Mining Constrained Cross-Graph Cliques in Dynamic Networks

Loïc Cerf, Bao Tran Nhan Nguyen, and Jean-François Boulicaut

Abstract Three algorithms — CubeMiner, Trias, and Data-Peeler — have been recently proposed to mine closed patterns in ternary relations, i.e., a generalization of the so-called formal concept extraction from binary relations. In this paper, we consider the specific context where a ternary relation denotes the value of a graph adjacency matrix (i. e., a Vertices $\times$ Vertices matrix) at different timestamps. We discuss the constraint-based extraction of patterns in such dynamic graphs. We formalize the concept of δ-contiguous closed 3-clique and we discuss the availability of a complete algorithm for mining them. It is based on a specialization of the enumeration strategy implemented in Data-Peeler. Indeed, the relevant cliques are specified by means of a conjunction of constraints which can be efficiently exploited. The added-value of our strategy for computing constrained clique patterns is assessed on a real dataset about a public bicycle renting system. The raw data encode the relationships between the renting stations during one year. The extracted δ-contiguous closed 3-cliques are shown to be consistent with our knowledge on the considered city.

Loïc Cerf
Université de Lyon, CNRS, INRIA
INSA-Lyon, LIRIS Combining, UMR5205, F-69621, France
e-mail: lcerf@liris.cnrs.fr

Bao Tran Nhan Nguyen
Université de Lyon, CNRS, INRIA
INSA-Lyon, LIRIS Combining, UMR5205, F-69621, France
e-mail: baonhan@pmail.ntu.edu.sg

Jean-François Boulicaut
Université de Lyon, CNRS, INRIA
INSA-Lyon, LIRIS Combining, UMR5205, F-69621, France
e-mail: jboulica@liris.cnrs.fr

1 Introduction

Mining binary relations (often encoded as Boolean matrices) has been intensively studied. For instance, a popular application domain concerns basket data analysis and mining tasks on *Transactions* × *Products* relations. In a more general setting, binary relations may denote relationships between objects and a given set of properties giving *Objects* × *Properties* matrices. Many knowledge discovery processes from potentially large binary relations have been considered. We are interested in descriptive approaches that can be based on pattern discovery methods. Pattern types can be frequent itemsets (see, e. g., [1, 22]), closed itemsets or formal concepts (see, e. g., [15, 25, 5]), association rules (see, e. g., [2]) or their generalizations like, e. g., [3]. Interestingly, when looking at the binary relation as the encoding of a bi-partite graph (resp. a graph represented by its adjacency matrix), some of these patterns can be interpreted in terms of graph substructures. A typical example that is discussed in this chapter concerns the analogy between formal concepts and maximal bi-cliques (resp. cliques).

Constraint-based mining is a popular framework for supporting relevant pattern discovery thanks to user-defined constraints (see, e. g., [6]). It provides more interesting patterns when the analyst specifies his/her subjective interestingness by means of a combination of primitive constraints. This is also known as a key issue to achieve efficiency and tractability. Some constraints can be deeply pushed into the extraction process such that it is possible to get complete (every pattern which satisfies the user-defined constraint is computed) though efficient algorithms. As a result, many efficient algorithms are available for computing constrained patterns from binary relations. Among others, this concerns constraint-based mining of closed patterns from binary relations (see, e. g., [23, 28, 4, 25, 26]).

It is clear that many datasets of interest correspond to n-ary relations where $n \geq 3$. For instance, a common situation is that space and time information are available such that we get the generic setting of *Objects* × *Properties* × *Dates* × *Places* 4-ary relations. In this chapter, we consider the encoding of *dynamic graphs* in terms of collections of adjacency matrices, hence a ternary relation *Vertices* × *Vertices* × *Date*. The discovery of closed patterns from ternary relations has been recently studied. From a semantics perspective, such patterns are a straightforward extension of formal concepts. Computing them is however much harder. To the best of our knowledge, the extension towards higher arity relations has given rise to three proposals, namely CubeMiner [17] or Trias [16] for ternary relations, and Data-Peeler for arbitrary n-ary relations [10, 11]. A major challenge is then to exploit user-defined constraints during the search of application relevant closed patterns. We assume that the state-of-the-art approach is the Data-Peeler enumeration strategy which can mine closed patterns under a large class of constraints called *piecewise (anti)-monotone constraints* [11].

In this chapter, we consider that data (i. e., a ternary relation) denote a dynamic graph. We assume that the encoded graphs have a fixed set of vertices and that directed links can appear and/or disappear at the different timestamps. Furthermore, we focus on clique patterns which are preserved along almost-contiguous timestamps. For instance, it will provide interesting hypothesis about sub-networks of stations within a bicycle renting system.

We have three related objectives.

- First, we want to illustrate the genericity of the Data-Peeler algorithm. We show that relevant pattern types can be specified as closed patterns that further satisfy other user-defined constraints in the ternary relation that denotes the dynamic graph. We study precisely the pattern type of δ-contiguous closed 3-cliques, i. e., maximal sets of vertices that are linked to each other and that run along some "almost" contiguous timestamps. To denote a clique pattern, a closed pattern will have to involve identical sets of vertices (using the so-called symmetry constraint). Notice that being a closed pattern will be also expressed in terms of two primitive constraints (namely the connection and closedness constraints) that are efficiently processed by the Data-Peeler enumeration. We do not provide all the details about the algorithm (see [11] for an in-depth presentation) but its most important characteristics are summarized and we formalize the constraint properties that it can exploit efficiently. Doing so, we show that the quite generic framework of arbitrary n-ary relation mining can be used to support specific analysis tasks in dynamic graphs.
- Next, our second objective is to discuss the specialization of the algorithm to process more efficiently the conjunction of the connection, closedness, symmetry and contiguity constraints, i. e., what can be done to specialize the generic mechanisms targeted to closed pattern discovery from arbitrary n-ary relations when we are looking for preserved cliques in $\mathit{Vertices} \times \mathit{Vertices} \times \mathit{Date}$ ternary relations. This technical contribution enables to discuss efficiency issues and optimized constraint checking.
- Last but not the least, we show that this algorithmic contribution can be used in concrete applications. Graph mining is indeed a popular topic. Many researchers consider graph pattern discovery from large collections of graphs while others focus on data analysis techniques for one large graph. In the latter case, especially in the context of dynamic graphs, we observe two complementary directions of research. On one hand, global properties of such graphs are studied like power-law distribution of node degree or diameters (see, e. g., [20]). On another hand, it is possible to use pattern discovery techniques to identify local properties in the graphs (see, e. g., [27]). We definitively contribute to this later approach. We compute δ-contiguous closed 3-cliques in a real-life dynamic graph related to bicycle renting in a large European city. We illustrate that these usage patterns can be interpreted thanks to domain knowledge and that they provide a feedback on emerging sub-networks.

The rest of the paper is organized as follows. We formalize the mining task and we discuss the type of constraints our algorithm handles in Sect. 2. In Section 3, we summarize the fundamental mechanisms used in the DATA-PEELER algorithm. Section 4 details how the δ-contiguity constraint is enforced. Section 5 describes the strategy for computing closed 3-cliques by pushing various primitive constraints into the enumeration strategy. Section 6 provides an experimental validation on a real dataset. Related work is discussed in Sect. 7, and Sect. 8 briefly concludes.

2 Problem Setting

Let $\mathcal{T} \in \mathbb{R}^{|\mathcal{T}|}$ a finite set of timestamps. Let $\mathcal{N}$ a set of nodes. A (possibly directed) graph is uniquely defined by its adjacency matrix $A \in \{0,1\}^{\mathcal{N}\times\mathcal{N}}$. A dynamic graph involving the nodes of $\mathcal{N}$ along $\mathcal{T}$ is uniquely defined by the $|\mathcal{T}|$-tuple $(A_t)_{t\in\mathcal{T}}$ gathering the adjacency matrices of the graph at every timestamp $t \in \mathcal{T}$. Visually, such a stack of adjacency matrices can be seen as a $|\mathcal{T}| \times |\mathcal{N}| \times |\mathcal{N}|$ cube of 0/1 values. We write $a_{t,n^1,n^2} = 1$ (resp. $a_{t,n^1,n^2} = 0$) when, at the timestamp t, a link from n^1 to n^2 is present (resp. absent).

Example 1. Figure 1 depicts a dynamic directed graph involving four nodes a, b, c and d. Four snapshots of this graph are available at timestamps 0, 0.5, 2 and 3. Table 1 gives the related 4-tuple $(A_0, A_{0.5}, A_2, A_3)$.

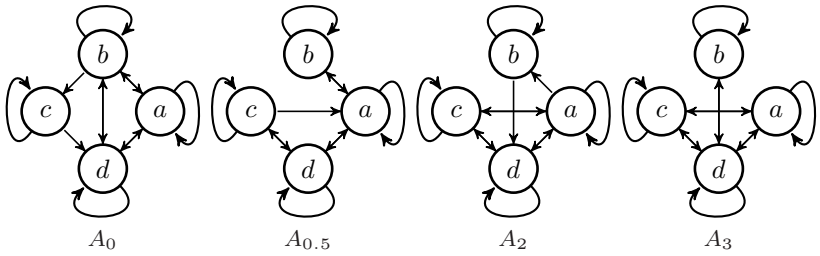

Fig. 1 Example of a dynamic directed graph ($\mathcal{N} = \{a, b, c, d\}$, $\mathcal{T} = \{0, 0.5, 2, 3\}$)

Visually, a closed 3-set $(T, N^1, N^2) \in 2^{\mathcal{T}} \times 2^{\mathcal{N}} \times 2^{\mathcal{N}}$ appears as a combinatorial sub-cube of the data (modulo arbitrary permutations on any dimension) satisfying both the *connection* and the *closedness* primitive constraints. Informally, it means that $T \times N^1 \times N^2$ only contains '1' values (connection), and any "super-cube" of (T, N^1, N^2) violates the connection constraint (closedness). Let us define them more formally.

Definition 1 ($\mathcal{C}_{\text{connected}}$). A 3-set (T, N^1, N^2) is said connected, denoted $\mathcal{C}_{\text{connected}}(T, N^1, N^2)$, iff $\forall (t, n^1, n^2) \in T \times N^1 \times N^2, a_{t,n^1,n^2} = 1$.

Table 1 $(A_0, A_{0.5}, A_2, A_3)$ related to the dynamic graph depicted Fig. 1

	A_0				$A_{0.5}$				A_2				A_3			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
a	1	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1
b	1	1	1	1	1	1	0	0	0	1	0	1	0	1	0	1
c	0	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
d	1	1	0	1	1	0	1	1	1	0	1	1	1	1	1	1

Definition 2 ($\mathcal{C}_{\text{closed}}$). It is said that a 3-set (T, N^1, N^2) is closed, denoted $\mathcal{C}_{\text{closed}}(T, N^1, N^2)$, iff $\begin{cases} \forall t \in \mathcal{T} \setminus T, \neg\mathcal{C}_{\text{connected}}(\{t\}, N^1, N^2) \\ \forall n^1 \in \mathcal{N} \setminus N^1, \neg\mathcal{C}_{\text{connected}}(T, \{n^1\}, N^2) \\ \forall n^2 \in \mathcal{N} \setminus N^2, \neg\mathcal{C}_{\text{connected}}(T, N^1, \{n^2\}) \end{cases}$.

A closed 3-set can now be formally defined.

Definition 3 (Closed 3-set). (T, N^1, N^2) is a closed 3-set iff it satisfies the conjunction $\mathcal{C}_{\text{connected}}(T, N^1, N^2) \wedge \mathcal{C}_{\text{closed}}(T, N^1, N^2)$.

