A Sliding Window Algorithm for Relational Frequent Patterns Mining from Data Streams

Fabio Fumarola, Anna Ciampi, Annalisa Appice, Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari
via Orabona, 4 - 70126 Bari - Italy
{ffumarola, aciampi, appice, malerba}@di.uniba.it

Abstract. Some challenges in frequent pattern mining from data streams are the drift of data distribution and the computational efficiency. In this work an additional challenge is considered: data streams describe complex objects modeled by multiple database relations. A multi-relational data mining algorithm is proposed to efficiently discover approximate relational frequent patterns over a sliding time window of a complex data stream. The effectiveness of the method is proved on application to the Internet packet stream.

1 Introduction

A data stream is a sequence of time-stamped transactions which arrive on-line, at consecutive time points. The large volume of data continuously generated in short time and the change over time of statistical properties of data, make traditional data mining techniques unsuitable for data streams. The main challenges are avoiding multiple scans of the entire data sets, optimizing memory usage, and mining only the most recent patterns. In this work, we consider a further issue: the stream is a sequence of complex data elements, composed of several objects of various data types are someway related. For instance, network traffic in a LAN can be seen as a stream of connections, which have an inherent structure (e.g., the sequence of packets in the connection). The structure of complex data elements can be naturally modeled by means of multiple database relations and foreign key constraints (*(multi-)relational representation*). Therefore, we face a problem of *relational data stream mining*.

The task considered in this paper is frequent pattern mining. The proposed approach is based on the *sliding window model*, which completely discard stale data, thus saving memory storage and facilitating the detection of the distribution drift. This model is common to several algorithms for frequent pattern mining in data streams [9, 11, 6, 13]. However, all these algorithms work on a single database relation (*propositional representation*) and are not able to deal directly with complex data stored in multiple database relations.

Although it is possible to "propositionalize" relational data, i.e., transform them into a propositional form by building features which capture relational properties of data, this transformation can cause information loss. Multi-relational

data mining (MRDM) algorithms [4], which can navigate the relational structure in its original format, generate potentially new forms of evidence (*relational patterns*), which are not readily available in a propositional representation [5]. Several MRDM systems allow frequent patterns mining. Two representative examples of the state-of-the-art are WARMR [3] and SPADA [7], which both represent relational data and domain (or background) knowledge *à la* Datalog [2]. However, these systems are not designed to efficiently process data streams and to capture the possible drift of data distribution.

In this work, we propose a novel MRDM algorithm, called SWARM (**S**liding **W**indow **A**lgorithm for **R**elational Pattern **M**ining), which discovers approximate frequent relational patterns over a sliding time window of a relational data stream. SWARM is a false positive oriented algorithm, i.e., it does not discover any false negative frequent pattern. The contributions of SWARM are threefold. First, the multi-relational approach to complex data stream mining. Second, the use of the SE-tree to efficiently store and retrieve relational patterns. Third, the efficient and accurate approximation of the support of the frequent patterns over the sliding time window.

The paper is organized as follows. Section 2 introduces some preliminary concepts. The algorithm is described in Section 3, while experiments on an Internet packet stream are reported in Section 4. Finally, conclusions are drawn.

2 Preliminary Concepts and Definitions

In this work, objects stored in distinct relations of a database D play different roles. We distinguish between the set S of *reference* (or target) objects, which are the main subject of analysis, and the sets R_k, $1 \leq k \leq M$, of *task-relevant* (non-target) objects, which are related to the former and can contribute to define the units of analysis. It has been proved that this "individual centered" representation has several computational advantages, both theoretical (e.g., PAC-learnability) and practical (efficient exploration of the search space) [1].

Henceforth, we adopt a logic framework for the representation of units of analysis, and we categorize predicates into three classes. The unary *key predicate* identifies the reference objects in S (e.g., connection in Example 1). Binary *structural predicates* either relate task-relevant objects (e.g., next) or relate reference objects with task-relevant objects (e.g., packet) in the same unit of analysis. *Property predicates* define the value taken by a property. They can be either binary, when the attribute represents a property of a single object (e.g., nation source), or ternary, when the attribute represents a property of a relationship between two objects (e.g., distance between consecutive packets).

