Mining Spatiotemporal Patterns in Dynamic Plane Graphs

Adriana Prado[1] Baptiste Jeudy[2] Elisa Fromont[2] Fabien Diot[2]

[1] Université de Lyon, CNRS, INSA-Lyon, LIRIS,
UMR5205, F-69621, France.
adriana.bechara-prado@insa-lyon.fr

[2] Université de Lyon, Université de St-Etienne F-42000,
UMR CNRS 5516, Laboratoire Hubert-Curien, France.
{baptiste.jeudy,elisa.fromont,fabien.diot}@univ-st-etienne.fr

Abstract

Dynamic graph mining is the task of searching for subgraph patterns that capture the evolution of a dynamic graph. In this paper, we are interested in mining dynamic graphs in videos. A video can be regarded as a dynamic graph, whose evolution over time is represented by a series of plane graphs, one graph for each video frame. As such, subgraph patterns in this series may correspond to objects that frequently appear in the video. Furthermore, by associating spatial information to each of the nodes in these graphs, it becomes possible to track a given object through the video in question. We present, in this paper, two plane graph mining algorithms, called Plagram and DyPlagram, for the extraction of spatiotemporal patterns. A spatiotemporal pattern is a set of occurrences of a given subgraph pattern which are not too far apart w.r.t time nor space. Experiments demonstrate that our algorithms are effective even in contexts where general-purpose algorithms would not provide the complete set of frequent subgraphs. We also show that they give promising results when applied to object tracking in videos.

1 Introduction

Graph mining is a popular data mining task with many applications in, for example, analysis of (social, gene, protein) networks. The idea of this task is to search for all subgraphs that frequently appear in a dataset of graphs or even within a single large graph. In some cases, however, this dataset may consist of a series of graphs representing the evolution of a single graph over time, that is,

a dynamic graph. Here, frequent subgraphs must capture the evolution of such graph, such as the insertion or deletion of edges or nodes.

In this paper, we are interested in mining dynamic graphs applied to videos. We regard a video as a dynamic graph, whose evolution over time is represented by a series of graphs, one graph for each video frame. More precisely, we represent each frame as a *plane graph*, that is, a planar graph already drawn in the plane without edge intersections, by means of region adjacency relationships [4] or Delaunay triangulation. In the former case, e.g., the barycenters of the different regions in a frame are nodes in the corresponding graph, and an edge exists between two nodes if the regions are adjacent in the frame. Some example video frames along with their respective triangulated and region adjacency representations are illustrated in Figure 8.

By representing a video as a series of plane graphs, subgraph patterns in this series may correspond to objects that frequently appear in a video, such as the airplanes in the frames of Figure 8. In addition, by associating spatial information to each node in these graphs, such as the barycenter of the originating frame regions, it becomes possible to track a given object through the video being mined.

In this context, we present two graph mining algorithms, called PLAGRAM (**Pla**ne **Gra**ph **M**ining) and DYPLAGRAM (**Dy**namic **Pla**ne **Gra**ph **M**ining), which are dedicated to mine frequent plane subgraphs from a database of plane graphs. One of the main characteristics of these algorithms is that they can be used as the basis for the extraction of what we called *spatiotemporal patterns*. A spatiotemporal pattern is a set of occurrences of a given pattern which are not too far apart w.r.t time nor space.

We have therefore divided the paper in the following way. The next section is dedicated to the related work on frequent (dynamic) graph mining. In Section 3, we formally define the problem studied in this paper. The algorithms PLAGRAM and DYPLAGRAM, along with a complexity study, are presented in Section 4. In Section 5, we report on some experiments on the efficiency of the proposed algorithms. We also discuss their usefulness in tackling the problem of object tracking in videos. We conclude in Section 6.

2 Related Work

Typical frequent graph mining algorithms generate plausible *pattern* subgraphs and then compute their frequency while finding all their occurrences in a database of *target* graphs. Afterwards, frequent patterns (w.r.t. a user-defined threshold) are extended in a valid way such that bigger patterns can also be evaluated.

Due to the numerous necessary NP-complete subgraph isomorphism tests, current graph mining approaches, such as GASTON [15], GSPAN [18], FSG [13], and AGM [9] can only deal with applications where the subgraph isomorphism test is not too costly or when there are not too many such tests, e.g., when the target graphs are small, their nodes have low degree, have many node (or edge) labels, have a low number of cycles, etc.

Indeed, most of the recent work on the subject has been dedicated to finding subclasses of graphs (outerplanar graphs [8], graphs with a different label for each node [12], acyclic graphs, sparse graphs etc.), which lead to more efficient algorithms. Following the same line, the focus of our algorithms, PLAGRAM and DYPLAGRAM, is on plane subgraphs. Plane graphs are particularly interesting since the subgraph isomorphism test for this type of graphs is known to be polynomial (see, for example, [5]). In addition, we impose a restriction on graph extensions such that a graph can only be extended with faces. This technique is related to that used in AGM [9], which searches for all frequent node-induced subgraphs. Although also improving the efficiency of the extension building phase, this technique is still less efficient than GSPAN and our algorithms.

Yet, current general-purpose graph mining algorithms do not computationally benefit from the plane property of target graphs and therefore cannot tackle in a satisfactory way our video application. As pointed out in the experimental section, the popular algorithm GSPAN could not finish its executions on our video datasets within 3 days of computation. In addition, we observed that the extension building phase of GSPAN was, on average, 90% of its total execution time. This is due to the fact that GSPAN extends the pattern graphs by adding a single edge at a time, which led to an extremely high number of extensions, even for the lowest tested minimum frequency.

Regarding the research on dynamic graph mining, current algorithms consider a dynamic graph with only edges insertions or deletions, i.e., the time series of graphs share the same set of nodes over time (see, e.g.,[14]), or in which nodes and edges are only added and never deleted (see, e.g., [2]). In our approach, however, there is no information about the correspondence between the nodes in one graph (video frame) and those in the others. Instead, we know, for each node in a plane graph, the barycenter of its originating frame region. This allows us to mine for spatiotemporal patterns in the series of graphs, as presented in the introduction.

Our algorithms borrow many features from the algorithm GSPAN. GSPAN performs a depth-first search in a space of canonical codes, which are computed such that two isomorphic graphs are not evaluated twice. One of the most acknowledged bottleneck of this algorithm (we show in the experimental section that this is not the main one) comes from the subgraph isomorphism tests. In particular, GSPAN is not well suited to mine graphs with many cycles as their presence increases exponentially the number of frequent subgraphs. However, the particular plane subgraphs considered in this paper are far less numerous than the subgraphs considered by GSPAN, which makes our approach not only unsurprisingly faster, but also capable of dealing with much more complex graphs (in terms of degrees and size) than GSPAN.

The algorithm PLAGRAM was introduced in [16]. Here, we extend it to DYPLAGRAM, in the context of dynamic graphs, and show how both algorithms can be applied to mine frequent spatiotemporal patterns. To the best of our knowledge, the papers that are the closest to ours are [6] and [11]. In the former, the authors use combinatorial maps to represent images and propose an algorithm to mine frequent submaps. Although we use similar structures to represent

video frames, i.e., plane graphs, our work goes much further since we consider dynamic aspects applied to videos. In [11], the authors use the algorithm SUBDUE [7] to extract the background from videos filmed by a static surveillance camera. They assume that the background appears more frequently than the foreground in such videos. Differently from our approach, a video is represented by one single graph (dynamic aspects are not taken into account), and the goal is to extract only the most relevant subgraph w.r.t a ranking measure based on the MDL principle. In our approach, we are interesting in identifying the most interesting objects w.r.t their frequency in a dynamic environment.

3 Definitions

3.1 Plane graphs

We use ordered graphs to represent plane graphs. Ordered graphs are not necessarily planar, but here we restrict ourselves to planar ordered graphs [10].

Definition 1 (Labeled ordered graph) *A labeled ordered graph* $G = (V, N, L_e, L_n)$ *is a set of nodes* V *and three functions* N, L_e, *and* L_n. *For each node* v, $N(v)$ *is the circular ordered list* $\langle v_0, v_1, \ldots v_{k-1} \rangle$ *of its* k *neighbors. If* $u \in N(v)$ *then* $v \in N(u)$ *and* (u, v) *is called an edge of the graph. The labeling functions* L_e *and* L_n *map, respectively, each edge and each node of the graph to a label.*

Definition 2 (Plane graph) *Given a planar embedding of a labeled graph, a labeled ordered graph is constructed by defining* $N(v)$ *as the list of neighbors of* v *in an anti-clockwise order around* v. *This labeled ordered graph is called a* plane graph.