Example 2. $(\{0, 2, 3\}, \{a, b, c, d\}, \{d\})$ is a closed 3-set in the toy dataset from Table 1: $\forall (t, n^1, n^2) \in \{0, 2, 3\} \times \{a, b, c, d\} \times \{d\}$, we have $a_{t,n^1,n^2} = 1$, and $\begin{cases} \forall t \in \{0.5\}, \neg\mathcal{C}_{\text{connected}}(\{t\}, \{a, b, c, d\}, \{d\}) \\ \forall n^1 \in \emptyset, \neg\mathcal{C}_{\text{connected}}(\{0, 2, 3\}, \{n^1\}, \{d\}) \\ \forall n^2 \in \{a, b, c\}, \neg\mathcal{C}_{\text{connected}}(\{0, 2, 3\}, \{a, b, c, d\}, \{n^2\}) \end{cases}$.
$(\{2, 3\}, \{a, c, d\}, \{a, c, d\})$ and $(\{0, 3\}, \{b, d\}, \{b, d\})$ are two other closed 3-sets. $(\{0.5, 2, 3\}, \{c, d\}, \{c, d\})$ is not a closed 3-set because it violates $\mathcal{C}_{\text{closed}}$. Indeed $\mathcal{C}_{\text{connected}}(\{0.5, 2, 3\}, \{c, d\}, \{a\})$ holds, i. e., the third set of the pattern can be extended with a.

Given $\delta \in \mathbb{R}_+$, a δ-contiguous 3-set is such that it is possible to browse the whole subset of timestamps by jumps from one timestamp to another without exceeding a delay of δ for each of these jumps.

Definition 4 (δ-contiguity). A 3-set (T, N^1, N^2) is said δ-contiguous, denoted $\mathcal{C}_{\delta\text{-contiguous}}(T, N^1, N^2)$, iff $\forall t \in [\min(T), \max(T)], \exists t' \in T$ s.t. $|t - t'| \leq \delta$.

Notice that t does not necessarily belong to $\mathcal{T}$ (if $|T| \geq 2$, $[\min(T), \max(T)]$ is infinite). $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}}$ being stronger than $\mathcal{C}_{\text{connected}}$ alone, a related and weaker closedness constraint can be defined. Intuitively, a δ-closed 3-set is closed w.r.t. both $\mathcal{N}$ sets and to the timestamps of $\mathcal{T}$ in the vicinity of those inside the 3-set. Hence, a timestamp that is too far away (delay exceeding δ) from any timestamp inside the 3-set, cannot prevent its δ-closedness.

Definition 5 (δ-closedness). It is said that a 3-set (T, N^1, N^2) is δ-closed, denoted $\mathcal{C}_{\delta\text{-closed}}(T, N^1, N^2)$, iff
$$\begin{cases} \forall t \in \mathcal{T} \setminus T, (\exists t' \in T \text{ s. t. } |t - t'| \leq \delta \Rightarrow \neg\mathcal{C}_{\text{connected}}(\{t\}, N^1, N^2)) \\ \forall n^1 \in \mathcal{N} \setminus N^1, \neg\mathcal{C}_{\text{connected}}(T, \{n^1\}, N^2) \\ \forall n^2 \in \mathcal{N} \setminus N^2, \neg\mathcal{C}_{\text{connected}}(T, N^1, \{n^2\}) \end{cases}.$$

Definition 6 (δ-contiguous closed 3-set). (T, N^1, N^2) is a δ-contiguous closed 3-set iff it satisfies the conjunction $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}} \wedge \mathcal{C}_{\delta\text{-closed}}$.

A δ-contiguous closed 3-set is an obvious generalization of a closed 3-set. Indeed, $\forall \delta \geq \max(\mathcal{T}) - \min(\mathcal{T}), \begin{cases} \mathcal{C}_{\delta\text{-contiguous}} \equiv true \\ \mathcal{C}_{\delta\text{-closed}} \equiv \mathcal{C}_{\text{closed}} \end{cases}$.

Example 3. $(\{2, 3\}, \{a, b, c, d\}, \{d\})$ is a 1.75-contiguous closed 3-set in the toy dataset from Table 1. However, it is neither 0.5-contiguous (the timestamps 2 and 3 are not close enough) nor 2-closed (0 can extend the set of timestamps). This illustrates the fact that the number of δ-contiguous closed 3-sets is not monotone in δ.

We want to extract sets of nodes that are entirely interconnected. In this context, a 3-set (T, N^1, N^2) where $N^1 \neq N^2$ is irrelevant and a symmetry constraint must be added.

Definition 7 (Symmetry). A 3-set (T, N^1, N^2) is said symmetric, denoted $\mathcal{C}_{\text{symmetric}}(T, N^1, N^2)$, iff $N^1 = N^2$.

Again, let us observe that $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}} \wedge \mathcal{C}_{\text{symmetric}}$ being stronger than $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}}$, a related and weaker closedness constraint can be defined. Intuitively, if not *both* the row and the column pertaining to a node n can *simultaneously* extend a 3-set without breaking $\mathcal{C}_{\text{connected}}$, the closedness is not violated.

Definition 8 (Symmetric δ-closedness). It is said that a 3-set (T, N^1, N^2) is symmetric δ-closed, denoted $\mathcal{C}_{\text{sym-}\delta\text{-closed}}(T, N^1, N^2)$, iff
$$\begin{cases} \forall t \in \mathcal{T} \setminus T, (\exists t' \in T \text{ s. t. } |t - t'| \leq \delta \Rightarrow \neg\mathcal{C}_{\text{connected}}(\{t\}, N^1, N^2)) \\ \forall n \in \mathcal{N} \setminus (N^1 \cap N^2), \neg\mathcal{C}_{\text{connected}}(T, N^1 \cup \{n\}, N^2 \cup \{n\}) \end{cases}.$$

Definition 9 (δ-contiguous closed 3-clique). It is said that (T, N^1, N^2) is a δ-contiguous closed 3-clique iff it satisfies $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}} \wedge \mathcal{C}_{\text{symmetric}} \wedge \mathcal{C}_{\text{sym-}\delta\text{-closed}}$.

Example 4. Two out of the three closed 3-sets illustrating Ex. 2 are symmetric: $(\{2, 3\}, \{a, c, d\}, \{a, c, d\})$ and $(\{0, 3\}, \{b, d\}, \{b, d\})$. In Ex. 2, it was shown that $(\{0.5, 2, 3\}, \{c, d\}, \{c, d\})$ is not closed w.r.t. $\mathcal{C}_{\text{closed}}$. However it is symmetric 1.75-closed. Indeed, the node a cannot simultaneously extend its second and third sets of elements without violating $\mathcal{C}_{\text{connected}}$.

Problem Setting

Assume $(A_t)_{t\in\mathcal{T}} \in \{0,1\}^{\mathcal{T}\times\mathcal{N}\times\mathcal{N}}$ and $\delta \in \mathbb{R}_+$. This chapter deals with computing the complete collection of the δ-contiguous closed 3-cliques which hold in this data. In other terms, we want to compute every 3-set which satisfies the conjunction of the four primitive constraints defined above, i. e., $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}} \wedge \mathcal{C}_{\text{symmetric}} \wedge \mathcal{C}_{\text{sym-}\delta\text{-closed}}$. In practical settings, such a collection is huge. It makes sense to constrain further the extraction tasks (i. e., to also enforce a new user-defined constraint $\mathcal{C}$) to take subjective interestingness into account and to support the focus on more relevant cliques. Thus, the problem becomes the complete extraction of the δ-contiguous closed 3-cliques satisfying $\mathcal{C}$.

Instead of writing, from scratch, an ad-hoc algorithm for computing constrained δ-contiguous closed 3-cliques, let us first specialize the generic closed n-set extractor Data-Peeler [10, 11]. Its principles and the class of constraints it can exploit are stated in the next section. In Sect. 4, we study its adaptation to δ-contiguous closed 3-set mining, and Sect. 5 presents how to force the closed 3-sets to be symmetric and symmetric δ-closed.

3 Data-Peeler

3.1 Traversing the Search Space

Data-Peeler [11] aims to extract a complete collection of constrained closed n-sets from an n-ary relation. This section only outlines the basic principles for enumerating the candidates in the particular case $n = 3$. The interested reader would refer to [11] for detailed explanations. To emphasize the generality of Data-Peeler, the three sets $\mathcal{T}$, $\mathcal{N}$ and $\mathcal{N}$ are, here, replaced by $\mathcal{D}^1$, $\mathcal{D}^2$ and $\mathcal{D}^3$. Indeed, when extracting closed 3-sets, there is no need for $\mathcal{D}^1$ to contain real numbers, and for $\mathcal{D}^2$ and $\mathcal{D}^3$ to be identical. These three sets must only be finite.

Like many complete algorithms for local pattern detection, Data-Peeler is based on enumerating candidates in a way that can be represented by a binary tree where:

- at every node, an element e is enumerated;
- every pattern extracted from the left child *does contain* e;
- every pattern extracted from the right child does *not* contain e.

This division of the extraction into two sub-problems partitions the search space, i. e., the union of the closed 3-sets found in both enumeration sub-tree are exactly the closed 3-sets to be extracted from the parent node (*correctness*) *and* each of these closed 3-sets is found only once (*uniqueness*). In the

case of Data-Peeler, the enumerated element e can always be freely chosen among all the elements (from all three sets $\mathcal{D}^1$, $\mathcal{D}^2$ and $\mathcal{D}^3$) remaining in the search space.

Three 3-sets $U = (U^1, U^2, U^3)$, $V = (V^1, V^2, V^3)$ and $\mathcal{S} = (\mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^3)$, are attached to every node. The 3-set $U \in 2^{\mathcal{D}^1} \times 2^{\mathcal{D}^2} \times 2^{\mathcal{D}^3}$ contains the elements that are contained in any closed 3-set extracted from the node. The 3-set $V \in 2^{\mathcal{D}^1} \times 2^{\mathcal{D}^2} \times 2^{\mathcal{D}^3}$ contains the elements that may be present in the closed 3-sets extracted from the node, i. e., the search space. The 3-set $\mathcal{S} \in 2^{\mathcal{D}^1} \times 2^{\mathcal{D}^2} \times 2^{\mathcal{D}^3}$ contains the elements that may prevent the 3-sets, extracted from this node, from being closed. To simplify the notations we will often assimilate a 3-set (S^1, S^2, S^3) with $S^1 \cup S^2 \cup S^3$. For example, given two 3-set $A = (A^1, A^2, A^3)$ and $B = (B^1, B^2, B^3)$ and an element e ($e \in \mathcal{D}^1 \cup \mathcal{D}^2 \cup \mathcal{D}^3$), we write:

- $e \in A$ instead of $e \in A^1 \cup A^2 \cup A^3$
- $A \setminus \{e\}$ instead of $\begin{cases} (A^1 \setminus \{e\}, A^2, A^3) \text{ if } e \in \mathcal{D}^1 \\ (A^1, A^2 \setminus \{e\}, A^3) \text{ if } e \in \mathcal{D}^2 \\ (A^1, A^2, A^3 \setminus \{e\}) \text{ if } e \in \mathcal{D}^3 \end{cases}$
- $A \cup B$ instead of $(A^1 \cup B^1, A^2 \cup B^2, A^3 \cup B^3)$