Example 1. A unit of analysis formed by a connection c (reference object) and a sequence of packets $p1$, $p2$, ... (task-relevant objects) is reported below:

connection(c), time(c,12:05), sourceNation(c, japan), ...,packet(c,p1), time(p1,12:05), number(p1,1), packet(c,p2), time(p2,12:06), number(p2,2), next(p1,p2), distance(p1,p2,1), packet(c,p2), ...

A relational pattern is a set of atoms (atomset). An atom is a predicate applied to a tuple of terms (variables or constants). Variables denote objects in S or some R_k, while constants denote values of property predicates.

Definition 1 (Relational pattern). *A relational pattern P is a set of atoms*
$$p_0(t_0), \{p_i(t_{i_1}, t_{i_2})\}_{i=0,\ldots,n}, \{p_j(t_{j_1}, t_{j_2}, t_{j_3})\}_{j=0,\ldots,m}$$
where p_0 is the key predicate, p_i $(i = 0, \ldots, n)$ are either structural predicates or binary property predicates, p_j $(j = 0, \ldots, m)$ are ternary property predicates.

Example 2. A relational pattern is reported below:
"*connection(C), packet(C,P), number(P,4), next(P,Q), distance(P,Q,3ms), number(Q,2), next (Q,R)*".

The *support* of a relational pattern P, denoted as $sup(P|_D)$, is the percentage of units of analysis in D "covered" (i.e., logically entailed) by P. P is frequent if $sup(P)$ is greater than a user-defined threshold σ.

Following the sliding window model, the units of analysis in D depend on a time-sensitive sliding window.

Definition 2 (Time-sensitive sliding-window). *Given a time point p, the set of units of analysis arriving in the period $[t - p + 1, t]$ forms a* slide *B. Let B_i be the i-th slide, the time-sensitive sliding-window W_i associated with B_i is the set of w consecutive slides from B_{i-w+1} to B_i.*

The window moves forward by a certain amount of unit of analysis by adding the new slide (B_{i+1}) and dropping the expired one (B_{i-w+1}). The number of units of analysis that are added to (and removed from) each window is $|B_i|$. We assume that a unit of analysis is associated with a timestamp and data elements forming a single unit of analysis flow in the stream at the same time.

3 The Algorithm

A buffer continuously consumes the stream units of analysis and pours them slide-by-slide into SWARM system. After a slide goes through SWARM, it is discarded. SWARM operations consist of discovering relational patterns over a slide, maintaining relational patterns over a window and approximating frequent relational patterns over a window. Input parameters are: the minimum support threshold σ, the maximum support error ϵ ($\epsilon < \sigma$), the period p of a slide, the number w of slides in a window, and the maximum depth $MaxDepth$ of patterns.

3.1 Relational Pattern Discovery over a Slide

Once a slide flows in the buffer, relational patterns are locally discovered by exploring the lattice of relational patterns ordered according to a generality order ($\geq$). This generality order is based on θ-subsumption [10] and is monotonic with respect to support. The search proceeds in a Set Enumerated tree (SE-tree)

search framework [12], starting from the most general pattern (the one with only the key predicate), and iteratively alternating the candidate generation and candidate evaluation as in the level-wise method [8]. The SE-tree search framework has several advantages. First, the SE-tree enumerates all possible patterns by allowing a complete search. Second, it prevents the generation and evaluation of candidates which are equivalent under θ-subsumption. Third, it effectively exploits the monotonicity property of $\geq$ to prune the search space.

A node of the SE-tree is associated with a progressive natural index and it is represented by the *head* and the *tail*. The head of the root is the pattern that contains only the key predicate. The tail is the ordered set of atoms which may be appended to the head by the downward refinement operator ρ.

