false positive rates. For high values of k, the mining step discover patterns representing behavior local to the snapshots collected in two successive sliding windows of equal size. Thus, injecting noisy features in the detection step. We note that (i) PBCD-1 exhibits moderately lower false alarm rates than PBCD-2, also outperforming KARMA for high values of k, and StreamKRIMP for low values of k, and (ii) PBCD-2 outperforms KARMA and StreamKRIMP for low values of k only.

Table 4. Running times of PBCD-1 and PBCD-2, when tuning k, against KARMA on real-world networks ($\beta = 0.20$, $|\Pi| = 10\%$ of each dataset).

Dataset	Running times (s) @ k												
	PBCD-1						PBCD-2						KARMA
	5	10	15	20	25	30	5	10	15	20	25	30	
mawi	6.769	9.68	1.919	14.886	17.554	18.847	5.169	6.17	6.82	7.977	8.547	8.736	86.493
noaa	14.794	16.513	18.068	19.935	19.603	23.217	15.164	16.35	17.18	19.061	20.533	20.263	65.697
nodobo	1.72	1.525	2.11	2.915	3.728	4.371	1.825	1.582	1.822	2.587	3.276	3.988	35.253
keds	0.955	0.924	0.873	0.946	1.109	1.034	0.979	0.892	0.795	0.982	1.075	1.014	1.108
wikitalks	83.846	78.454	80.182	77.904	81.605	79.131	77.128	76.77	76.021	80.45	77.299	75.876	94.914

We conclude that both the accuracy and the efficiency of non-exhaustive PBCDs benefits from the limited pattern sets which have been discovered. In particular, the combination of a mixed window model with the weighted Jaccard dissimilarity leads to accurate detection, while the combination of sliding windows and the Tanimoto coefficient leads to efficient detection, while improving the detection accuracy for very low values of k. From this perspective, the two approaches offers two efficient alternatives to the KARMA algorithm, in which the running times can be greatly reduced (up to two orders of magnitude in this set of experiments).

7.3 Q3: Efficiency of Non-exhaustive PBCDs on Real-World Networks

We also provide a practical idea of the efficiency on real-world networks by reporting the results of a comparative evaluation (Table 4) between PBCD-1, PBCD-2 and KARMA on 5 real-world networks, when tuning k. More specifically, we used the same networks adopted in [9]: the *keds*, *mawi*, *noaa*, *nodobo* and the *wikitalks* dataset. To guarantee a fair comparison, we fixed the value $\beta = 0.20$ in each experiment. However, since the networks span different periods, we independently fixed the size of block $|\Pi|$ to the 10% of each dataset, this guaranteed 100 iterations of each PBCD on every dataset. The minimum support α has been fixed to 0.05 for keds, nodobo and mawi, 0.20 for noaa, and 0.40 for wikitalks. From the obtained results it is evident that both PBCD-1 and PBCD-2 are always more efficient than KARMA for all the values of k. Furthermore, as observed on synthetic networks in Sect. 7.2 and as confirmed in Sect. 7.1, PBCD-2 continues to be more efficient than PBCD-1. The increasing tendency of the running times with k is verified in the *mawi*, *noaa* and *nodobo* datasets.

8 Conclusions

In this paper, we have collected several improvements contributing to the efficiency and the accuracy of traditional PBCDs. This have been possible by inheriting the general PBCD schema from the KARMA algorithm, and extending it. Specifically, we have relaxed the exhaustiveness of the PBCDs mining step with a non-exhaustive mining strategy, inspired by beam search algorithms. The effect is that the mining algorithm discovers now limited sets of patterns by pruning the search space according to the interestingness of patterns. Moreover, we proposed an extended detection step which takes into account the growth-rate of the discovered patterns. Ultimately, we have conducted an extensive exploratory evaluation on both real and synthetic networks.

The experiments have shed some lights on the most accurate and on the most efficient PBCDs among the possible approaches. Furthermore, they have shown that non-exhaustive PBCDs are more efficient than exhaustive PBCDs, while achieving comparable levels of accuracy. Future directions of research involve the evaluation of the performances when adopting more sophisticated feature spaces, for example by considering graph embedding.

References

1. Akoglu, L., Tong, H., Koutra, D.: Graph based anomaly detection and description: a survey. Data Min. Knowl. Discov. **29**(3), 626–688 (2015)
2. Bailey, J.: Statistical measures for contrast patterns. In: Contrast Data Mining: Concepts, Algorithms, and Applications, pp. 13–20. CRC Press (2013)

3. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28645-5_29
4. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling databases. In: Proceedings of 7th International Conference Discovery Science, DS 2004, 2–5 October 2004, Padova, Italy, pp. 278–289 (2004)
5. He, Z., Xu, X., Huang, J.Z., Deng, S.: FP-outlier: frequent pattern based outlier detection. Comput. Sci. Inf. Syst. **2**(1), 103–118 (2005)
6. Koh, Y.S.: CD-TDS: change detection in transactional data streams for frequent pattern mining. In: 2016 International Joint Conference on Neural Networks, IJCNN 2016, 24–29 July 2016, Vancouver, BC, Canada, pp. 1554–1561 (2016)
7. Koufakou, A., Secretan, J., Georgiopoulos, M.: Non-derivable itemsets for fast outlier detection in large high-dimensional categorical data. Knowl. Inf. Syst. **29**(3), 697–725 (2011)
8. van Leeuwen, M., Siebes, A.: Streamkrimp: detecting change in data streams. In: Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (Part I), ECML/PKDD 2008, 15–19 September 2008, Antwerp, Belgium, pp. 672–687 (2008)
9. Loglisci, C., Ceci, M., Impedovo, A., Malerba, D.: Mining microscopic and macroscopic changes in network data streams. Knowl. Based Syst. **161**, 294–312 (2018)
10. Padillo, F., Luna, J.M., Ventura, S.: Subgroup discovery on big data: pruning the search space on exhaustive search algorithms. In: 2016 IEEE International Conference on Big Data, BigData 2016, 5–8 December 2016, Washington DC, USA, pp. 1814–1823 (2016)
11. Trabold, D., Horváth, T.: Mining strongly closed itemsets from data streams. In: Yamamoto, A., Kida, T., Uno, T., Kuboyama, T. (eds.) DS 2017. LNCS (LNAI), vol. 10558, pp. 251–266. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67786-6_18