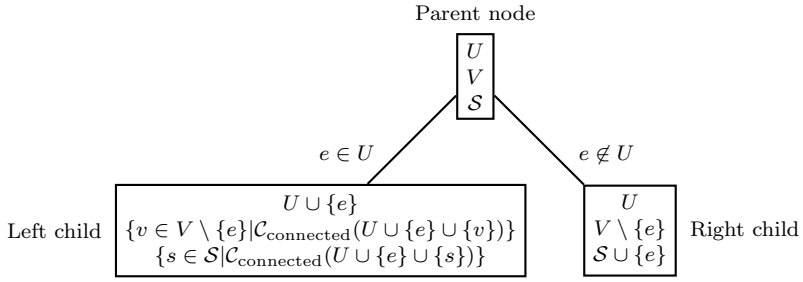

Fig. 2 Enumeration of any element $e \in V$

Figure 2 depicts the enumeration. The 3-sets attached to a child node are computed from its parent's analogous 3-sets, the enumerated element and the data (for the left children only). In particular, in the left child, Data-Peeler ensures that U can receive any element from V without breaking $\mathcal{C}_{\text{connected}}$. Hence, at every node, the 3-set U is connected, i. e., $\mathcal{C}_{\text{connected}}(U)$. To ensure that the extracted 3-sets are closed, Data-Peeler checks, at every node, whether the 3-set $U \cup V$ is closed, i. e., $\mathcal{C}_{\text{closed}}(U \cup V)$. To do so, Data-Peeler checks whether $\forall s \in \mathcal{S}, \neg\mathcal{C}_{\text{connected}}(U \cup V \cup \{s\})$. If not, every 3-set descendant from this node is not closed. Indeed, $\forall V' \subseteq V, \exists s \in \mathcal{S} | \mathcal{C}_{\text{connected}}(U \cup V \cup \{s\}) \Rightarrow \exists s \in \mathcal{S} | \mathcal{C}_{\text{connected}}(U \cup V' \cup \{s\})$. In this case Data-Peeler safely prunes the sub-tree rooted by the node.

The enumeration tree is traversed in a depth first way. At the root node, $U = (\emptyset, \emptyset, \emptyset)$, $V = (\mathcal{D}^1, \mathcal{D}^2, \mathcal{D}^3)$ ans $\mathcal{S} = (\emptyset, \emptyset, \emptyset)$. At a given node, if $V =$

$(\emptyset, \emptyset, \emptyset)$ then this node is a leaf and U is a closed 3-set. Algorithm 1 sums up Data-Peeler's principles.

Input: $U, V, \mathcal{S}$
Output: All closed 3-sets containing the elements in U and, possibly, some elements in V and satisfying $\mathcal{C}$
if $\mathcal{C}$ may be satisfied by a 3-set descending from this node $\wedge \mathcal{C}_{\text{closed}}(U \cup V)$ **then**
  if $V = (\emptyset, \emptyset, \emptyset)$ **then**
    output(U)
  else
    Choose $e \in V$
    Data-Peeler$(U \cup \{e\}, \{v \in V \setminus \{e\} | \mathcal{C}_{\text{connected}}(U \cup \{e\} \cup \{v\})\}, \{s \in \mathcal{S} | \mathcal{C}_{\text{connected}}(U \cup \{e\} \cup \{s\})\})$
    Data-Peeler$(U, V \setminus \{e\}, \mathcal{S} \cup \{e\})$
  end if
end if

Algorithm 1: Data-Peeler

$\mathcal{C}$ is a user-defined constraint which allows to focus on relevant patterns while decreasing the extraction time by pruning enumeration sub-trees. To be able to efficiently check whether a 3-set descendant from a node satisfies $\mathcal{C}$, $\mathcal{C}$ must be a piecewise (anti)-monotone constraint.

3.2 Piecewise (Anti)-Monotone Constraints

Data-Peeler can efficiently check any piecewise (anti)-monotone constraint $\mathcal{C}$. By "efficiently", we mean it *sometimes* can, from the 3-sets U and V attached to a node (no access to the data), affirm that the enumeration sub-tree rooted by this node is empty of (not necessarily connected or closed) 3-sets satisfying $\mathcal{C}$. When the node is a leaf, it, not only *sometimes*, but *always* can check a piecewise (anti)-monotone constraint, hence ensuring the correctness, i. e., every extracted closed 3-set verifies $\mathcal{C}$. Let us first define the monotonicity and anti-monotonicity per argument.

Definition 10 ((Anti)-monotonicity per argument). A constraint $\mathcal{C}$ is said monotone (resp. anti-monotone) w.r.t. the i^{th} argument iff it is monotone (resp. anti-monotone) when all its arguments but the i^{th} are considered constant.

Example 5. Consider the following constraint, which forces the patterns to cover at least eight 3-tuples in the relation:

$$\text{A 3-set } (D^1, D^2, D^3) \text{ is 8-large} \Leftrightarrow |D^1 \times D^2 \times D^3| \geq 8 \ .$$

It is monotone on the first argument. Indeed, $\forall (D^1, D^{1\prime}, D^2, D^3) \in 2^{\mathcal{D}^1} \times 2^{\mathcal{D}^1} \times 2^{\mathcal{D}^2} \times 2^{\mathcal{D}^3}, D^1 \subseteq D^{1\prime} \Rightarrow (|D^1 \times D^2 \times D^3| \geq 8 \Rightarrow |D^{1\prime} \times D^2 \times D^3| \geq 8)$. It is monotone on the second and on the third argument too.

When a constraint $\mathcal{C}$ is either monotone or anti-monotone on every argument, Data-Peeler can efficiently check it. At a given node, it replaces the i^{th} argument by:

- $U^i \cup V^i$ if $\mathcal{C}$ is monotone on this argument;
- U^i if $\mathcal{C}$ is anti-monotone on this argument.

In this way, a 3-set (D^1, D^2, D^3) is obtained $(\forall i \in \{1,2,3\}, D^i \in \{U^i, U^i \cup V^i\})$. If $\mathcal{C}(D^1, D^2, D^3)$ then at least this 3-set, descendant from the current node, verifies $\mathcal{C}$. Otherwise the sub-tree rooted by the current node can safely be pruned: it does not contain any 3-set satisfying $\mathcal{C}$.

Example 6. Given the two 3-sets $U = (U^1, U^2, U^3)$ and $V = (V^1, V^2, V^3)$ attached to a node, Data-Peeler checks the 8-large constraint (defined in Ex. 5), by testing whether $|U^1 \cup V^1| \times |U^2 \cup V^2| \times |U^3 \cup V^3| \geq 8$.

The class of piecewise (anti)-monotone constraints contains every constraint which is either monotone or anti-monotone on each of its arguments. But it contains many other useful constraints. The definition of piecewise (anti)-monotonicity relies on attributing a separate argument to every occurrence of every variable and, then, proving that the obtained constraint is (anti)-monotone w.r.t. each of its arguments.

Definition 11 (Piecewise (anti)-monotonicity). A constraint $\mathcal{C}$ is piecewise (anti)-monotone iff the rewritten constraint $\mathcal{C}'$, attributing a separate argument to every occurrence of every variable in the expression of $\mathcal{C}$, is (anti)-monotone w.r.t. each of its arguments.

To illustrate this class of constraints, the particular context where $\mathcal{D}^1 = \mathcal{T} \in \mathbb{R}_+^{|\mathcal{T}|}$ is chosen:

Example 7. Consider the following constraint $\mathcal{C}_{16-\text{small-in-average}}$:

$$\mathcal{C}_{16-\text{small-in-average}}(T, D^2, D^3) \Leftrightarrow T \neq \emptyset \wedge \frac{\sum_{t \in T} t}{|T|} \leq 16 \ .$$

This constraint is both monotone and anti-monotone on the second and the third argument (neither D^2 nor D^3 appearing in the expression of the constraint) but it is neither monotone nor anti-monotone on the first argument. However, giving three different variables T_1, T_2 and T_3 to each of the occurrences of T creates this new constraint which is monotone on the first and third arguments (T_1 and T_3) and anti-monotone on the second one (T_2):

$$\mathcal{C}'_{16-\text{small-in-average}}(T_1, T_2, T_3, D^2, D^3) \equiv T_1 \neq \emptyset \wedge \frac{\sum_{t \in T_2} t}{|T_3|} \leq 16 \ .$$

Therefore $\mathcal{C}_{16-\text{small-in-average}}$ is piecewise (anti)-monotone.

Data-Peeler can efficiently check any piecewise (anti)-monotone constraint. First, it considers the analogous constraint where every occurrence of the three original attributes is given a different variable. Then, it applies the rules stated previously, i. e., at a given node, it replaces the i^{th} argument by:

- $U^i \cup V^i$ if $\mathcal{C}$ is monotone on this argument;
- U^i if $\mathcal{C}$ is anti-monotone on this argument.

The built assertion is false if, in the enumeration sub-tree that would derive from the node, there is no 3-set satisfying the original constraint. Notice that, in this general setting, the reverse may be false, i. e., the assertion can hold even if no 3-set descendant from the node that verifies the original constraint. Therefore, it can be written that Data-Peeler relaxes the constraint to efficiently check it.

4 Extracting δ-Contiguous Closed 3-Sets

4.1 A Piecewise (Anti)-Monotone Constraint...

The constraint $\mathcal{C}_{\delta\text{-contiguous}}$ (see Def. 4) is piecewise (anti)-monotone.

Proof. Let $\mathcal{C}'_{\delta\text{-contiguous}}$ the following constraint:

$$\begin{aligned}&\mathcal{C}'_{\delta\text{-contiguous}}(T_1, T_2, T_3, N_1, N_2)\\ \equiv\ &\forall t \in [\min(T_1), \max(T_2)], \exists t' \in T_3 \text{ s.t. } |t - t'| \leq \delta\ .\end{aligned}$$

The three arguments T_1, T_2 and T_3 substitute the three occurrences of T (in the definition of $\mathcal{C}_{\delta\text{-contiguous}}$). $\mathcal{C}'_{\delta\text{-contiguous}}$ is monotone in on its third argument and anti-monotone on its first and second arguments ($T \subseteq T_1 \Rightarrow \min(T) \geq \min(T_1)$ and $T \subseteq T_2 \Rightarrow \max(T) \leq \max(T_2)$). Moreover, since the two last arguments of $\mathcal{C}'_{\delta\text{-contiguous}}$ do not appear in its expression, this constraint is both monotone and anti-monotone on them. Therefore, by definition, $\mathcal{C}_{\delta\text{-contiguous}}$ is piecewise (anti)-monotone. □

4.2 ... Partially Handled in Another Way

Given the 3-sets $U = (U^{\mathcal{T}}, U^{\mathcal{N}^1}, U^{\mathcal{N}^2})$ and $V = (V^{\mathcal{T}}, V^{\mathcal{N}^1}, V^{\mathcal{N}^2})$ attached to the current enumeration node, the proof of Sect. 4.1 suggests to check whether it is possible to browse all elements in $[\min(U^{\mathcal{T}}), \max(U^{\mathcal{T}})] \cap (U^{\mathcal{T}} \cup V^{\mathcal{T}})$ by jumps of, at most, δ.