Definition 3 (Face) *Given a plane graph, a* face *is a connected region of the plane which is bounded by a circuit of edges. It is represented by the list of nodes encountered when following the circuit such that the face is always on the left-hand side. The unbounded region in the embedding of the graph is called the* outer face *of the graph. The other faces are called* internal faces.

Example 1 *Figure 1 presents three plane graphs and Table 1 gives their lists of neighbors. Notice that since these lists are circular, any of their circular permutations is also valid. Graph* g_1 *has two internal faces* $\langle 1, 2, 3 \rangle$ *and* $\langle 2, 4, 5, 3 \rangle$, *and its outer face is* $\langle 1, 3, 5, 4, 2 \rangle$.

3.2 Plane Subgraph Isomorphism

A plane graph is a plane subgraph of another plane graph if there exists a correspondence between their nodes which preserves the labels, the edges and also the internal faces (if the outer face is also preserved, then the graphs are plane isomorphic).

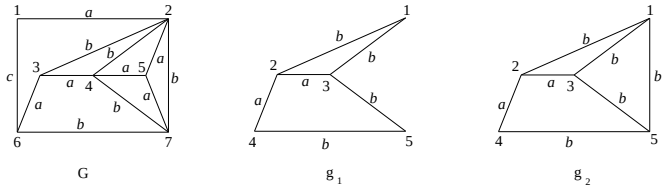

Figure 1: Plane graphs. The edge labels are in $\{a, b, c\}$ and we assume that all node labels are equal to a (not represented). Graph g_1 is a plane subgraph of G while g_2 is not.

	G	g_1	g_2
$N(1) =$	$\langle 2, 6\rangle$	$\langle 2, 3\rangle$	$\langle 2, 3, 5\rangle$
$N(2) =$	$\langle 1, 3, 4, 5, 7\rangle$	$\langle 1, 4, 3\rangle$	$\langle 1, 4, 3\rangle$
$N(3) =$	$\langle 2, 4, 6\rangle$	$\langle 1, 2, 5\rangle$	$\langle 1, 2, 5\rangle$
$N(4) =$	$\langle 2, 3, 7, 5\rangle$	$\langle 2, 5\rangle$	$\langle 2, 5\rangle$
$N(5) =$	$\langle 2, 4, 7\rangle$	$\langle 3, 4\rangle$	$\langle 1, 3, 4\rangle$
$N(6) =$	$\langle 1, 7, 3\rangle$		
$N(7) =$	$\langle 2, 5, 4, 6\rangle$		

Table 1: Neighbor lists of the graphs in Figure 1.

Definition 4 (Plane subgraph isomorphism) *Let $G = (V, N, L_e, L_n)$ and $G' = (V', N', L'_e, L'_n)$ be two plane graphs. Graph G' is* plane subgraph isomorphic *to G (or G' is a* plane subgraph *of G), denoted $G' \subseteq G$, if there is an injective function f from V' to V such that: for all nodes u of G', $L_n(f(u)) = L'_n(u)$, for all edges (u, v) of G', $L_e((f(u), f(v))) = L'_e(u, v)$, and for all internal faces $F = \langle v_1, ..., v_k\rangle$ of G', $f(F) = \langle f(v_1), ..., f(v_k)\rangle$ is a face of G.*

Definition 5 (Occurrence of a plane graph in a larger graph) *Let $G = (V, N, L_e, L_n)$ and $G' = (V', N', L'_e, L'_n)$ be two plane graphs. If G' is plane subgraph isomorphic to G, the corresponding injective function f is called an* occurrence *of G' in G.*

Example 2 *In Figure 1, graph g_1 is a plane subgraph of G. The internal faces $\langle 1, 2, 3\rangle$ and $\langle 2, 4, 5, 3\rangle$ of g_1 correspond, respectively, to faces $\langle 2, 3, 4\rangle$ and $\langle 3, 6, 7, 4\rangle$ of G, with $f(1) = 2$, $f(2) = 3$, $f(3) = 4$, $f(4) = 6$ and $f(5) = 7$. Graph g_2 has three internal mutually adjacent faces, one with four edges and two with three edges. Since such configuration of faces does not exist in G, g_2 is not a plane subgraph of G.*

3.3 Dynamic Graphs and Spatiotemporal Patterns

Definition 6 (Dynamic graph) *A dynamic graph $\mathcal{D}$ is an ordered set of plane graphs $\{G_1, G_2, .., G_n\}$ in which each node of these graphs is associated to spatial coordinates (x, y) and a weight w.*

Example 3 *In our video application, each plane graph G_i represents a video frame. Each node in a graph represents a segmented frame region, and is associated to the coordinates (x, y) of the barycenter of the pixels of this region along with its size in pixels (weight).*

Definition 7 (Occurrences of a plane graph in a dynamic graph) *Given a plane graph P and a dynamic graph $\mathcal{D} = \{G_1, ..., G_n\}$, the set of occurrences of P in $\mathcal{D}$ is defined as $Occ(P) = \{(i, f) \mid f$ is an occurrence of P in $G_i\}$.*

The spatial coordinates (x_o, y_o) of an occurrence $o = (i, f) \in Occ(P)$ is defined as the weighted average of the spatial coordinates (x, y) of the nodes in $f(P)$.

Definition 8 (Frequency of a plane graph in a dynamic graph) *The frequency $freq(P)$ of a plane graph P in a dynamic graph $\mathcal{D}$ is the number of graphs $G_i \in \mathcal{D}$ in which there is an occurrence of P, i.e., $|\{i \mid \exists f,\ (i, f) \in Occ(P)\}|$.*

In typical subgraph mining problems, where the input collection of graphs does not represent a dynamic graph, the frequency of a pattern graph P is computed regardless of the fact that its occurrences may be far apart w.r.t. time and/or space. In the context of dynamic graphs, we define in the following the notion of *spatiotemporal pattern* in which occurrences of the same pattern that are close to one another are aggregated.

Definition 9 (Spatiotemporal pattern) *Two occurrences of a plane graph P in a dynamic graph $\mathcal{D}$, $o = (i, f)$ and $o' = (i', f')$, are close if the distance between their coordinates is lower than a spatial threshold ϵ and their time distance $|i' - i|$ is lower than a time threshold τ. Then, given a plane graph P and a dynamic graph $\mathcal{D}$, we define the* occurrence graph *of P as a graph where the set of nodes is $Occ(P)$ and the set of edges is $\{(o, o') \mid o$ is close to $o'\}$. Each connected component of the occurrence graph of P is a spatiotemporal pattern S based on P.*

Definition 10 (Frequency of a spatiotemporal pattern) *The frequency of a spatiotemporal pattern S based on a plane graph P in a dynamic graph $\mathcal{D}$, denoted $freq_{st}(P, S)$, is $|\{i \mid \exists f,\ (i, f) \in S\}|$.*

Example 4 *Figure 2 shows 7 occurrences of a pattern P in a video with four frames. Therefore, $freq(P) = 4$. Since occurrences 1 and 2 are close to each other, i.e., they appear in consecutive frames at a similar position, there is an edge $(1, 2)$ in the occurrence graph of P. Conversely, the edge $(1, 5)$ does not exist in the occurrence graph, as occurrences 1 and 5 are far from each other. There are 3 spatiotemporal patterns $S_1 = \{1, 2, 3\}$, $S_2 = \{5, 6, 7\}$, and $S_3 = \{4\}$. The frequencies of these patterns are: $freq_{st}(P, S_1) = freq_{st}(P, S_2) = 3$, and $freq_{st}(P, S_3) = 1$.*

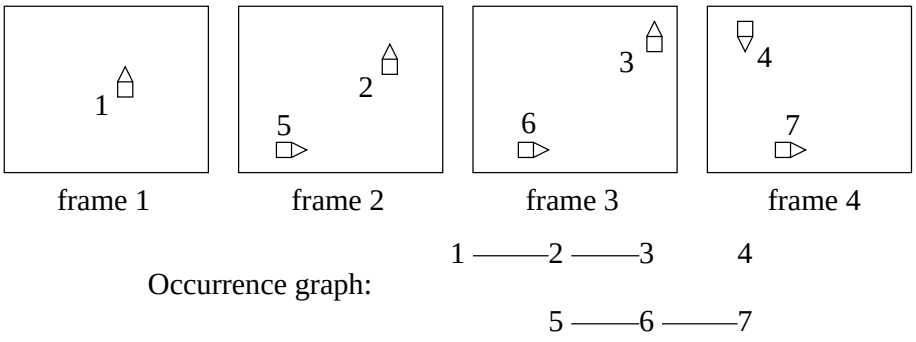

Figure 2: Occurrences of a pattern and occurrence graph.