Definition 3 (Downward refinement operator). *Let P be a relational pattern. Then $\rho(P) = \{P \cup \{p(\ldots)\} | p$ is either a structural predicate or a property predicate that shares at least one argument with one of the atoms in $P\}$.*

Let $n[head, tail]$ be a node of the SE-tree and $q(\ldots)$ be an atom in $tail(n)$. Then n has a child $n_q[head, tail]$ whose head is defined as follows:

$$head(n_q) = head(n) \cup q(\ldots). \tag{1}$$

If q is based on a property predicate, its tail is defined as follows:

$$tail(n_q) = \Pi_{>q} tail(n) \tag{2}$$

where $\Pi_{>q} tail(n)$ is the order set of atoms stored after q in $tail(n)$. Differently, if q is based on a structural predicate, its tail is defined as follows:

$$tail(n_q) = \Pi_{>q} tail(n) \cup \{r(\ldots)\} \tag{3}$$

where $\{r(\ldots)\}$ is a set of atoms $r(\ldots)$. Each $r(\ldots)$ is an atom that belongs to one of the refinement $\rho(head(n_q))$ under the conditions that $r(\ldots)$ shares variables with $q(\ldots)$ and $r(\ldots)$ is not included in $tail(n)$. When $r(\ldots)$ is based on a structural predicate, one of its arguments must be a new variable.

The monotonicity property of $\geq$ with respect to support makes the expansion of infrequent nodes (i.e., nodes whose local support is less than ϵ) useless. In addition, we prevent the expansion of nodes at a depth greater than $MaxDepth$.

3.2 Relational Pattern Maintenance over a Window

Distinct sets of relational patterns are discovered for each slide. The naive solution is to keep in memory a distinct SE-tree for each slide of the window. This would lead to enumerate several times relational patterns which are discovered in distinct slides. To reduce memory usage, a single SE-tree is maintained on the window. At this aim, each node n of the SE-tree maintains a w sized sliding vector $\boldsymbol{sv}(n)$, which stores one support for each slide in the window. By default, the local support values which are stored in $\boldsymbol{sv}(n)$ are set to unknown. According

to the sliding model when a new slide flows in the buffer, the support vector is shifted on the left in order to remove the expired support. In this way, only the last w support values are maintained in the nodes of the SE-tree.

The maintenance of the SE-tree proceeds as follows. When a relational pattern P_n is discovered over a slide B, we distinguish between two cases, namely, P_n is enumerated in the SE-tree or not. In the former case, the SE-tree is expanded with the new node n which enumerates P_n, while in the latter case the node n already exists in the SE-tree and $\boldsymbol{sv}(n)$ is shifted on the left. In both cases, the value of support $sup(P_n|_B)$ is computed over B and is then stored in the last position of $\boldsymbol{sv}(n)$. Finally, nodes are pruned when they enumerate relational patterns which are unknown on each slide of the window.

3.3 Relational Frequent Pattern Approximation over a Window

A relational pattern P_n is identified as approximately frequent over W iff the approximate support $sup_A(P_n|_W)$ estimated over W is greater than σ. The approximate support of P_n is computed on the basis of the local support values which are stored in $\boldsymbol{sv}(n)$.

$$sup_A(P_n|_W) = \sum_{i=1}^{w} (\boldsymbol{sv}(n)[i] \times |B_i|) / \sum_{i=1}^{w} |B_i| \tag{4}$$

When the local support $\boldsymbol{sv}(n)[i]$ is unknown over a slide B_i, it is estimated by using the known support of an ancestor of P_n. In particular, the pattern Q_m is found such that Q_m is the most specific ancestor of P_n in the SE-tree with a known support value over B_i. Theoretically, the complete set of at worst $2^k - 1$ ancestors should be explored, where k denotes the pattern length. This solution may be impractical for high value of k. To improve efficiency, only the ancestors along the path from n to the root are truly explored. This way, the time complexity of this search is $\mathrm{O}(k)$.

Since the SE-tree enumerates patterns discovered by using the maximum support error ϵ as support threshold, Q_m can either be *infrequent* ($sup(Q_m|_{B_i}) < \epsilon$), or *sub-frequent* ($\epsilon \leq sup(Q_m|_{B_i}) < \sigma$) or *frequent* ($sup(Q_m|_{B_i}) \geq \sigma$). In the first case, the support of Q_m is used to estimate the support of P_n. In the other two cases the support of P_n is correctly determined as zero. Indeed, the fact that a pattern is refined until it is not *infrequent*, except when ρ refinements of a pattern have zero valued support over the slide, and the monotonicity property of $\geq$ with respect to support, ensure that $sup(P_n|_{B_i}) = 0$.