By also taking a look "around" $[\min(U^{\mathcal{T}}, \max(U^{\mathcal{T}})] \cap (U^{\mathcal{T}} \cup V^{\mathcal{T}})$, Data-Peeler can do better than just telling whether there is no hope in ex-

tracting δ-contiguous 3-sets from the current enumeration node. It can prevent the traversal of some of such nodes. More precisely, DATA-PEELER removes from $V^{\mathcal{T}}$ the elements that would, if enumerated, generate left children violating $\mathcal{C}_{\delta\text{-contiguous}}$. To do so, the delay between $t = \min(U^{\mathcal{T}})$ and $\text{before}(t) = \max(\{t' \in V^{\mathcal{T}} | t' < t\})$ is considered. If it is strictly greater than δ then every element in $\{t' \in V^{\mathcal{T}} | t' < t\}$ can be removed from $V^{\mathcal{T}}$. Otherwise, the process goes on with $t = \text{before}(t)$ until a delay greater than δ is found or until $t = \min(V^{\mathcal{T}})$ (in this case no element from $V^{\mathcal{T}}$ lesser than $\min(U^{\mathcal{T}})$ is removed). In a reversed way, the elements in $V^{\mathcal{T}}$ that are too great to be moved to $U^{\mathcal{T}}$ without violating $\mathcal{C}_{\delta\text{-contiguous}}$ are removed as well. Algorithm 2 gives a more technical definition of DATA-PEELER's way to purge $V^{\mathcal{T}}$ thanks to $\mathcal{C}_{\delta\text{-contiguous}}$.

Input: $U^{\mathcal{T}}, V^{\mathcal{T}}$
if $U^{\mathcal{T}} \neq \emptyset$ **then**
  $V^{\mathcal{T}} \leftarrow$ **sort**$(V^{\mathcal{T}})$
  $t \leftarrow \min(U^{\mathcal{T}})$
  if $t > \min(V^{\mathcal{T}})$ **then**
    $\text{before}(t) \leftarrow \max(\{t' \in V^{\mathcal{T}} | t' < t\})$ {Binary search in $V^{\mathcal{T}}$}
    while $\text{before}(t) \neq \min(V^{\mathcal{T}}) \wedge t - \text{before}(t) \leq \delta$ **do**
      $t \leftarrow \text{before}(t)$
      $\text{before}(t) \leftarrow$ **previous**$(V^{\mathcal{T}}, t)$ {$V^{\mathcal{T}}$ is browsed backward}
    end while
    if $t - \text{before}(t) > \delta$ **then**
      $V^{\mathcal{T}} \leftarrow V^{\mathcal{T}} \setminus [\min(V^{\mathcal{T}}), \text{before}(t)]$
    end if
  end if
  $t \leftarrow \max(U^{\mathcal{T}})$
  if $t < \max(V^{\mathcal{T}})$ **then**
    $\text{after}(t) \leftarrow \min(\{t' \in V^{\mathcal{T}} | t' > t\})$ {Binary search in $V^{\mathcal{T}}$}
    while $\text{after}(t) \neq \max(V^{\mathcal{T}}) \wedge \text{after}(t) - t \leq \delta$ **do**
      $t \leftarrow \text{after}(t)$
      $\text{after}(t) \leftarrow$ **next**$(V^{\mathcal{T}}, t)$ {$V^{\mathcal{T}}$ is browsed forward}
    end while
    if $\text{after}(t) - t > \delta$ **then**
      $V^{\mathcal{T}} \leftarrow V^{\mathcal{T}} \setminus [\text{after}(t), \max(V^{\mathcal{T}})]$
    end if
  end if
end if

Algorithm 2: PURGE_$V^{\mathcal{T}}$

In the same way, some elements of $\mathcal{S}^{\mathcal{T}}$ may be too far away from the extrema of $U^{\mathcal{T}} \cup V^{\mathcal{T}}$ to prevent the δ-closedness of any descending 3-set. These elements are those that cannot be added to $U^{\mathcal{T}}$ without making the current enumeration node violate $\mathcal{C}_{\delta\text{-contiguous}}$. Hence, DATA-PEELER removes these elements by applying a procedure PURGE_$\mathcal{S}^{\mathcal{T}}$ to every enumeration node. It is very similar to PURGE_$V^{\mathcal{T}}$ (see Alg. 2) except that it is $\mathcal{S}^{\mathcal{T}}$ which is browsed

backward from before($\min(U^{\mathcal{T}} \cup V^{\mathcal{T}})$) and forward from after($\max(U^{\mathcal{T}} \cup V^{\mathcal{T}})$).

Example 8. Considering the extraction of 1-contiguous 3-sets from the example dataset defined by Table 1, if the first enumerated element is 0.5, Fig. 3 depicts the root enumeration node and its two children. In the left child, PURGE_$V^{\mathcal{T}}$ removes 2 and 3 from its attached $V^{\mathcal{T}}$ set because $2 - 0.5 > 1$.

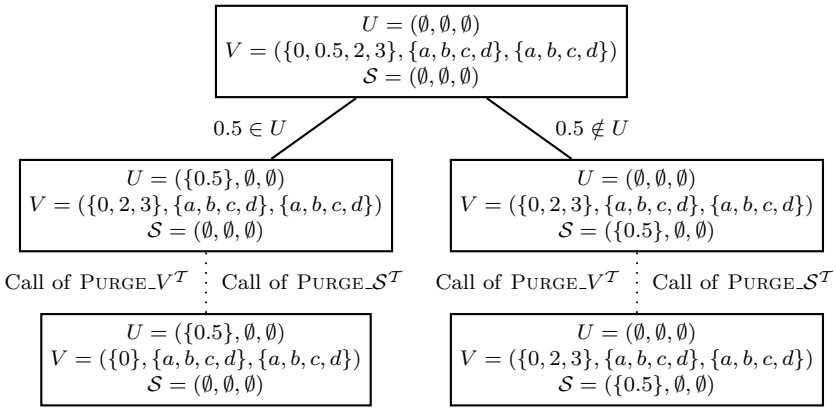

Fig. 3 Enumeration of $0.5 \in V$ during the extraction of 1-contiguous 3-sets from the example dataset defined by Table 1

These purges of V and $\mathcal{S}$ remind the way DATA-PEELER handles $\mathcal{C}_{\text{connected}}$. $\mathcal{C}_{\text{connected}}$ is anti-monotone on all its arguments, whereas $\mathcal{C}_{\delta\text{-contiguous}}$ is *only* piecewise (anti)-monotone. Hence some enumeration nodes violating $\mathcal{C}_{\delta\text{-contiguous}}$ may be generated despite the calls of PURGE_$V^{\mathcal{T}}$ (whereas a generated enumeration node always complies with $\mathcal{C}_{\text{connected}}$). As a consequence, checking, at every enumeration node, whether $\mathcal{C}_{\delta\text{-contiguous}}$ holds remains necessary. For the same reason, some elements in the 3-sets V and/or $\mathcal{S}$ attached to both left and right children may be purged thanks to $\mathcal{C}_{\delta\text{-contiguous}}$ (whereas $\mathcal{C}_{\text{connected}}$ cannot reduce the search space of a right child).

4.3 Enforcing the δ-Closedness

The constraint $\mathcal{C}_{\delta\text{-closed}}$ (see Def. 5) is piecewise (anti)-monotone.

Proof. Let $\mathcal{C}'_{\delta\text{-closed}}$ the following constraint:

$$\mathcal{C}'_{\delta\text{-closed}}(T_1, T_2, T_3, T_4, N_1^1, N_2^1, N_3^1, N_1^2, N_2^2, N_3^2)$$
$$\equiv \begin{cases} \forall t \in \mathcal{T} \setminus T_1, (\exists t' \in T_2 \text{ s.t. } |t - t'| \leq \delta \Rightarrow \neg\mathcal{C}_{\text{connected}}(\{t\}, N_1^1, N_1^2)) \\ \forall n^1 \in \mathcal{N} \setminus N_2^1, \neg\mathcal{C}_{\text{connected}}(T_3, \{n^1\}, N_2^2) \\ \forall n^2 \in \mathcal{N} \setminus N_3^2, \neg\mathcal{C}_{\text{connected}}(T_4, N_3^1, \{n^2\}) \end{cases}.$$

$\mathcal{C}'_{\delta\text{-closed}}$ is anti-monotone on its second argument and monotone on all its other arguments. Therefore, by definition, $\mathcal{C}_{\delta\text{-closed}}$ is piecewise (anti)-monotone. □

A way to enforce $\mathcal{C}_{\delta\text{-closed}}$ follows from the proof of its piecewise (anti)-monotonicity: an enumeration node, i. e., its attached $U = (U^{\mathcal{T}}, U^{\mathcal{N}^1}, U^{\mathcal{N}^2})$ and $V = (V^{\mathcal{T}}, V^{\mathcal{N}^1}, V^{\mathcal{N}^2})$, may lead to some δ-closed 3-set if $(U^{\mathcal{T}} \cup V^{\mathcal{T}}, U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1}, U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$:

- cannot be extended by any element in $\mathcal{T} \setminus (U^{\mathcal{T}} \cup V^{\mathcal{T}})$ distant, by at most δ, from an element in $U^{\mathcal{T}}$;
- cannot be extended by any element in $\mathcal{N} \setminus (U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1})$;
- cannot be extended by any element in $\mathcal{N} \setminus (U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$.

As done for $\mathcal{C}_{\text{closed}}$, to avoid useless (and costly) tests, Data-Peeler maintains the 3-set $\mathcal{S} = (\mathcal{S}^{\mathcal{T}}, \mathcal{S}^{\mathcal{N}^1}, \mathcal{S}^{\mathcal{N}^2})$ containing only the elements that may prevent the closure of the 3-sets descending from the current enumeration node, i. e., the previously enumerated elements and not those that were removed from V thanks to $\mathcal{C}_{\text{connected}} \wedge \mathcal{C}_{\delta\text{-contiguous}}$. Moreover, as explained in Sect. 4.2, Data-Peeler purges $\mathcal{S}$ before checking $\mathcal{C}_{\delta\text{-closed}}$. Since it is used in conjunction with $\mathcal{C}_{\delta\text{-contiguous}}$, $\mathcal{C}_{\delta\text{-closed}}$ can be more strongly enforced: no element in $\mathcal{S}^{\mathcal{T}} \cap [\min(U^{\mathcal{T}}) - \delta, \max(U^{\mathcal{T}}) + \delta]$ is allowed to extend $(U^{\mathcal{T}} \cup V^{\mathcal{T}}, U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1}, U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$. Indeed, an element in $\mathcal{S}^{\mathcal{T}} \cap [\min(U^{\mathcal{T}}) - \delta, \max(U^{\mathcal{T}}) + \delta]$ may be distant, by strictly more than δ, from any element in $U^{\mathcal{T}}$ but this will never be the case at the leaves descending from the current enumeration since $U^{\mathcal{T}}$ must then be δ-contiguous. All in all, Data-Peeler prunes the sub-tree descending from the current enumeration node if $(U^{\mathcal{T}} \cup V^{\mathcal{T}}, U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1}, U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$ can be extended by any element in $\mathcal{S}^{\mathcal{T}} \cap [\min(U^{\mathcal{T}}) - \delta, \max(U^{\mathcal{T}}) + \delta]$, $\mathcal{S}^{\mathcal{N}^1}$ or $\mathcal{S}^{\mathcal{N}^2}$.

5 Constraining the Enumeration to Extract 3-Cliques

5.1 A Piecewise (Anti)-Monotone Constraint...

In a 3-clique, both subsets of $\mathcal{N}$ are identical. An equivalent definition to the symmetry constraint (Def. 7) would be as follows: $\mathcal{C}_{\text{symmetric}}(T, N^1, N^2) \equiv N^1 \subseteq N^2 \wedge N^2 \subseteq N^1$. In this form, a piecewise (anti)-monotone constraint is identified.