3.4 Problem Definition

Given a frequency threshold σ (also called minimum support), a spatial threshold ϵ and a time threshold τ, the problem we tackle in this paper is to compute all spatiotemporal patterns with freq_{st} greater than σ.

4 Plagram and DyPlagram Algorithms

Given two plane graphs such that $P' \subseteq P$, if there is an occurrence of P in G_i, then there is also an occurrence of P' in G_i. Thus $\text{freq}(P) \leq \text{freq}(P')$ and, therefore, if P' is not frequent, then neither is P. Given this behavior, we say that freq has the anti-monotonicity property. Such property can certainly be exploited to prune non-promising candidate subgraphs, as in classical graph mining algorithms.

However, freq_{st} is not anti-monotone. Suppose that two occurrences a and b of P are close to each other, leading to a single frequent spatiotemporal pattern S. Conversely, two occurrences $a' \subseteq a$ and $b' \subseteq b$ of P' may be far from each other, possibly resulting in two non-frequent spatiotemporal patterns S' and S''. In other words, two spatiotemporal patterns S' and S'' based on P' may be infrequent, while the spatiotemporal pattern S based on P is frequent.

Nevertheless, the frequency of a spatiotemporal pattern S based on a plane graph P (i.e., $\text{freq}_{st}(P, S)$) can be upper bounded with two anti-monotone measures as follows:

$$\text{freq}_{st}(P, S) \leq \text{freq}_{seq}(P) \leq \text{freq}(P),$$

where $\text{freq}_{seq}(P)$ is the *subsequence frequency* of P defined below.

Definition 11 (Subsequence frequency) *The subsequence frequency of a plane graph P in $\mathcal{D}$, denoted $freq_{seq}(P)$, is defined as the size of the longest subsequence G_{i_1},G_{i_2},..., $i_1 < i_2 < ...$ of $\mathcal{D}$ such that (a) for all j, G_{i_j} contains an occurrence of P and (b) for all j, $i_{j+1} - i_j$ is lower than the time threshold τ.*

Observe that $\text{freq}_{seq}(P)$ is an upper-bound on $\text{freq}_{st}(P, S)$, since the sequence of the G_is that contain an occurrence of S satisfies (a) and (b) in Definition 11. Moreover, if $P' \subseteq P$ then any sequence of G_is satisfying (a) and (b) for pattern P also satisfies them for pattern P'. $\text{freq}_{seq}(P) \leq \text{freq}_{seq}(P')$ and therefore freq_{seq} has the anti-monotonicity property.

To solve the problem defined in Section 3.4, the idea is to first mine for all frequent pattern graphs (using either freq or freq_{seq}) and then, in a post-processing step, construct the occurrence graph of each frequent pattern to compute the spatiotemporal patterns, as described in Definition 9. The algorithm that uses freq is called PLAGRAM and the one that uses freq_{seq} is DYPLAGRAM.

4.1 Extensions

Our algorithms use a depth-first exploration strategy: each time a frequent pattern is found, it is extended into a bigger candidate pattern for further evaluation. As GSPAN, our algorithms only generate promising candidate graphs, that is, subgraphs that actually occur in $\mathcal{D}$. However, our extension strategy limits the number of different extensions that can be generated from a given frequent pattern, as described below.

Definition 12 (Valid extension) *Given a plane graph g and two nodes $u \neq v$ on the outer face of g, g can only be extended by the addition of a new path $P = (u = x_1, x_2, \ldots, x_k = v)$ to g between u and v. This path must lie in the outer face of g. Nodes x_2,..., x_{k-1} are $(k-2) \geq 0$ new nodes with $N(x_i) = \langle x_{i-1}, x_{i+1} \rangle$. This new graph is denoted $g \cup P$. Given a plane graph G such that $g \subset G$, P is a* valid extension *of g in G if $g \cup P \subseteq G$.*

In other words, this definition states that any pattern graph g composed of aggregated faces can only be extended by the addition of another face lying in the outer face of g. This new face must share at least one edge with g (since $u \neq v$). A consequence of this extension strategy is that the generated patterns are always 2-connected (this means that for any two nodes of the pattern, there is always a cycle that contains both).

Example 5 *In Figure 1, there is only one occurrence of g_1 in G and, for this occurrence, there are three valid extensions of g_1 in G. Since these extensions have two edges, a new node 6 is added in the outer face of g_1. The extensions are: $P_1 = (1, 6, 3)$ (which corresponds to $2, 5, 4$ in G), $P_2 = (3, 6, 5)$ (corresponding to $4, 5, 7$ in G) and $P_3 = (4, 6, 1)$ (corresponding to $6, 1, 2$ in G). Observe that the path $P_4 = (1, 5)$ is not a valid extension since $g_1 \cup P_4$ is the graph g_2, which is not a plane subgraph of G (see Section 3.2).*

Given a pattern graph g and a graph G_i in $\mathcal{D}$, our algorithms compute all occurrences of g in G_i. Then, for each occurrence, they generate all possible extensions. For each occurrence of g in G_i and from each node of the external face of g, there is only one possible extension. This is one reason why PLAGRAM and DYPLAGRAM generate fewer extensions than GSPAN, as we show in

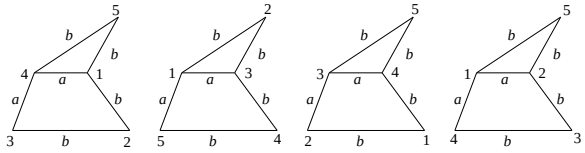

Figure 3: Four copies of g_1 of Figure 1 with node indices corresponding, respectively, to the codes α, β, γ, and δ in Table 2.

Edge	α	β	γ	δ
1	(1,2,a,b,a)	(1,2,a,b,a)	(1,2,a,b,a)	(1,2,a,a,a)
2	(2,3,a,b,a)	(2,3,a,b,a)	(2,3,a,a,a)	(2,3,a,b,a)
3	(3,4,a,a,a)	(3,1,a,a,a)	(3,4,a,a,a)	(3,4,a,b,a)
4	(4,1,a,a,a)	(3,4,a,b,a)	(4,1,a,b,a)	(4,1,a,a,a)
5	(4,5,a,b,a)	(4,5,a,b,a)	(3,5,a,b,a)	(1,5,a,b,a)
6	(5,1,a,b,a)	(5,1,a,a,a)	(5,4,a,b,a)	(5,2,a,b,a)

Table 2: Four valid codes for graph g_1.

Section 5. gSpan extends pattern graphs edge by edge, and several extensions may be generated from one node.

4.2 Graph Codes

To avoid multiple generations of the same pattern, pattern graphs are represented by canonical codes. Therefore, to find the frequent patterns, our algorithms explore a code search space.

A code for a plane graph g is a sequence of the edges of g. Each edge is represented by a 5-tuple $(i, j, L_n(i), L_e(i, j), L_n(j))$, where i and j are the indices of the nodes (from 1 to n, where n is the number of nodes in g). The nodes are numbered as they first appear in the code.

Definition 13 (Valid code for a plane graph) *If $g = (V, N, L_n, L_e)$ is a plane graph with only one internal face $\langle v_0, ..., v_{n-1}\rangle$ (i.e., g is a cycle), then a valid code for g is $(1, 2, L_n(1), L_e(1, 2), L_n(2)).(2, 3, \ldots)$, $(3, 4, \ldots)\ldots$, $(n - 1, n, \ldots).(n, 1, \ldots)$. We use a "dot" to denote the concatenation of each 5-tuple representing an edge of g. If $g = g' \cup P$ and P is a valid extension of g' in g, then a valid code for g is the concatenation of a valid code for g' and the code of P.*

It is not obvious from Definition 13 that every 2-connected plane graph g has at least one valid code. Indeed, since g is 2-connected, it is always possible to construct a valid code by first choosing an internal face of g and then iteratively adding valid extensions to it.

Example 6 *Table 2 shows four valid codes of graph g_1 in Figure 1 (among seven valid codes). Figure 3 shows the corresponding node numbering on graph g_1 (recall that there is a different numbering of nodes for each code). Codes α,*

γ, δ start with the 4-edge face and then a 2-edge extension is added to build the second face. Code β starts with the 3-edge face and then a 3-edge extension is added. In each column, the line separates the edges of the first face from the edges of the valid extension. A valid code for this graph can start with any of the six edges. For the edge that belongs to the two internal faces, the code can start with any of the two faces, hence the seven possible codes.