4 Experiments

We evaluate SWARM on a real Internet packet stream that was logged by the firewall of our Department, from June 1st till June 28th, 2004. This stream consists of 380,733 ingoing connections for a total of 651,037 packets. A connection is described by means of six properties (e.g. service, protocol, ...). A packet is

described by means of the order of arrival of the packet within the connection. This order of arrival allows us to represent a relationship of sequentiality between 270,304 pairs of consecutive packets. The time distance between two packets is a property of each pair of consecutive packets. The stream is segmented in time slides with a period p and approximate relational frequent patterns are discovered on sliding windows covering w consecutive slides. Experiments are run by varying p ($p = 30, 60$ minutes), w ($w = 6h/p, 12h/p, 18h/p$) and ϵ ($\epsilon = 0.5, 0.7$). σ is set to 0.7 and $MaxDepth$ is set to 8.

Relational patterns discovered by SWARM are compared with relational patterns discovered by a multi-relational implementation that we have done of the algorithm SW [6]. Initially, we analyze the total number of false positive patterns which are discovered over the sliding windows of the entire stream. No false negative pattern is discovered by both SWARM and SW due to the overestimation of the support. The number of false positive patterns is reported in Table 1. False positive are those approximate patterns which are not included in the set of true frequent patterns we have directly discovered over the entire windows. These results confirm that SWARM discovers a lower number of false positive than SW by providing a more significant approximation of support when local support values are unknown. Additionally, the number of false positive patterns is significantly lower when sub-frequent candidates ($\epsilon = 0.5 < \sigma = 0.7$) are locally generated at slide level. As expected, the number of false positive patterns increases by enlarging the window size and/or reducing the slide period.

Table 1. The total number of false positive patterns discovered on the entire stream: comparison between SWARM and SW. $\sigma = 0.7$.

Experimental Setting	SWARM $\sigma = 0.7$ $\epsilon = 0.5$	SWARM $\sigma = \epsilon = 0.7$	SW
$p = 30$ min $w = 12$	1	42	145
$p = 30$ min $w = 24$	6	68	203
$p = 30$ min $w = 36$	8	42	240
$p = 60$ min $w = 6$	0	27	68
$p = 60$ min $w = 12$	3	28	104
$p = 60$ min $w = 18$	3	23	135

Further considerations are suggested by the analysis of the absolute error of the approximated support, averaged over the true positive patterns. Only the sliding windows where the error is greater than zero are plotted in Figure 1. Due to space limitations, the plot concerns only the parameter setting $p = 30, 60$ minutes and $w = 12h/p$, but the considerations we report below can be extended to other settings we tried. We observe that SWARM always exhibits a lower error rate than SW. Additionally, the discovery of sub-frequent local patterns ($\epsilon < \sigma$) makes more accurate the approximation of the support.

A different perspective of the results is offered by the comparison of the relational patterns discovered by both SWARM and SW over the sliding windows that cover the same portion of the data stream, but are generated with different

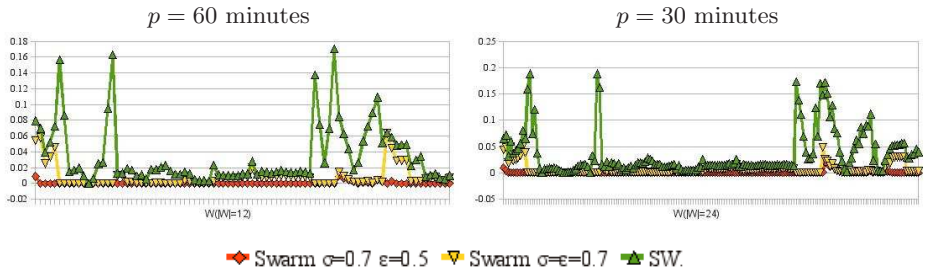

Fig. 1. Average absolute error rate: SWARM with $\epsilon = 0.5$ and $\epsilon = 0.7$ vs. SW.