Proof. Let $\mathcal{C}'_{\text{symmetric}}$ the following constraint:

$$\mathcal{C}'_{\text{symmetric}}(T, N_1^1, N_2^1, N_1^2, N_2^2) \equiv N_1^1 \subseteq N_1^2 \wedge N_2^2 \subseteq N_2^1 \ .$$

N_1^1 and N_2^1 substitute the two occurrences of N^1 (in the alternative definition of $\mathcal{C}_{\text{symmetric}}$). In the same way, N_1^2 and N_2^2 substitute the two occurrences of N^2. $\mathcal{C}'_{\text{symmetric}}$ is monotone on its third and fourth arguments (N_2^1 and N_1^2) and anti-monotone on its second and fifth arguments (N_1^1 and N_2^2). Moreover, since the first argument (T) does not appear in the expression of $\mathcal{C}'_{\text{symmetric}}$, this constraint is both monotone and anti-monotone on this argument. Therefore, by definition, $\mathcal{C}_{\text{symmetric}}$ is piecewise (anti)-monotone. □

Being piecewise (anti)-monotone, the symmetry constraint can be efficiently exploited by Data-Peeler. However, the enumeration tree can be further reduced if this constraint is enforced when choosing the element to be enumerated.

5.2 . . . Better Handled in Another Way

In this section, a distinction between the "first" set of nodes (i. e., the rows of the adjacency matrices) and the "second" one (i. e., the columns of the adjacency matrices) must be made. They are respectively named $\mathcal{N}^1$ and $\mathcal{N}^2$. Intuitively, when an element n^1 from $V^1 \subseteq \mathcal{N}^1$ is chosen to be present (respectively absent) in any 3-clique extracted from the node (see Sect. 3.1), the element n^2 from $V^2 \subseteq \mathcal{N}^2$ standing for the same node should be enumerated just after and only to be present (respectively absent) too. Thus, the enumeration tree is not a binary tree anymore (some enumeration nodes only have one child).

When handled as a piecewise (anti)-monotone constraint, the symmetry constraint leads to many more enumeration nodes. When n^2 is chosen to be enumerated, the left (respectively right) child where n^2 is present (respectively absent) is generated even if its counterpart n^1 in the other set was previously set absent (respectively present). Then the symmetry constraint prunes the sub-tree rooted by this node. Since there is no reason for n^2 to be enumerated just after n^1, the intuition tells us that the number of such nodes, whose generation could be avoided by modifying the enumeration (as explained in the previous paragraph), increases exponentially with the average number of enumeration nodes between the enumeration of n^1 and that of n^2. This is actually not a theorem because $\mathcal{C}_{\text{sym-}\delta\text{-closed}}$ or $\mathcal{C}$ may prune some descendant sub-trees before n^2 is enumerated. Anyway, in practical settings, handling the symmetry constraint via a modification of the enumeration usually is much more efficient than via the general framework for piecewise (anti)-monotone constraints.

Figures 4a and 4b informally depict these two approaches (the probable diminutions of the V sets in the left children and the possible pruning due to $\mathcal{C}_{\text{closed}}$ or $\mathcal{C}$ are ignored). T_1 and T_2 are subsets of $\mathcal{T}$. N_1 and N_2 are subsets of $\mathcal{N}$. In both examples, the elements m^2 and n^2 of $\mathcal{N}^2$ are enumerated. The resulting nodes are, of course, the same (the dotted nodes being pruned). However this result is straightforward when the enumeration constraint is handled through a modification of the enumeration (Fig. 4b), whereas it usually requires more nodes when it is handled as an ordinary piecewise (anti)-monotone constraint (Fig. 4a). The number of additional nodes in the latter case grows exponentially with the number of elements enumerated between n^1 and n^2 (e. g., m^1 could be enumerated in between).

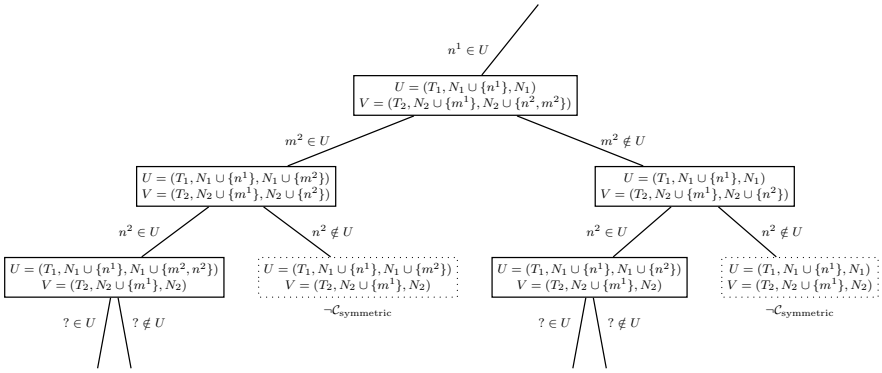

Fig. 4a Symmetry handled as an ordinary piecewise (anti)-monotone constraint

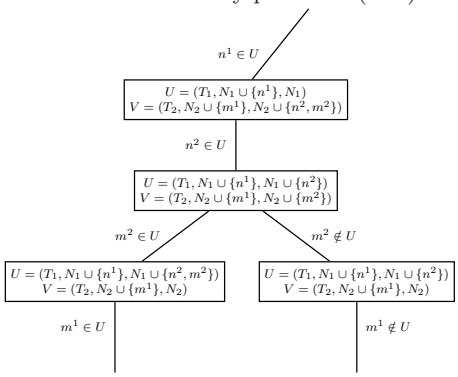

Fig. 4b Symmetry handled by a modified enumeration

5.3 Constraining the Enumeration

Let $\mathcal{N}^1 = (n_i^1)_{i=1\dots|\mathcal{N}|}$ and $\mathcal{N}^2 = (n_i^2)_{i=1\dots|\mathcal{N}|}$ its counterpart, i. e., $\forall i = 1\dots|\mathcal{N}|$, n_i^1 and n_i^2 stand for the same node. (T, N^1, N^2) being symmetric is a constraint that can be expressed as this list of, so called, *enumeration constraints*:

$$\begin{array}{cc}
n_1^1 \in N^1 \Rightarrow n_1^2 \in N^2 & n_1^2 \in N^2 \Rightarrow n_1^1 \in N^1 \\
n_2^1 \in N^1 \Rightarrow n_2^2 \in N^2 & n_2^2 \in N^2 \Rightarrow n_2^1 \in N^1 \\
\vdots & \vdots \\
n_i^1 \in N^1 \Rightarrow n_i^2 \in N^2 & n_i^2 \in N^2 \Rightarrow n_i^1 \in N^1 \\
\vdots & \vdots \\
n_{|\mathcal{N}|}^1 \in N^1 \Rightarrow n_{|\mathcal{N}|}^2 \in N^2 & n_{|\mathcal{N}|}^2 \in N^2 \Rightarrow n_{|\mathcal{N}|}^1 \in N^1
\end{array}$$

These constraints belong to a more general class of constraints:

Definition 12 (Enumeration constraint). An enumeration constraint $\mathcal{C}_{\text{enum}}$ is such that, given a 3-set (T, N^1, N^2), $\mathcal{C}_{\text{enum}}(T, N^1, N^2) \equiv \exists k \in \mathbb{N} | a_1 \wedge a_2 \wedge \dots \wedge a_k \Rightarrow a_{k+1}$, where $\forall i = 1 \dots k+1, a_i$ is of the form $e \in A$ or $e \notin A$, e being an arbitrary element from an arbitrary dimension $A \in \{T, N^1, N^2\}$.

Example 9. Here are three examples of enumeration constraints that can be enforced on any 3-set (T, N^1, N^2):

- $t_1 \in T \Rightarrow t_8 \notin T$
- $t_1 \notin T \wedge n_1^1 \in N^1 \Rightarrow t_2 \in T$
- true $\Rightarrow t_1 \notin T$ ($k = 0$ in Def. 12)

Notice that the last constraint is not equivalent to removing the element t_1 from the data. Indeed, a closed 3-set in the data set deprived of t_1 may not be closed in the data set containing t_1. Hence it must not be extracted (and it is actually not extracted when the constraint enumeration is used).

Before choosing the element to be enumerated (see Alg. 1), Data-Peeler browses the set of enumeration constraint, and tests whether the left parts of them are true or not. Considered as constraints, these left parts are, again, piecewise (anti)-monotone. Indeed, when there is a term of the form $e \in A$ (respectively $e \notin A$), the left part of the constraint is anti-monotone (respectively monotone) in this occurrence of A. Given the 3-sets U and V attached to the current enumeration node, three cases may arise:

1. The left part will never be fulfilled in the sub-tree rooted by the current enumeration node:
   - if an element in the left part is to be present but it is neither in U not in V.

- if an element in the right part is to be absent but it is in U.

2. The left part is fulfilled by at least one (but not every) node descending from the current enumeration node.
3. The left part is fulfilled by every node descending from the current enumeration node:

   - if an element in the left part is to be present, it is in U.
   - if an element in the left part is to be absent, it is neither in U nor in V.

   Data-Peeler reacts differently at each of these cases:

1. This enumeration constraint is removed from the set of enumeration constraints when traversing the sub-tree rooted by the current enumeration node. Indeed, it never applies in this sub-tree. Uselessly checking it for every descendant enumeration node would only decrease the performances of Data-Peeler.
2. This enumeration constraint is kept.
3. The right part of this enumeration constraint is considered.

When the right part of an enumeration constraint is considered, three new cases may arise:

3.1 The right part is already fulfilled:

- if the element in the right part is to be present, it is already in U.
- if is to be absent, it is already neither in U nor in V.

3.2 The right part can be fulfilled: the element in the right part is in V.
3.3 The right part cannot be fulfilled:

- if the element in the right part is to be present, it is neither in U nor in V.
- if it is to be absent, it is in U.

Data-Peeler reacts differently at each of these cases:

3.1 This enumeration constraint is removed from the set of enumeration constraints when traversing the sub-tree rooted by the current enumeration node. Indeed, it is satisfied for all 3-sets in this sub-tree. Uselessly checking it for every descendant enumeration node would only decrease the performances of Data-Peeler.
3.2 The element on the right part of the constraint can be enumerated as specified (one child only).
3.3 The sub-tree rooted by the current enumeration node is pruned. Indeed, none of the 3-sets in this sub-tree verifies the constraint.

In Case 3.2, we write "the element *can* be enumerated" because, at a given enumeration node, several enumeration constraint may be in this case but only one can be applied.

5.4 Contraposition of the Enumeration Constraints

If an enumeration constraint holds, its contraposition, logically, holds too. In the general case (conjunction of terms in the left part), the contraposition of an enumeration constraint is not an enumeration constraint (disjunction of terms in the right part). In the particular case of enumeration constraints of the form $a_1 \Rightarrow a_2$ (see Def. 12), e. g., those generated from $\mathcal{C}_{\text{symmetric}}$ (see Sect. 5.3), their contrapositions are enumeration constraints too. Thus, DATA-PEELER enforces a larger set of enumeration constraints (the original set of enumeration constraints and the contrapositions of those of the form $a_1 \Rightarrow a_2$) for even faster extractions. Algorithm 3 gives a more technical definition of how this larger set is computed.

Input: Set E of enumeration constraints
Output: Set E enlarged with contrapositions
$E' \leftarrow E$
for $a_1 \wedge a_2 \wedge \cdots \wedge a_k \Rightarrow a_{k+1} \in E$ **do**
  if $k = 1$ **then**
    $E' \leftarrow E' \cup \{\neg a_2 \Rightarrow \neg a_1\}$
  end if
end for
return E'

Algorithm 3: APPEND_CONTRAPOSITION

Example 10. Among the enumeration constraints of Ex. 9, only the first one ($t_1 \in T \Rightarrow t_8 \notin T$) admits a contraposition ($t_8 \in T \Rightarrow t_1 \notin T$) that is, itself, an enumeration constraint.