4.3 Code Search Space and Canonical Codes

The set of valid codes is organized in a *code tree*. A code C' is a child of C in the code tree if there is a valid extension P of C such that C' is the concatenation of C with the codes of the edges of P. The root of the code tree is the empty code.

Example 7 *An example tree rooted at code α (of Table 2) is represented in Figure 4. Notice that the codes at a given level of the tree represent graphs that have one more face than the codes of the level just above. In this code tree, each graph is represented by several codes (for instance, we have already seen that graph g_1 has seven valid codes). In Figure 4 we also see that codes $\alpha.A.D$ and $\alpha.C.F$ represent the same graph.*

Naturally, exploring several codes that represent the same graph is not efficient. We therefore define *canonical codes* such that each graph has exactly one such code: we start by defining an order on the valid codes. We assume that there exists an order on the labels. Then, we define an order on the edges by taking the lexicographic order derived from the natural order on node indices and the order on labels. It means that $(i, j, L_n(i), L_e(i, j), L_n(j)) < (x, y, L_n(x), L_e(x, y), L_n(y))$ if $i < x$ or ($i = x$ and $j < y$) or ($i = x$ and $j = y$ and $L_n(i) < L_n(x)$), and so on. Afterwards, we extend this order on edges to a lexicographic order on the codes. We thus define the *canonical code* of a graph as the biggest code that can be constructed for this graph.

Definition 14 (Canonical code for a plane graph) *The* canonical code *of a plane graph is defined as the biggest valid code that can be constructed for this graph.*

Example 8 *In Figure 3, we assume that $a < b < c$. Therefore, $\alpha > \beta$ since they have the same first two edges and the third edge of β is smaller than the third edge of α. Because of the second edge, $\beta > \gamma$ and, finally, $\gamma > \delta$ since the first edge of γ is bigger than the first edge of δ. Code α is then the biggest code for graph g_1.*

Plagram and DyPlagram do a depth-first exploration of a code tree. The next theorem states that, if they find a non-canonical code C, then it is not necessary to explore the descendants of C; the whole subtree rooted at C can be safely pruned.

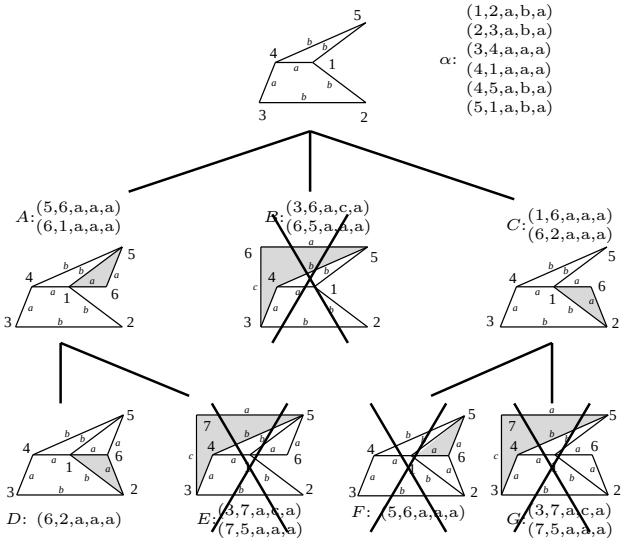

Figure 4: Part of the code tree starting from code α of Table 2. For each pattern, the gray face corresponds to the last added extension. The extension codes are A, ..., G (the complete code of the last line leftmost pattern is thus $\alpha.A.D$). The crossed codes are pruned since they are not canonical.

Theorem 1 *In the code search tree, if a code is not canonical, then neither are its descendants.*

Proof 1 *Let C be a non-canonical code of a graph G and $C.E$ a code of a descendant G' of G. Let C_c be the canonical code of G. As such, code C_c can be extended to a new code $C_c.F$ for G'. Since C_c is the canonical code of G, $C_c > C$ and thus $C_c.F > C.E$. Therefore, $C.E$ is not the biggest one and thus not canonical.*

Example 9 *In Figure 4, $\alpha.A.D$ and $\alpha.C.F$ are two codes for the same graph. Since $\alpha.A.D > \alpha.C.F$, any extension of $\alpha.A.D$ will be bigger than any extension of $\alpha.C.F$. Therefore, the latter code can be safely pruned.*

4.4 Pseudo-codes

The pseudo-codes of PLAGRAM and DYPLAGRAM are shown, respectively, in Figures 5 and 6. The overall outline is very similar to that of GSPAN. The main differences are the graph code used to represent a plane graph and the way extensions are generated. It is a depth-first recursive exploration of the code tree. Although the first level of the code tree contains codes representing graphs with one face, for efficiency reasons, PLAGRAM and DYPLAGRAM start their exploration with frequent edges. In both algorithms, the function *mine*

Algorithm: Plagram$(\mathcal{D}, \sigma)$
Input: graph database $\mathcal{D}$ and frequency threshold σ.
Output: plane subgraphs P in $\mathcal{D}$ such that freq$(P) > \sigma$.

```
1   Find all frequent edge codes in D
2   for all frequent edge code E do
3       mine(E,D,σ)
```

mine$(P,\mathcal{D}, \sigma)$
Input: the code of a pattern P, $\mathcal{D}$, and σ.

```
1   LE = ∅    //list of extensions of P
2   for all graph G_i ∈ D do
3       for all occurrences f of P in G_i do
4           LE = LE ∪ build_extensions(P, G_i, f)
5   for all extensions E in LE do
6       if freq(E) > σ then
7           if P is canonical then
8               output(P.E)
9               mine(P.E,D,σ)
```

Figure 5: Algorithm Plagram.

Algorithm: DyPlagram$(\mathcal{D}, \sigma, \tau)$
Input: graph database $\mathcal{D}$, frequency threshold σ, and time threshold τ.
Output: plane subgraphs P in $\mathcal{D}$ such that freq$_{seq}(P) > \sigma$.

```
1   Find all frequent edge codes in D
2   for all frequent edge code E do
3       mine(E,D,σ,τ)
```

mine$(P,\mathcal{D}, \sigma, \tau)$
Input: the code of a pattern P, $\mathcal{D}$, σ, and τ.

```
1   LE = ∅    //list of extensions of P
2   for all graph G_i ∈ D do
3       for all occurrences f of P in G_i do
4           LE = LE ∪ build_extensions(P, G_i, f)
5   for all extensions E in LE do
6       if freq_seq(E) > σ then
7           if P is canonical then
8               output(P.E)
9               mine(P.E,D,σ,τ)
```

Figure 6: Algorithm DyPlagram.

explores the part of the code tree rooted at a code given by its parameter. It computes their extensions on every target graph in $\mathcal{D}$ (lines 1-4) and make a recursive call on the frequent and canonical ones (line 9).

The difference between PLAGRAM and DYPLAGRAM is on the exploited frequency measure in function *mine* (line 6). The subsequence frequency used by DYPLAGRAM needs the time threshold τ, which defines the maximum gap allowed between two occurrences of a pattern (see Definition 11). Since $\text{freq} \geq \text{freq}_{seq}$, the number of extensions that are pruned (in line 6) is higher in DYPLAGRAM than in PLAGRAM.

Next, we present a complexity study of the main steps of function *mine*. We denote m the number of edges of a given pattern P, and m_i the number of edges of every target graph G_i.

Pattern matching (line 3): For each pattern P, function *mine* must find all occurrences of P in every target graph G_i. Each occurrence is found with a subgraph isomorphism test [5], which works as follows: first, it looks for an edge e of G_i that corresponds to the first edge of P. Once this match is performed, the complexity of matching the remaining edges of P is $O(m)$. So, the complexity of finding one occurence is, in the worst case, $O(m.m_i)$.

Function *mine* uses an optimization that makes this subgraph isomorphism test linear: it stores, along with pattern P, the list of edges e that match the first edge of P, in every target graph G_i. This list is updated in line 4 when generating the extensions. Therefore, to find the occurrences of P in G_i, it is not necessary to consider every edge of G_i, but only those in this list. In this way, for each occurrence, the cost of a matching becomes $O(m)$. Since the number of occurrences of P in a target graph G_i cannot be higher than $2m_i$ (the first edge of P may match each edge of G_i in two "directions"), the complexity of computing all occurrences of P in all target graphs G_i is $O(m \sum m_i)$ (which is bounded later by $O(\sum m_i^2)$ in Theorem 2).