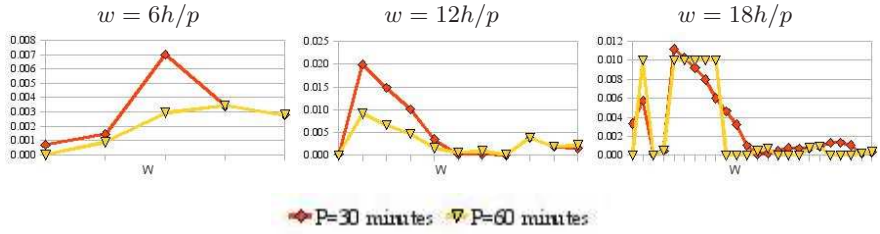

Fig. 2. Average absolute error rate of SWARM ($\theta = 0.7$ and $\epsilon = 0.5$): $p = 60$ minutes vs. $p = 30$ minutes.

slide period. Although the same number of false positive patterns is discovered independently from the slide period, some differences are observed in the error rate plotted in Figure 2. The general trend is that the error decreases by enlarging the period of a slide. Few exceptions are observed with greater values of w.

Statistics on the elapsed time are shown in Figure 3. The discovery of approximate frequent patterns on a slide-by-slide basis is more efficient than the discovery of exact frequent patterns on the entire window. As expected, elapsed time decreases by reducing the slide period.

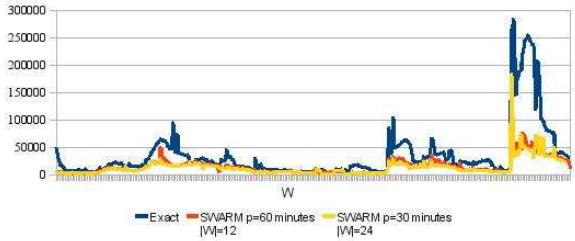

Fig. 3. Elapsed time: discovering approximate frequent patterns on a slide-by-slide basis vs. discovering exact frequent patterns on the entire window.

5 Conclusions

We present a novel multi-relational data mining algorithm for approximate frequent relational pattern discovery over sliding time windows of a data stream. The algorithm is evaluated in a real Internet packet stream. Experiments prove that our algorithm is both accurate and efficient. In a future work, we intend to investigate the quality of the approximation the unknown local support of a pattern when it is based on *all* ancestors of the pattern and not only the most specific ancestor along the path to the top of SE-tree.

6 Acknowledgments

This work is supported by both the Project "Scoperta di conoscenza in domini relazionali" funded by the University of Bari and the Strategic Project PS121 "Telecommunication Facilities and Wireless Sensor Networks in Emergency Management" funded by Apulia Region.

References

1. H. Blockeel and M. Sebag. Scalability and efficiency in multi-relational data mining. *SIGKDD Explorations*, 5(1):17–30, 2003.
2. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, New York, NY, 1990.
3. L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *the 7th International Workshop on Inductive Logic Programming, ILP 1997*, volume 1297, pages 125–132. Springer-Verlag, 1997.
4. S. Džeroski and N. Lavrač. *Relational Data Mining*. Springer-Verlag, 2001.
5. S. Kramer. *Relational Learning vs. Propositionalization: Investigations in Inductive Logic Programming and Propositional Machine Learning*. PhD thesis, 1999.
6. C. Lin, D. Chiu, and Y. Wu. Mining frequent itemsets from data streams with a time-sensitive sliding window. In *Proc. of the SIAM Int. Data Mining Conf.*, 2005.
7. F. A. Lisi and D. Malerba. Inducing multi-level association rules from multiple relations. *Machine Learning*, 55(2):175–210, 2004.
8. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
9. B. Mozafari, H. Thakkar, and C. Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In *Proc. Int. Conf. on Data Engineering*, pages 179–188, Los Alamitos, CA, 2008. IEEE Computer Society.
10. G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.
11. J. Ren and K. Li. Find recent frequent items with sliding windows in data streams. In *Proc. 3rd Int. Conf. on Information Hiding and Multimedia Signal Processing*, pages 625–628. IEEE Computer Society, 2007.
12. R. Rymon. An SE-tree based characterization of the induction problem. In *Proc. Int. Conf on Machine Learning*, pages 268–275. Morgan Kaufmann, 1993.
13. C. Silvestri and S. Orlando. Approximate mining of frequent patterns on streams. *Intelligent Data Analysis*, 11(1):49–73, 2007.