5.5 Enforcing the Symmetric δ-Closedness

The constraint $\mathcal{C}_{\text{sym-}\delta\text{-closed}}$ (see Def. 8) is piecewise (anti)-monotone.

Proof. Let $\mathcal{C}'_{\text{sym-}\delta\text{-closed}}$ the following constraint:

$$\begin{aligned}&\mathcal{C}'_{\text{sym-}\delta\text{-closed}}(T_1, T_2, T_3, N_1^1, N_2^1, N_3^1, N_1^2, N_2^2, N_3^2)\\ \equiv &\begin{cases}\forall t \in \mathcal{T} \setminus T_1, (\exists t' \in T_2 \text{ s.t. } |t - t'| \leq \delta \Rightarrow \neg\mathcal{C}_{\text{connected}}(\{t\}, N_1^1, N_1^2))\\ \forall n \in \mathcal{N} \setminus (N_2^1 \cap N_2^2), \neg\mathcal{C}_{\text{connected}}(T, N_3^1 \cup \{n\}, N_3^2 \cup \{n\})\end{cases}.\end{aligned}$$

$\mathcal{C}'_{\text{sym-}\delta\text{-closed}}$ is anti-monotone on its second argument (T_2) and monotone on all its other arguments. Therefore, by definition, $\mathcal{C}_{\text{sym-}\delta\text{-closed}}$ is piecewise (anti)-monotone. □

A way to enforce $\mathcal{C}_{\delta\text{-closed}}$ follows from the proof of its piecewise (anti)-monotonicity: an enumeration node, i. e., its attached $U = (U^{\mathcal{T}}, U^{\mathcal{N}^1}, U^{\mathcal{N}^2})$ and $V = (V^{\mathcal{T}}, V^{\mathcal{N}^1}, V^{\mathcal{N}^2})$, may lead to some δ-closed 3-set if $(U^{\mathcal{T}} \cup V^{\mathcal{T}}, U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1}, U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$:

- cannot be extended by any element in $\mathcal{T} \setminus (U^{\mathcal{T}} \cup V^{\mathcal{T}})$ distant, by at most δ, from an element in $U^{\mathcal{T}}$;
- cannot be simultaneously extended by any element in $\mathcal{N} \setminus (U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1})$ (row of the adjacency matrices) and its related element in $\mathcal{N} \setminus (U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$ (column of the adjacency matrices).

In a similar way to what was done with $\mathcal{C}_{\delta\text{-closed}}$ (see Sect. 4.3), Data-Peeler maintains the 3-set $\mathcal{S} = (\mathcal{S}^{\mathcal{T}}, \mathcal{S}^{\mathcal{N}^1}, \mathcal{S}^{\mathcal{N}^2})$ containing only the elements that may prevent the closure of the 3-sets descending from the current enumeration node and prunes the sub-tree descending from it if $(U^{\mathcal{T}} \cup V^{\mathcal{T}}, U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1}, U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$ can be extended by any element in $\mathcal{S}^{\mathcal{T}} \cap [\min(U^{\mathcal{T}}) - \delta, \max(U^{\mathcal{T}}) + \delta]$ or by any element in $\mathcal{S}^{\mathcal{N}^1}$ *and* its related element in $\mathcal{S}^{\mathcal{N}^2}$. Thus, when $\mathcal{S}^{\mathcal{N}^1}$ (respectively $\mathcal{S}^{\mathcal{N}^2}$) is purged from an element (because it cannot extend $(U^{\mathcal{T}} \cup V^{\mathcal{T}}, U^{\mathcal{N}^1} \cup V^{\mathcal{N}^1}, U^{\mathcal{N}^2} \cup V^{\mathcal{N}^2})$ without violating $\mathcal{C}_{\text{connected}}$), the related element in $\mathcal{S}^{\mathcal{N}^2}$ (respectively $\mathcal{S}^{\mathcal{N}^1}$) is removed as well.

An overall view of the complete extraction of the δ-contiguous closed 3-cliques under constraint can now be presented. The details and justifications of how every identified constraint is handled are present within the two previous sections, hence proving its correctness. Algorithm 4 is the main procedure solving the problem presented in Sect. 2. It calls Algorithm 5 which can be regarded as a specialization of Algorithm 1.

Input: $(A_t)_{t \in \mathcal{T}} \in \{0,1\}^{\mathcal{T} \times \mathcal{N} \times \mathcal{N}}$, $\delta \in \mathbb{R}_+$ and a user-defined piecewise (anti)-monotone constraint $\mathcal{C}$
Output: All δ-contiguous closed 3-cliques in $(A_t)_{t \in \mathcal{T}}$ satisfying $\mathcal{C}$
$E \leftarrow$ Set of enumeration constraints pertaining to $\mathcal{C}_{\text{symmetric}}$ (see Sect. 5.3)
$E' \leftarrow$ Append_contraposition(E)
Data-Peeler($(\emptyset, \emptyset, \emptyset), (\mathcal{T}, \mathcal{N}, \mathcal{N}), (\emptyset, \emptyset, \emptyset)$)

Algorithm 4: main

6 Experimental Results

The experiments were performed on an AMD Sempron™ 2600+ computer with 512 MB of RAM and running a GNU/Linux™ operating system. Data-Peeler was compiled with GCC 4.3.2.

Input: $U, V, \mathcal{S}$
Output: All δ-contiguous closed 3-cliques containing the elements in U and, possibly, some elements in V and satisfying $\mathcal{C}$
PURGE_$V^{\mathcal{T}}$
PURGE_$\mathcal{S}^{\mathcal{T}}$
if $\mathcal{C} \wedge \mathcal{C}_{\delta\text{-contiguous}} \wedge \mathcal{C}_{\text{sym-}\delta\text{-closed}}$ may be satisfied by a 3-set descending from this node **then**
  Process E' as detailed in Sect. 5.3
  if Case 3.3 was never encountered **then**
    if $V = (\emptyset, \emptyset, \emptyset)$ **then**
      output(U)
    else
      if Case 3.2 was encountered with an enumeration constraint concluding on a_{k+1} (see Def. 12) **then**
        if a_{k+1} is of the form $e \in A$ **then**
          DATA-PEELER$(U \cup \{e\}, \{v \in V \setminus \{e\} | \mathcal{C}_{\text{connected}}(U \cup \{e\} \cup \{v\})\}, \{s \in \mathcal{S} | \mathcal{C}_{\text{connected}}(U \cup \{e\} \cup \{s\})\})$
        else
          a_{k+1} is of the form $e \notin A$
          DATA-PEELER$(U, V \setminus \{e\}, \mathcal{S} \cup \{e\})$
        end if
      else
        Choose $e \in V$
        DATA-PEELER$(U \cup \{e\}, \{v \in V \setminus \{e\} | \mathcal{C}_{\text{connected}}(U \cup \{e\} \cup \{v\})\}\}, \{s \in \mathcal{S} | \mathcal{C}_{\text{connected}}(U \cup \{e\} \cup \{s\})\})$
        DATA-PEELER$(U, V \setminus \{e\}, \mathcal{S} \cup \{e\})$
      end if
    end if
  end if
end if

Algorithm 5: DATA-PEELER specialization

6.1 Presentation of the Vélo'v Dataset

Vélo'v is a bicycle rental service run by the city of Lyon, France. 338 Vélov stations are spread over this city. At any of these stations, the users can take a bicycle and return it to any other station. Whenever a bicycle is rented or returned, this event is logged. We focus here on the data generated during the year 2006. These data are aggregated to obtain one graph per period of time (we chose a period of 30 minutes). For instance, one of these graphs presents the activity of the network during an average Monday of 2006 between nine o'clock and half past nine. The set of nodes $\mathcal{N}$ of such a graph corresponds to the Vélo'v stations. Its edges are labelled with the total number of rides in 2006 between the two linked stations (whatever their orientation) during the considered period of time. Setting a threshold allows to select the most significant edges. Many statistical tests can be used to fix this threshold (which can be different between the graphs). We opted for the rather simple procedure below.

α-binarization Given a graph whose edges are labelled by values quantifying them, let m be the maximum of these values. Given a user-defined real number $\alpha \in [0, 1]$ (common to all graphs), the threshold is fixed to $(1 - \alpha) \times m$.

Once the thresholds set, all edges linked to some station may be considered insignificant. Such an infrequently used station is removed from the dynamic graph. In our experiments, 204 stations remained after an α-binarization with $\alpha = 0.8$. Unless an experiment requires different datasets (scalability w.r.t. the density), every experiment uses this extraction context.

To filter out the 3-cliques corresponding to frequent rides between two stations only, a monotone constraint pertaining to the number of stations is enforced: the 3-cliques must involve at least 3 nodes to be extracted.

6.2 Extracting Cliques Via Enumeration Constraints

To confirm that the use of enumeration constraints actually helps in reducing the extraction time, three different strategies for 3-clique extraction are empirically compared:

1. Data-Peeler extracts all closed 3-sets. Among them, the 3-cliques are collected by post-processing: all closed 3-sets are browsed and those that are not symmetric are filtered out. Notice that this strategy is correct for this application because the considered dynamic graph is undirected, hence, a 3-set that does not satisfy $\mathcal{C}_{\delta\text{-closed}}$ will not satisfy $\mathcal{C}_{\text{sym-}\delta\text{-closed}}$ either.
2. Data-Peeler handles the symmetry constraint via "classical" piecewise (anti)-monotone constraints (see Sect. 5.1).
3. Data-Peeler handles the symmetry constraint via enumeration constraints (see Sect. 5.2).

Figure 5a depicts the extraction times of these three strategies under different minimal size constraints on the number of time periods to be present (abscissa). In this experiment the second strategy is only slightly faster than the extraction of all closed 3-sets (notice however that the required post-treatment is not included in the plotted results), whereas the use of enumeration constraints significantly reduces the extraction time.

This advantage grows with the density of the dataset. To test this, another binarization is used. It directly controls the number of edges kept in the dynamic graph:

β-binarization Given a graph whose edges are labelled by values quantifying them and a user-defined real number $\beta \in [0, 1]$ (common to all graphs), the edges labelled with the $\beta \times |\mathcal{N}^2|$ highest values are kept.

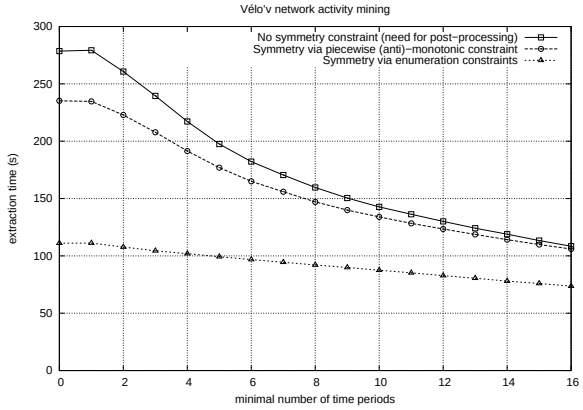

Fig. 5a Extraction times for different strategies (variable minimal size constraint)

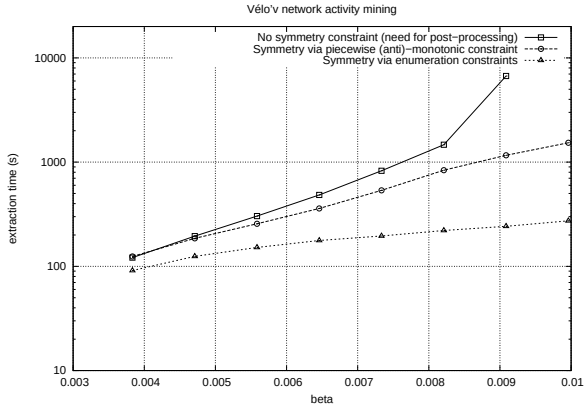

Fig. 5b Extraction times for different strategies (variable density)

Figure 5b shows how handling the symmetry via enumeration constraints more and more reduces the extraction time when β grows. As explained earlier, the number of nodes may be changed when we increase the number of edges. In this experiment, it varies between 201 (when $\beta = 0.0038$) and 240 (when $\beta = 0.01$). In addition to the minimal size constraint on the number of stations (at least three) involved in every extracted pattern, each of these patterns is, here, forced to gather at least two periods of time too. When $\beta = 0.0091$, it takes almost two hours to extract all 1,033,897 closed 3-sets. Among them, the post-process would retain the 18,917 ones that are symmetric. In contrast, these cliques are directly extracted in less than 20 minutes when the symmetry constraint is enforced as a piecewise (anti)-monotone constraint. The use of enumeration constraints provides the best performance: the extraction takes about four minutes.