Extension building (line 4): For every occurrence f of P in a target graph G_i, function *mine* builds all possible extensions. This is done by finding a valid extension starting from every node of the outer face of $f(P)$. The complexity of this operation is linear in the total size of P plus the size of the extensions. This is lower than $2m_i$ since one edge of G_i is either in $f(P)$ or in at most two of its extensions. Since there are at most $2m_i$ occurrences of P in G_i, the complexity of building all extensions of all occurrences of P in all target graphs G_i is $O(\sum m_i^2)$.

Every time a new extension is added to the list LE, its frequency is updated. This enables the test in line 6. In the case of DYPLAGRAM, the last value of i such that the extension appears in G_i is also stored for the computation of freq_{seq}. The LE list is implemented in a way such that the addition of a new extension (together with its frequency counting) is done with a logarithmic complexity (as a function of the number of edges of the extension). Therefore, for a fixed pattern P, we bound this complexity by the total size of all its

extensions in all G_is, i.e, by $O(\sum m_i^2)$.

According to the conducted experiments, the extension building step of function *mine* was found to be the most expensive.

Canonical test (line 7): This test is done by comparing code $P.E$ with the canonical code of the graph represented by $P.E$. Since two plane graphs are isomorphic if their canonical codes are the same, the complexity of this test is at least as high as an isomorphism test. The complexity of graph isomorphism, in the general case, is unknown, but for plane graphs, polynomial algorithms exist (see, for instance, [5] for a quadratic algorithm). Here is a sketch of our canonical test: the canonical code of a graph is constructed by first choosing a starting face and a starting edge in this face. Since a pattern P has m edges and considering that each edge belongs to at most two faces, there are at most $2m$ such choices. Then, the code is extended with the biggest valid extension code. Each of these steps has a complexity of $O(m)$ and must be repeated as many times as the number of faces in P, which is lower than m. Therefore, the complexity of finding the canonical code of a graph is, in the worst case, $O(m^3)$. Although not quadradic, experimental evaluations show that the canonical tests are not the bottleneck of our algorithms.

Theorem 2 (Complexity) *The total complexity of function* mine *(excluding the complexity of recursive calls in line 9) is $O(m^3 + \sum m_i^2)$, where m is the size of the pattern P (in number of edges) and m_i is the size of the target graph G_i (in number of edges).*

A consequence of this theorem is that, contrary to general graph mining algorithms as gSpan, Plagram and DyPlagram have a polynomial output delay, i.e., the time between the output of two frequent patterns is polynomial in the size of the input $\sum m_i$ (since, of course, $m < \sum m_i$).

Theorem 3 (Correctness) Plagram *and* DyPlagram *find and output exactly once all frequent 2-connected plane subgraphs in $\mathcal{D}$ (using, respectively, freq and $freq_{seq}$ as the frequency measure).*

Proof 2 *Since there is a one-to-one correspondence between canonical codes and 2-connected plane graphs, we must show that the algorithms do not miss any frequent canonical code. The algorithms prune a branch of the tree either because the code is not frequent (line 6) or because it is not canonical (line 7). The frequency of the descendants of a code C cannot be higher than the frequency of C. Therefore, if a code is not frequent, its descendants are not either, and thus the pruning step in line 6 is safe. If the code is not canonical, we know from theorem 1 that its descendants cannot be either. So, the pruning in line 7 is safe as well. In this way, the algorithms can never miss a frequent canonical code. Finally, every output code (line 8) is frequent and canonical and, since there is only one canonical code for each graph, a graph is output only once.*

Input: List of occurrences of P, frequency (freq_{st}) threshold σ, time threshold τ, and spatial threshold ϵ.
Output: frequent spatiotemporal patterns based on P.

1 The occurrence graph of P is empty.
2 **for all occurrences** (x, y, k) **do**
3 **for all** $0 < j \leq k$ and $k - j \leq \tau$ **do**
4 **for all occurrences** (x', y', j) **do**
5 **if** $(x' - x)^2 + (y' - y)^2 < \epsilon^2$ **then**
6 add edge $((x, y, k), (x', y', j))$ to the occurrence graph
7 Build the connected components of the occurrence graph
`// each connected component is a spatiotemporal pattern`
8 Output the frequent connected components.

Figure 7: Generation of spatiotemporal patterns.

4.5 Generation of Spatiotemporal Patterns

When the algorithms output a frequent pattern P (line 8, in function *mine*), they also output a list of the occurrences of P. This list consists of triplets (x, y, k) where (x, y) are the coordinates of an occurrence (see Definition 7), and k is the index of $G_k \in \mathcal{D}$ where this occurrence appear. From this list, the algorithm of Figure 7 computes the spatiotemporal patterns based on P as follows: first, it builds the occurrence graph of pattern P w.r.t to ϵ and τ, as defined in Definition 9 (lines 1-6). Given an occurrence (x, y, k), the algorithm computes its distance between every other occurrence in the τ previous graphs G_j. The number of these occurrences is at most $O(\tau. \max_i(m_i))$, where $\max_i(m_i)$ is the maximal size of the graphs in $\mathcal{D}$. Therefore, the complexity of building the occurrence graph of a pattern is $O(\tau. \max_i(m_i). \sum_i m_i)$ (since the number of occurrences of a pattern is at most $2 \sum_i m_i$). The computation of the connected components and their frequency (line 7) is done by a traversal of the occurrence graph (linear complexity). Finally, the complexity of computing all frequent spatiotemporal patterns based on a pattern P is $O(\tau. \max_i(m_i). \sum_i m_i)$.

5 Experiments

We now present the computational results obtained by our proposed algorithms Plagram and DyPlagram. Since, to the best of our knowledge, Plagram is the first frequent plane graph mining algorithm, we could not compare it with any other algorithm with the same purpose. Nevertheless, to check how efficient our dedicated algorithm is in comparison with a general-purpose one, we report here a comparison between Plagram and gSpan. We also compare DyPlagram with Plagram in terms of efficiency. In summary, the conducted experiments aimed to answer four main questions:

1. How do Plagram and gSpan scale on video data?

2. How efficient is PLAGRAM in finding the patterns we are interested in, in comparison with GSPAN?

3. How efficient is DYPLAGRAM in comparison with PLAGRAM?

4. Are the spatiotemporal patterns meaningful on video data?

For GSPAN, we asked the authors of [3] for their C++ code. For DYPLAGRAM and PLAGRAM, we adapted the source code of GSPAN to implement their features and to allow a fair comparison. The experiments were carried out on a 3.08 GHz CPU with 8 GB of RAM memory under Debian GNU/Linux (2.6.26-2-amd64 x86_64) operating system.

5.1 Efficiency

Here, we evaluate how efficient PLAGRAM is in comparison with the general-purpose algorithm GSPAN.

5.1.1 Video datasets

The datasets we used for these experiments were created from a set of frames of a synthetic video. The choice of making a synthetic video was beneficial to our experiments, since we did not have to deal with common video artifacts that occasionally disturb the segmentation process. The video has 721 frames in total. Three identical objects (airplanes) are moving in the video such that they may overlap or even get partially out of the video frames (this helped us to evaluate how well spatiotemporal patterns can be used to represent the trajectory of the airplanes individually, as reported in Section 5.3).

After generating the video, we represented each frame as a plane graph. For this task, we used 2 different methods, which led to 2 different datasets of such graphs, as described below:

Triangulation Assuming that the video frames were already segmented by their different pixel colors, for each frame, the barycenters of the segmented regions became nodes and a Delaunay triangulation of this set of nodes was constructed. The final graphs had, on average, 197.33 nodes with an average degree of 2.93. The labels of the nodes were generated based on the size of the regions (in number of pixels). The size of the regions were discretized into 10 equal bins, which led to 10 possible node labels. The final set of graphs formed the *Triangulated* dataset. Note that, in this dataset, each graph is a 2-connected graph.

RAGs (Region Adjacency Graphs) We also represented each frame as a RAG (Region Adjacency Graph). More precisely, the nodes are computed in the same way as for the *Triangulated* dataset, except that there is also one node representing the outer region. An edge exists between two regions (or nodes) if these regions are adjacent in the frame. On average, each frame led to a graph

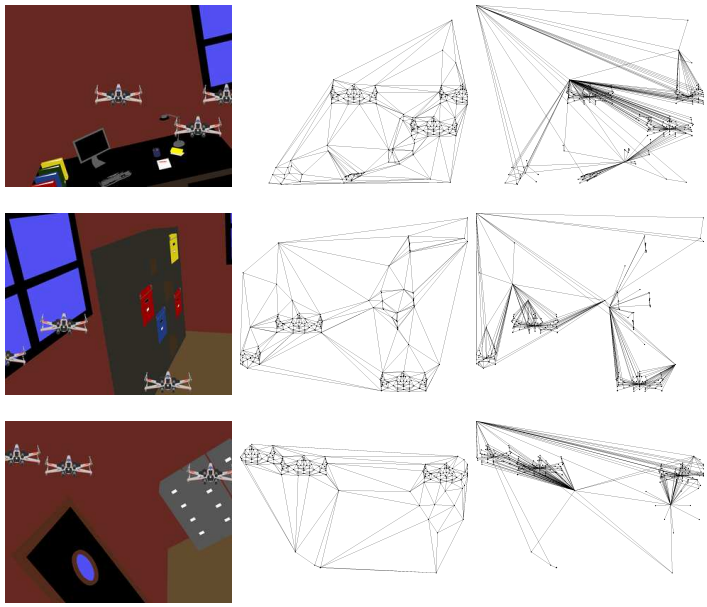

Figure 8: Example video frames (left) along with their corresponding triangulated (middle) and RAG (right) representations. In the latter, the upper-left node represents the outer region.

with 245.2 nodes, with an average degree of 2.23, and the labels of the nodes were discretized in the same way as for the *Triangulated* dataset. Here, the final set of graphs formed the *RAG* dataset.

Contrary to the graphs in the *Triangulated* dataset, the edges of the target graphs are more meaningful, since they represent adjacencies between regions. Moreover, if different regions have the same barycenters, they are not discarded as for the *Triangulated* dataset. This explains the higher number of nodes in this new dataset.

One disadvantage of the *RAG* dataset, however, is that the generated graphs may not be 2-connected. Since PLAGRAM mines only 2-connected patterns, it is not able to find a pattern that spans on several 2-connected components. Indeed, in the experiments, we found bigger patterns in the *Triangulated* dataset. Nevertheless, interesting patterns were also found by PLAGRAM in the *RAG* dataset.

Some example frames (left) along with their triangulated (middle) and RAG (right) representations are illustrated in Figure 8.

5.1.2 Results

Several factors may influence the efficiency of PLAGRAM in comparison with gSPAN. As PLAGRAM is dedicated to plane graphs, two patterns that are different for PLAGRAM (due to the order of their edges) may be only one pattern for gSPAN. In this way, our algorithm would find more patterns than gSPAN. However, since our extension building step is restricted to faces instead of single graph edges as in gSPAN, we would expect to generate fewer extensions as well as patterns. In any case, the complexity of our isomorphism test is lower. Therefore, in order to understand the most important of these factors, we considered the following in our experiments:

- The total execution time.
- The number of output patterns.
- The number of generated extensions.

We also considered the following ratios in order to make a fair comparison between the pattern matching and the extension building steps of PLAGRAM and those of gSPAN:

- The ratio of the total pattern matching step time to the total size of the output patterns (in number of edges).
- The ratio of the total extension building step time to the total size of the generated extensions (in number of edges).

Figure 9 and 10 present the results obtained on the *Triangulated* and *RAG* datasets, respectively. In each graph, the x-axis represents absolute minimum supports, which were lowered while the computation time of PLAGRAM was below 2 hours. Each point on each graph is the average result of 8 executions of the algorithms.

gSPAN could not finish its executions, even for the highest tested minimum support (721) on both datasets (in fact, it was interrupted after 3 days of computation). However, to better understand its behavior, we stopped it after 2 hours of execution and plotted here the intermediate results obtained with the highest minimum support of 721 (which is actually the same for the other minimum supports). This 2-hour execution of gSPAN is referred to here as gSPAN2.

Triangulated dataset Graph (a) presents the total execution time of PLAGRAM. Graphs (b) and (c) present, respectively, the number of extensions and the number of output patterns of PLAGRAM and gSPAN2.

Contrary to gSPAN, PLAGRAM finished its executions for every tested support. As presented in graphs (b) and (c), the total execution times increased along with the number of extensions and patterns, respectively. Considering gSPAN2, observe that its number of extensions was higher than that of PLAGRAM for almost all tested minimum supports (remember that PLAGRAM only

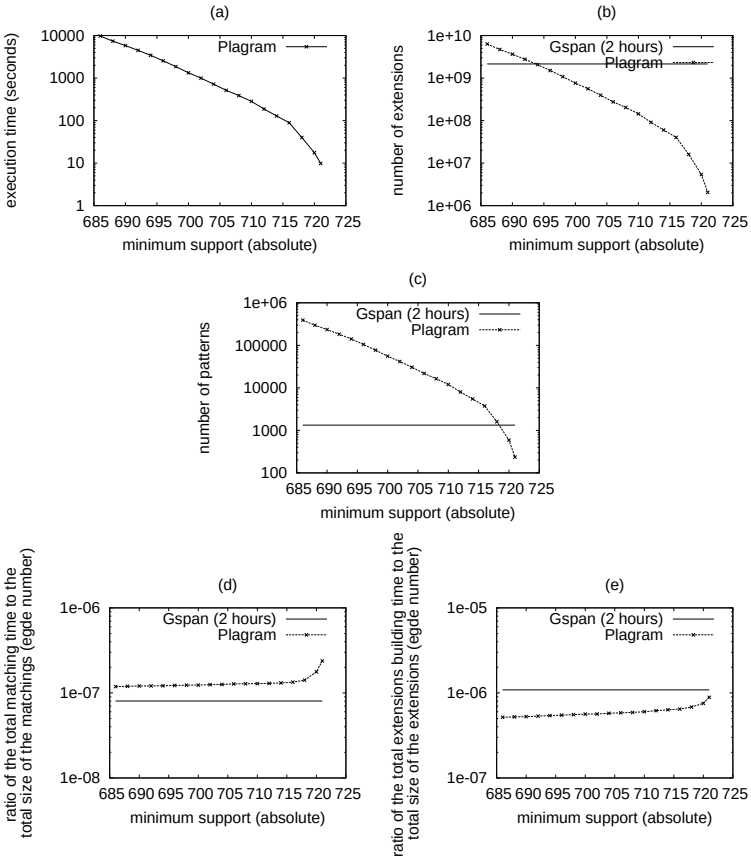

Figure 9: Efficiency of PLAGRAM and the 2-hour execution of GSPAN on the *Triangulated* dataset.

considers 2-connected plane graphs). In addition, for the minimum support of, e.g., 688, the patterns output by PLAGRAM had on average 30 edges, while, in the same period of time (two hours), GSPAN2 output fewer patterns with at most 10 edges.

What is worth observing as well are the results given by graph (d). It presents the ratio of the total time for all matchings to the total size (number of edges) of the matched graphs. Although this ratio was a little higher for PLAGRAM than for GSPAN2, it is worth noting that the patterns generated by GSPAN2 were smaller (at most 10) than those generated by PLAGRAM. If the complexity of the subgraph isomorphism test of PLAGRAM was the same as that of GSPAN, the matching ratio of PLAGRAM would be a lot higher.

Finally, graph (e) presents the ratio of the total extension step time to the total size of the generated extensions, in number of edges. Note that PLAGRAM had slightly better results in comparison with GSPAN2.

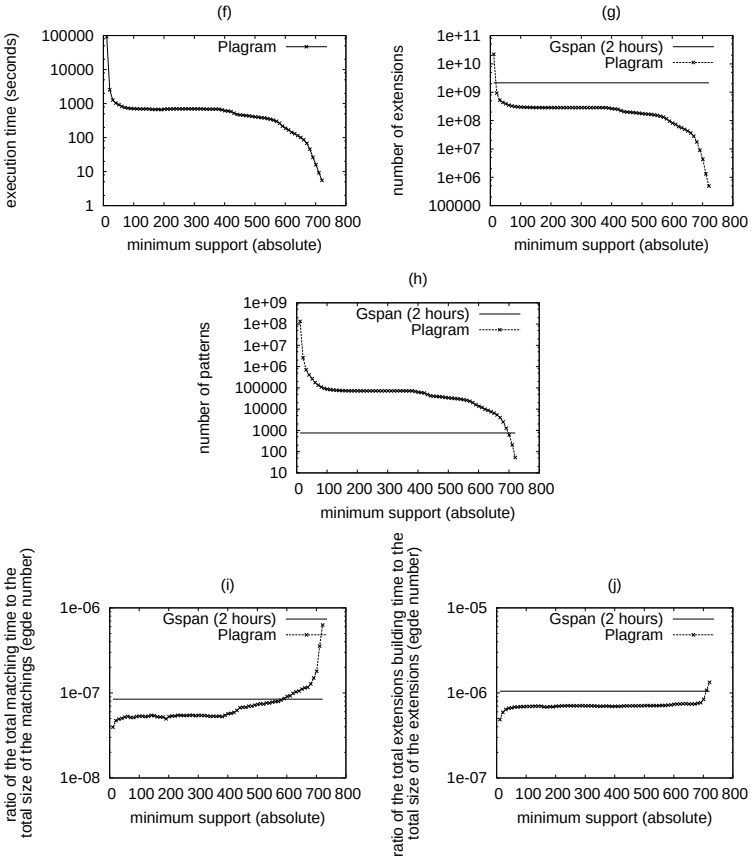

Figure 10: Efficiency of PLAGRAM and the 2-hour execution of GSPAN on the *RAG* dataset.