Adding, to the set of enumeration constraints generated from the symmetry constraint, their contrapositions (see Sect. 5.4), is believed to improve the extraction time. However the cost of checking the application (or the non application) of a larger set of enumeration constraints brings an overhead. The following experiment confirms the advantage in using a larger set of enumeration constraints.

For each node n (more precisely, for each $n^1 \in \mathcal{N}^1$ or $n^2 \in \mathcal{N}^2$), one of the following sets of enumeration constraints is sufficient to enforce the symmetry constraint:

Set 1	Set 2 (contraposition of Set 1)	Set 3 (union of Set 1 and Set 2)
		$n^1 \in \mathcal{N}^1 \Rightarrow n^2 \in \mathcal{N}^2$
$n^1 \in \mathcal{N}^1 \Rightarrow n^2 \in \mathcal{N}^2$	$n^2 \in \mathcal{N}^2 \Rightarrow n^1 \in \mathcal{N}^1$	$n^1 \notin \mathcal{N}^1 \Rightarrow n^2 \notin \mathcal{N}^2$
$n^2 \in \mathcal{N}^2 \Rightarrow n^1 \in \mathcal{N}^1$	$n^2 \notin \mathcal{N}^2 \Rightarrow n^1 \notin \mathcal{N}^1$	$n^2 \notin \mathcal{N}^2 \Rightarrow n^1 \notin \mathcal{N}^1$
		$n^1 \notin \mathcal{N}^1 \Rightarrow n^2 \notin \mathcal{N}^2$

The results are plotted in Figs. 6a and 6b. The experimental context is perfectly identical to that of the experiment depicted in Fig. 5a. The running times obtained with Set 2 are lower than those obtained with Set 1 because the closed 3-cliques involve small proportions of the nodes in $\mathcal{N}$. That is why what is *not* in the patterns more frequently triggers enumeration constraints. Anyway, the fastest extractions are obtained with the largest set of enumeration constraints. It may look odd that, while being faster, the extractions performed with Set 2 generates many more enumeration nodes than those generated with Set 1. The difference between the costs of generating left enumeration nodes and right enumeration nodes explain it. Indeed, although more enumeration nodes are traversed when using Set 2, these nodes mainly are right nodes (the constraints in Set 2 conclude on such nodes), whereas the constraints in Set 1 impose the creation of left enumeration nodes. The left enumeration nodes do not prune much the search space (hence their numbers) but are very cheap to generate since the cost only is that of moving an element from a vector to another (see Fig 2).

6.3 Extraction of δ-Contiguous Closed 3-Cliques

Figure 7a depicts the number of δ-contiguous closed 3-cliques when δ varies between 0 and 8 hours. Different minimal size constraints, on the number of time periods to be present in any extracted pattern, are used. When this minimal size is set to 1, the number of δ-contiguous closed 3-cliques decreases while δ increases. This means that this dynamic graph contains many 3-cliques with one time period only. When δ grows, some of these 3-cliques are merged, thus gathering more time periods. That is why, when the patterns are constrained to gather at least two (or more) time periods, the size of the

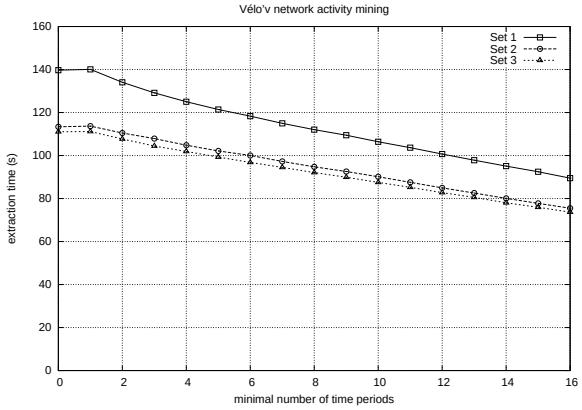

Fig. 6a Running time with different sets of enumeration constraints

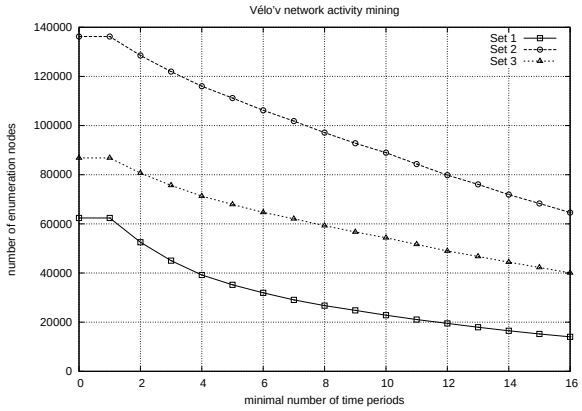

Fig. 6b Number of enumeration nodes with different sets of enumeration constraints

collection of δ-contiguous closed 3-cliques increases with δ. These behaviors are data-dependent. For example, under a size constraint greater or equal to 2, it is possible to find datasets where, when δ increases, the size of the collection would first increase (the involved timestamps were too distant to be extracted with smaller δs) and then decrease (the patterns found with smaller δs merge).

Figure 7b shows that smaller δs mean smaller extraction times. Hence, if a dynamic graph gathers many timestamps, enforcing a δ-contiguity helps a lot in making the knowledge extraction tractable. Furthermore this performance gain, that occurs when δ decreases, is greater when the minimal size constraints (on the number of timestamps) is smaller. Thus the performance gain is even more useful to compensate the difficulty to extract patterns that contain few timestamps. In the figure, the divergence of the curves, when δ increases, illustrates this interesting property.

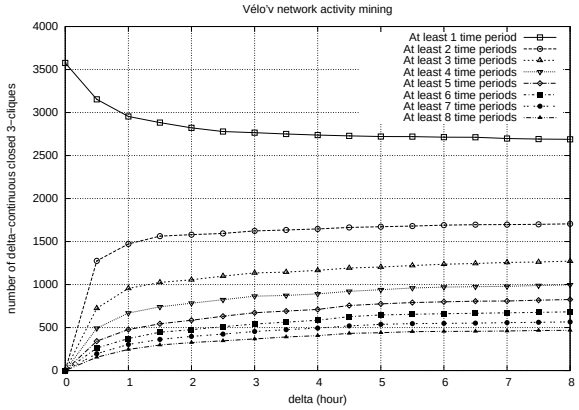

Fig. 7a Number of δ-contiguous closed 3-sets

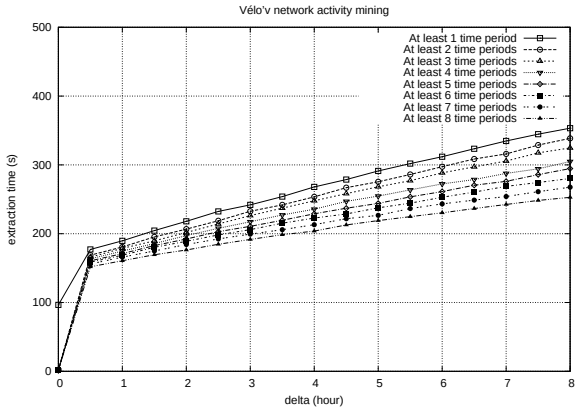

Fig. 7b Running time

6.4 Qualitative Validation

To assess, by hand, the quality of the extracted δ-contiguous closed 3-cliques, the returned collection must be small. Hence stronger constraints are enforced. The minimal number of Vélov stations that must be involved in a δ-contiguous closed 3-clique is raised to 6 and the minimal number of periods to 4. With $\delta = 0.5$ hours, only three patterns are returned. Two of them take place during the evening (they start at half past 19) and gather stations that are in the center of Lyon (the "2nd and 3rd arrondissement"). They differ by one station (one station is present in the first 0.5-contiguous closed 3-clique and absent from the other and vice versa) and one of them runs during one more time period. An agglomerative post-process, such as [12], would certainly merge these two patterns. The third 0.5-contiguous closed 3-clique is displayed in Fig 8a. The circles stand for the geographical positions of the

Vélov stations. The larger and filled circles are the stations involved in the shown pattern. The disposition of the stations follows one of the main street in Lyon: "Cours Gambetta". Obviously it is much used by the riders during the evening. The outlying Vélov station is, overall, the most frequently used one: "Part-Dieu/Vivier-Merle". At this place, the rider finds the only commercial center in Lyon, the main train station, etc.

Extracting, with the same minimal size constraints, the 1-contiguous closed 3-cliques provides a collection of nine patterns. Among them, the three 0.5-contiguous closed 3-cliques are found unaltered; some slight variations of them are found (one or two stations are changed); one pattern takes place during the morning (to obtain patterns involving night periods the constraints must be weakened a lot: nightly rides do not comply much with a model). The majority of the extracted 1-contiguous closed 3-cliques involves Vélov stations in the "2nd and 3rd arrondissement". Figure 8b depicts one of them. The disposition of the stations follows the street connecting the two most active districts in Lyon: "Rue de la Part-Dieu". The outlying Vélov station is, overall, one of the most frequently used: "Opéra". At this place, the rider can find, not only the opera, but also the town hall, the museum of fine arts, a cinema, bars, etc. For the maintenance of the Vélov network, these examples of constrained cliques correspond to relevant sub-networks. More generally, we believe that preserved clique patterns are a priori interesting (i. e., independently from the application context). The possibility to exploit other user-defined constraints supports the discovery of actionable patterns.

7 Related Work

The harder problem of extracting a complete collection of closed 3-sets directly from real-valued tensors (e. g., rough kinetic microarray datasets) is not discussed here. To the best of our knowledge, Data-Peeler only faces two competitors able to extract all closed 3-sets from ternary relations: CubeMiner [17] and Trias [16]. None of them have the generality of Data-Peeler. In particular, they cannot deal with n-ary relations and cannot enforce any piecewise (anti)-monotone constraints. This latter drawback makes them harder to specialize in the extraction of δ-contiguous closed 3-cliques. Furthermore, [11] shows that Data-Peeler outperforms both of them by orders of magnitude. The interested reader will refer to the "Related Work" section of that article for a detailed analysis of what makes Data-Peeler more efficient than both CubeMiner and Trias.