RAG dataset As shown in Figure 10, on this dataset the behaviors of PLAGRAM and GSPAN2 were quite similar to those on the *Triangulated* one. Here, however, PLAGRAM had a slightly better matching ratio than GSPAN2 for lower minimum supports. Since PLAGRAM mines only 2-connected patterns, the average size of the patterns found in the *Triangulated* dataset was higher than that in the *RAG* dataset, for the same minimum supports. For example, if we consider the support of 721 frames, the patterns found in the *Triangulated* dataset had on average 8 edges. Here, the patterns had 4 edges, on average.

Step Times We also measured the relative times of the main steps of the algorithms PLAGRAM and GSPAN2. On our datasets, the extension building step of GSPAN2 was on average 90% of the total execution time, whereas the matching step was always less than 5%, and the canonical-test step was negligible. For PLAGRAM, the most expensive step was also the extension building step, which

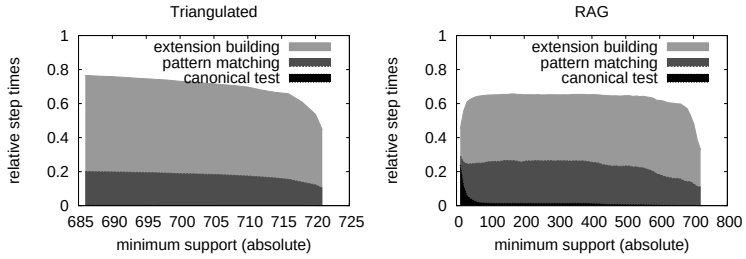

Figure 11: Relative step times of PLAGRAM. Black region: canonical-test time, dark gray region: matching time, and light gray region: extension building time (the unfilled space at the top-right of the graphs corresponds to other steps of the algorithm, e.g., I/O operations).

varied from 40% to 60% of the total execution time. The pattern matching step, in its turn, was around 20%, while the canonical-test step was almost always less than 5%. Figure 11 presents the computed relative times of PLAGRAM (y-axis) on the *Triangulated* (left) and *RAG* (right) datasets, for all tested minimum supports (x-axis). In conclusion, we believe that the main reason why PLAGRAM is more efficient than GSPAN is the lower number of extensions produced by PLAGRAM rather than only the faster pattern matching as one could expect.

5.2 DyPlagram vs. Plagram

We now present a comparison between DYPLAGRAM and PLAGRAM on the video described in Subsection 5.1.1. The idea here is to check how efficient is to consider freq_{seq} (with a time threshold τ of 1) instead of just freq.

Figure 12 shows the total execution time and the number of patterns generated by DYPLAGRAM and PLAGRAM on the datasets *Triangulated* (graphs (k) and (l)) and *RAG* (graphs (m) and (n)) for different minimum supports. Observe that DYPLAGRAM generated fewer patterns than PLAGRAM on both datasets, which makes its total execution time shorter than that of PLAGRAM. This is particularly clear on the *Triangulated* dataset, where it was possible to mine patterns with DYPLAGRAM with much lower minimum supports in lower execution times.

5.3 Meaningfulness of the Spatiotemporal Patterns

To evaluate the meaningfulness of a spatiotemporal pattern, we start by introducing two measures which assess how precise and complete a spatiotemporal pattern SP corresponds to the trajectory of a moving object o in the video frames. These measures are adaptations of the popular measures *precision* and *recall* as described below:

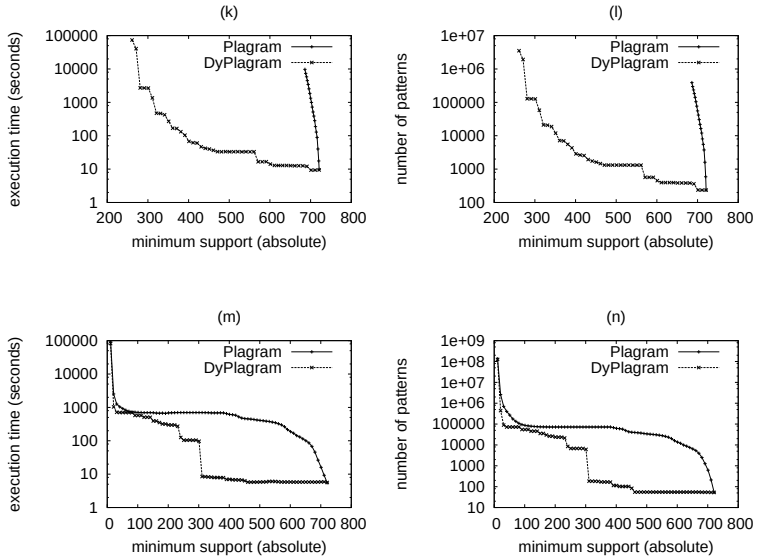

Figure 12: Efficiency of DyPlagram and Plagram on the *Triangulated* ((k) and (l)) and RAG ((m) and (n)) datasets.

- **precision**: fraction of the occurrences in SP where every node maps to o in the corresponding video frames. The intuition behind this measure is to evaluate the *purity* of SP, that is, SP has the maximum precision if all nodes in its occurrences map only to o and nothing else.

- **recall**: Let n be the number of frames in which o is present. The recall is defined as the fraction of n in which there exists at least one occurrence in SP where every node maps to o. Here, the intuition is to evaluate the *completeness* of SP. More precisely, the idea is to check whether the occurrences in SP map to all occurrences of o in the set of video frames.

Since our algorithms are exhaustive, that is, they mine for all frequent spatiotemporal patterns in the graph database without supervision, the mining result may consist of different spatiotemporal patterns corresponding to the trajectories of different objects, or even to no specific one (w.r.t. to the proposed measures). Therefore, for this evaluation, we focused on the 3 moving airplanes in our example video (see Figure 8). The idea is to check whether the defined spatiotemporal patterns can well represent the trajectory followed by the 3 airplanes individually.

In this context, the experiments were conducted as follows:

- First, we generated patterns with Plagram on the RAG dataset (on which the algorithms showed better efficiency results). Since the idea is

freq_{st} = 10 and ϵ = 10 (Total: 308)				
airplane	precision (%)	recall (%)	total	coverage recall (%)
1	99	12	75	74
2	93	3	94	28
3	99	1	92	39
freq_{st} = 50 and ϵ = 20 (Total: 158)				
airplane	precision (%)	recall (%)	total	coverage recall (%)
1	83	17	64	94
2	100	29	39	98
3	97	10	49	73
freq_{st} = 50 and ϵ = 170 (Total: 44)				
airplane	precision (%)	recall (%)	total	coverage recall (%)
1	56	56	18	93
2	87	17	7	88
3	90	14	13	42

Table 3: Characteristics of the spatiotemporal patterns obtained with different freq_{st} and ϵ constraints.

to obtain spatiotemporal patterns with high recall and given that our airplanes appear in every video frame, we used a minimum freq_{seq} of 721. In other words, we were interested in mining patterns that are present in every video frame, that is, with τ of 1.

- Then, we post-processed the mining results by selecting the *target patterns*, that is, the frequent patterns that had at least one occurrence mapping entirely (i.e., all nodes) to at least one airplane in the first video frame.

- Next, we generated spatiotemporal patterns based on these target patterns (with τ=1). This was done by using the post-processing method described in Definition 4.5. We experimented minimum freq_{st} of 10 and 50, and 3 different spatial threshold ϵ of 10, 20, and 170 pixels (from 20 to 160 pixels, the results were quite similar and thus are not reported here).

- Finally, we kept only the spatiotemporal patterns whose first occurrence entirely mapped to one of the airplanes. In other words, if this first occurrence of the pattern entirely mapped to a given airplane o, the pattern was assumed to represent the trajectory of o. Consequently, the pattern's precision and recall were computed w.r.t o.

Table 3 summarizes the results. For each experimented pair (freq_{st}, ϵ), it gives the total number of obtained spatiotemporal patterns between brackets (including those that were discarded). For each airplane in the video, the first two columns give the average precision and recall computed for its associated spatiotemporal patterns. The remaining columns show, respectively, the total number and the fraction of the video frames covered by such patterns, referred to here as coverage recall.

As can be seen in Table 3, the ϵ constraint has an important impact on the obtained results. Indeed, if it is set too low (to 10 pixels, in our example), we

obtain spatiotemporal patterns with high average precision for each airplane, as different occurrences of patterns which map to different airplanes are very well distinguished. However, this leads to a low average recall: since only very close occurrences of the same pattern are linked, the spatiotemporal patterns tend to be short (i.e., have low freq_{st}) and, consequently, resulting in a low coverage recall. Indeed, when using $\epsilon = 10$, no spatiotemporal patterns with freq_{st} higher than or equal to 50 were found for airplaine 2, which explains why we use a minimum freq_{st} of 10 in this case. Conversely, for a higher ϵ of 170 pixels, the average precision drops as the different airplanes are not well distinguished anymore. For example, it was possible to obtain spatiotemporal patterns with higher recall for airplane 1 (when comparing to the other experiments). However, they had low average precision. Finally, for $\epsilon = 20$ pixels, the computed spatiotemporal patterns led to better results w.r.t to coverage recall, while keeping a high average precision.

With this series of experiments, each target pattern derived not only a single spatiotemporal pattern for each airplane's trajectory (with high recall), but a set of them (with low recall). This is explained by the fact that the airplanes in our video are identical and may overlap during their trajectories. Besides, airplanes 1 and 3 go partially out of some of the video frames. Consider, e.g., the results obtained by $\text{freq}_{st} = 50$ and $\epsilon = 20$ pixels. Since airplane 1 gets partially out of the video frames around 6 times, a higher number of spatiotemporal patterns were derived for this airplane, which represent the different time intervals where this airplane is visible through the video. As another example, consider airplane 2. It gets hidden only twice by airplane 3 (during around 10 frames) and never goes out of the video frames. This explains the lower number of patterns found for this object, for the same freq_{st} and ϵ constraints. Note that, in the case of our example video, increasing the time constraint τ is not a solution for this effect. As the airplanes are identical and given that one airplane (number 2) never goes out of the video frames, there is never a gap higher than 1 between two occurrences of the target patterns.

In conclusion, our experiments showed that our techniques give promising results when applied to object tracking in videos, especially when moving objects in a scene have some degree of similarity. However, it is worth mentioning that successful results may depend on the adjustment of constraints, which may in turn vary according to the characteristics of the video.

6 Conclusions

We presented a frequent plane graph mining algorithm called PLAGRAM and its extension DYPLAGRAM to mine spatiotemporal patterns. Our conducted experiments showed that PLAGRAM (and, consequently, DYPLAGRAM) was able to efficiently run on graph-based video datasets, on which a general-purpose graph mining algorithm failed to finish its computations. The experiments also showed that besides improving efficiency, the 2-connectedness restriction on the patterns imposed in our algorithms does not limit the meaningfulness of the

final patterns. On the contrary, we believe that these algorithms may be a useful tool to track objects in videos using spatiotemporal patterns.

Having video applications in mind, we have identified three directions for further work: first, we would like to add more complex labels to the nodes and edges of the graphs to describe, e.g., different characteristics of the frame regions. This would allow us to use some kind of "approximate matching" instead of the typical subgraph isomorphism test used in our algorithms. That is, two nodes (or edges) will be considered equivalent if their attributes are similar with respect to a defined similarity function. Second, to track an object in a video, we proposed a strategy in which one should post-process the entire set of frequent patterns to select those that map to the object of interest. An interesting way to enhance the efficiency of this strategy would be to add these patterns as a constraint to the mining process. Finally, following the same line of reasoning as for the popular frequent itemset mining problem [1], a natural direction for further work is to adapt Plagram and DyPlagram to mine directly maximal or closed patterns.

Acknowledgments The authors would like to thank Siegfried Nijssen for providing the source code of gSpan and for his helpful remarks at the beginning of this research. The program at http://www.cs.cmu.edu/~quake/triangle.html was used to build the Triangulated dataset. This work has been partially supported by the project BINGO2 (ANR-07-MDCO 014-02).

References

[1] Rakesh Agrawal and Ramakrishnan Srikant, *Fast algorithms for mining association rules*, Proceedings of the International Conference in Very Large Data Bases (VLDB), 1994, pp. 487–499.

[2] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis, *Mining graph evolution rules*, Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), 2009, pp. 115–130.

[3] Björn Bringmann and Siegfried Nijssen, *What is frequent in a single graph?*, Proceedings of the Pacific-Asia conference on Advances in knowledge discovery and data mining (PAKDD), 2008, pp. 858–863.

[4] Ruey-Feng Chang, Chii-Jen Chen, and Chen-Hao Liao, *Region-based image retrieval using edgeflow segmentation and region adjacency graph*, Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), 2004, pp. 1883–1886.

[5] Guillaume Damiand, Colin De La Higuera, J.-C Janodet, Emilie Samuel, and Christine Solnon, *Polynomial algorithm for submap isomorphism: Application to searching patterns in images*, Proceedings of the 7th Workshop

on Graph-Based Representation in Pattern Recognition (GBR), vol. 5534, 2009, pp. 102–112.

[6] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon, *Frequent submap discovery*, Annual Symposium on Combinatorial Pattern Matching (CPM), 2011.

[7] Lawrence B. Holder, Diane J. Cook, and Surnjani Djoko, *Substructure discovery in the* SUBDUE *system*, Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, 1994, pp. 169–180.

[8] Tomás Horváth, Jan Ramon, and Stefan Wrobel, *Frequent subgraph mining in outerplanar graphs*, Data Mining and Knowledge Discovery **21** (2009), no. 3, 472–508.

[9] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda, *An apriori-based algorithm for mining frequent substructures from graph data*, Proceedings of the European Conference on Principles and Practice of Data Mining and Knowledge Discovery in Databases (ECML/PKDD), 2000, pp. 13–23.

[10] Xiaoyi Jiang and Horst Bunke, *Optimal quadratic-time isomorphism of ordered graphs*, Pattern Recognition **32** (1999), no. 7, 1273–1283.

[11] Hisashi Koga, Tsuji Tomokazu, Takanori Yokoyama, and Toshinori Watanabe, *New application of graph mining to video analysis*, Intelligent Data Engineering and Automated Learning, 2010, pp. 86–93.

[12] Mehmet Koyutürk, Ananth Grama, and Wojciech Szpankowski, *An efficient algorithm for detecting frequent subgraphs in biological networks*, Bioinformatics **20** (2004), no. 1, 200–207.

[13] Michihiro Kuramochi and George Karypis, *Frequent subgraph discovery*, Proceedings of the IEEE International Conference on Data Mining (ICDM), 2001, pp. 313–320.

[14] K-N. T. Nguyen, Loïc Cerf, Marc Plantevit, and Jean-Francois Boulicaut, *Discovering inter-dimensional rules in dynamic graphs*, Workshop on Dynamic Networks and Knowledge Discovery (DyNaK), 2010.

[15] Siegfried Nijssen and Joost N. Kok, *A quickstart in frequent structure mining can make a difference*, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2004, pp. 647–652.

[16] Adriana Prado, Baptiste Jeudy, Elisa Fromont, and Fabien Diot, *Plagram : un algorithme de fouille de graphes plans efficace*, Confrence d'Apprentissage (CAp'2011), 2011, pp. 343–359.

[17] Severin Stalder, Helmut Grabner, and Luc Van Gool, *Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition*, On-line learning for Computer Vision Workshop, 2009, pp. 1409–1416.

[18] Xifeng Yan and Jiawei Han, *gspan: Graph-based substructure pattern mining*, Proceedings of the IEEE International Conference on Data Mining (ICDM), 2002, pp. 721–724.

[19] Alper Yilmaz, Omar Javed, and Mubarak Shah, *Object tracking: A survey*, ACM Comput. Surv. **38** (2006), no. 4, 13+.