Extracting every clique in a single graph is a classical problem [7] and algorithms with polynomial delay were designed to extract the maximal (i. e., closed) ones (e. g., [19]). Collections of large graphs were built to help in understanding genetics. These graphs commonly have tens of thousands of nodes and are much noisy. For about four years, extracting knowledge by

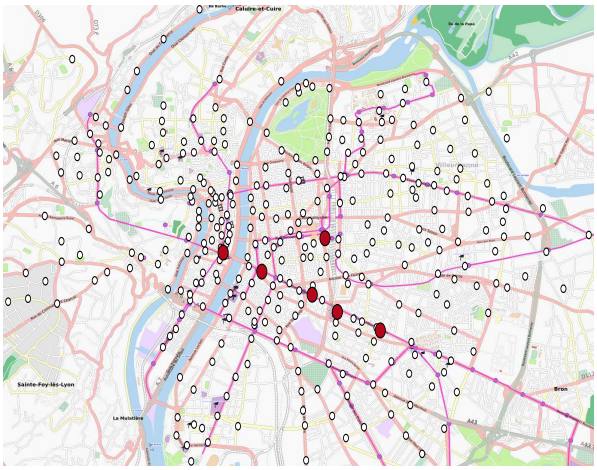

Fig. 8a A 0.5-contiguous closed 3-clique with $T = \{18.5, 19, 19.5, 20, 20.5\}$

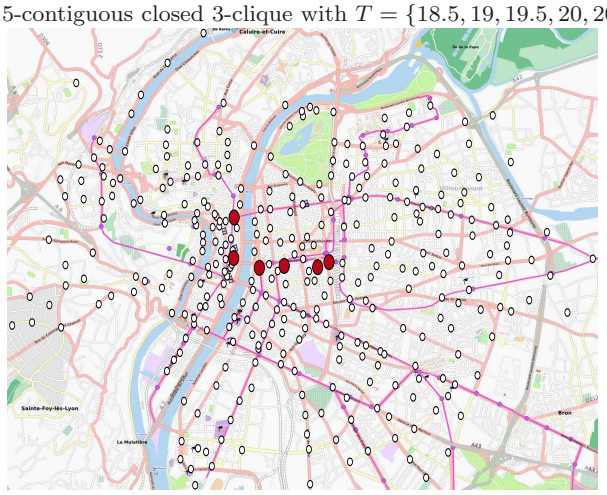

Fig. 8b A 1-contiguous closed 3-clique with $T = \{16, 17, 17.5, 18.5\}$

crossing such graphs has been a hot topic. For example, there is a need to extract patterns that remain valid across several co-expression graphs obtained from microarray data or to cross the data pertaining to physical interactions between molecules (e. g., protein-protein, protein-gene) with more conceptual data (e. g., co-expression of genes, co-occurrence of proteins in the literature). One of the most promising pattern helping in these tasks is the closed 3-clique or, better, the closed quasi-3-clique. CLAN [27] is able to extract closed 3-cliques from collections of large and dense graphs. Crochet+ [18], Cocain* [29] and Quick [21] are the state-of-the-art extractors of closed quasi-3-cliques. They all use the same definition of noise tolerance: every node implied in a pattern must have, in every graph independently from the others, a degree

exceeding a user-defined proportion of the maximal degree it would reach if the clique was exact.

As detailed in [9], Data-Peeler can be generalized towards the tolerance of noise. Combining it with the present work enables the extraction of closed quasi-3-cliques. However the chosen definition for noise tolerance being defined on any n-ary relation, it is different from that of the approaches cited in the previous paragraph. Indeed this tolerance applies to every node across all graph (to be part of a quasi-3-clique, a node must be *globally* much connected to the other nodes of the pattern) and to the graphs themselves. As a consequence our approach does not scale well to graphs connecting thousands of nodes but it can extract closed quasi-3-cliques in large collections of smaller graphs, whereas the previously presented approaches cannot (or they must be used with a very strong minimal size constraint on the number of involved graphs). When the graphs are collected along an ordered dimension (typically the time), the use of the δ-contiguity constraint further increases this difference. Notice that the previous approaches focus on collections of undirected graphs, whereas our approach works on (possibly) directed graphs.

The δ-contiguity stems from an analogous constraint, called *max-gap* constraint, initially applied to sequence mining. It was introduced in the GSP approach [24]. The way the δ-contiguity is enforced in our approach (see Sect. 4) is similar to that of this seminal article. The *min-gap* and the *window size* constraints [24] uses could as well be enforced in our approach. Nevertheless, in [24], these constraints modify the enumeration order, whereas, in our approach, they reduce the search space and let the enumeration strategy unaltered. Furthermore, the nature of the mined patterns is much different. In the context of [24], the considered datasets are multiple sequences of itemsets and the extracted patterns are sub-sequences of itemsets whose order (but not position in time) is to be respected in all (1-dimensional) supporting sequences. In our approach, the supporting domain contains (2-dimensional) graphs and their position in time must be aligned.

Notice that the max-gap constraint was used in other contexts too. For example, [8] enforces it to extract episodes (repetition of sub-sequences in one sequence) and [14] somehow aggregates the two tasks by extracting, under a max-gap constraint, frequent sub-sequences whose support is the sum of the number of repetitions in all sequences of the dataset. Finally let us notice that an extended abstract of this chapter was previously published [13].

8 Conclusion

This chapter focuses on specializing the Data-Peeler closed n-set extractor to mine δ-contiguous closed 3-cliques. All the additional constraints imposed to achieve this goal were piecewise (anti)-monotone. Hence, in its original form, Data-Peeler could handle them all. However, to be able to extract

δ-contiguous closed 3-cliques from large dynamic graphs (e. g., hundreds of nodes and of timestamps), ad-hoc strategies must be used. Interestingly, the idea is the same for all of them (and for the connection constraint too): they must be used as soon as possible in the enumeration tree. The symmetry constraint has even been split into many small constraints that are individually exploited as soon as possible. These constraints are particular since they change the structure of the enumeration which does not follow a binary tree anymore. This chapter focuses on the extraction of δ-contiguous closed 3-cliques. However, Data-Peeler is not restricted to it. It can mine closed n-sets (or cliques) with n an arbitrary integer greater or equal to 2, it can force the contiguity of the patterns on several dimensions at the same time (possibly with different δ values), etc. Furthermore, Data-Peeler can mine closed n-sets adapted to any specific problem that can be expressed in terms of piecewise (anti)-monotone constraints.

Acknowledgments. This work has been partly funded by EU contract IST-FET IQ FP6-516169, and ANR Bingo2 (MDCO 2007-2010). Tran Bao Nhan Nguyen has contributed to this study thanks to a Research Attachment programme between the Nanyang Technological University (Singapore), where he is an undergraduate student, and INSA-Lyon. Finally, we thank Dr. J. Besson for exciting discussions.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD'93: Proc. SIGMOD Int. Conf. on Management of Data, pp. 207–216. ACM Press (1993)
2. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI/MIT Press (1996)
3. Antonie, M.L., Zaïane, O.R.: Mining positive and negative association rules: An approach for confined rules. In: PKDD'04: Proc. European Conf. on Principles and Practice of Knowledge Discovery in Databases, pp. 27–38. Springer (2004)
4. Besson, J., Robardet, C., Boulicaut, J.F., Rome, S.: Constraint-based formal concept mining and its application to microarray data analysis. Intelligent Data Analysis **9**(1), 59–82 (2005)
5. Boulicaut, J.F., Besson, J.: Actionability and formal concepts: A data mining perspective. In: ICFCA'08: Proc. Int. Conf. on Formal Concept Analysis, pp. 14–31. Springer (2008)
6. Boulicaut, J.F., De Raedt, L., Mannila, H. (eds.): Constraint-Based Mining and Inductive Databases, *LNCS*, vol. 3848. Springer (2006)
7. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph (algorithm 457). Communications of the ACM **16**(9), 575–576 (1973)
8. Casas-Garriga, G.: Discovering unbounded episodes in sequential data. In: PKDD'03: Proc. European Conf. on Principles and Practice of Knowledge Discovery in Databases, pp. 83–94. Springer (2003)
9. Cerf, L., Besson, J., Boulicaut, J.F.: Extraction de motifs fermés dans des relations n-aires bruitées. In: EGC'09: Proc. Journées Extraction et Gestion de Connaissances, pp. 163–168. Cepadues-Editions (2009)

10. Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Data-Peeler: Constraint-based closed pattern mining in n-ary relations. In: SDM'08: Proc. SIAM Int. Conf. on Data Mining, pp. 37–48. SIAM (2008)
11. Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Closed patterns meet n-ary relations. ACM Trans. on Knowledge Discovery from Data **3**(1) (2009)
12. Cerf, L., Mougel, P.N., Boulicaut, J.F.: Agglomerating local patterns hierarchically with ALPHA. In: CIKM'09: Proc. Int. Conf. on Information and Knowledge Management, pp. 1753–1756. ACM Press (2009)
13. Cerf, L., Nguyen, T.B.N., Boulicaut, J.F.: Discovering relevant cross-graph cliques in dynamic networks. In: ISMIS'09: Proc. Int. Symp. on Methodologies for Intelligent Systems, pp. 513–522. Springer (2009)
14. Ding, B., Lo, D., Han, J., Khoo, S.C.: Efficient mining of closed repetitive gapped subsequences from a sequence database. In: ICDE'09: Proc. Int. Conf. on Data Engineering. IEEE Computer Society (2009)
15. Ganter, B., Stumme, G., Wille, R.: Formal Concept Analysis, Foundations and Applications. Springer (2005)
16. Jaschke, R., Hotho, A., Schmitz, C., Ganter, B., Stumme, G.: Trias–an algorithm for mining iceberg tri-lattices. In: ICDM'06: Proc. Int. Conf. on Data Mining, pp. 907–911. IEEE Computer Society (2006)
17. Ji, L., Tan, K.L., Tung, A.K.H.: Mining frequent closed cubes in 3D data sets. In: VLDB'06: Proc. Int. Conf. on Very Large Data Bases, pp. 811–822. VLDB Endowment (2006)
18. Jiang, D., Pei, J.: Mining frequent cross-graph quasi-cliques. ACM Trans. on Knowledge Discovery from Data **2**(4) (2009)
19. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: On generating all maximal independent sets. Information Processing Letters **27**(3), 119–123 (1988)
20. Leskovec, J., Kleinberg, J.M., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. ACM Trans. on Knowledge Discovery from Data **1**(1) (2007)
21. Liu, G., Wong, L.: Effective pruning techniques for mining quasi-cliques. In: ECML PKDD'08: Proc. European Conf. on Machine Learning and Knowledge Discovery in Databases - Part II, pp. 33–49. Springer (2008)
22. Mannila, H., Toivonen, H.: Multiple uses of frequent sets and condensed representations. In: KDD, pp. 189–194 (1996)
23. Pei, J., Han, J., Mao, R.: CLOSET: An efficient algorithm for mining frequent closed itemsets. In: SIGMOD'00: Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 21–30. ACM Press (2000)
24. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT'96: Proc. Int. Conf. on Extending Database Technology, pp. 3–17. Springer (1996)
25. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with TITANIC. Data & Knowledge Engineering **42**(2), 189–222 (2002)
26. Uno, T., Kiyomi, M., Arimura, H.: LCM ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In: OSDM'05: Proc. Int. Workshop on Open Source Data Mining, pp. 77–86. ACM Press (2005)
27. Wang, J., Zeng, Z., Zhou, L.: CLAN: An algorithm for mining closed cliques from large dense graph databases. In: ICDE'06: Proc. Int. Conf. on Data Engineering, pp. 73–82. IEEE Computer Society (2006)
28. Zaki, M.J., Hsiao, C.J.: CHARM: An efficient algorithm for closed itemset mining. In: SDM'02: Proc. SIAM Int. Conf. on Data Mining. SIAM (2002)
29. Zeng, Z., Wang, J., Zhou, L., Karypis, G.: Out-of-core coherent closed quasi-clique mining from large dense graph databases. ACM Trans. on Database Systems **32**(2), 13–42 (2007)