Graph mining for object tracking in videos

Fabien Diot

École Doctorale "Science, Ingénierie, Santé" (ED 488)

THÈSE

pour l'obtention du grade de

Docteur en Informatique

de l'UNIVERSITÉ JEAN-MONNET

défendue le 3 Juin 2014 par

Fabien DIOT

FOUILLE DE GRAPHES POUR LE SUIVI D'OBJETS DANS LES VIDÉOS

Commission d'examen:

Examinateurs:	Christine SOLNON	-	Professeur, INSA Lyon
	Christophe DUCOTTET	-	Professeur, Université Jean Monnet de Saint-Étienne
Rapporteurs:	Toon CALDERS	-	Professeur, Université Libre de Bruxelles
	Christian WOLF	-	Maître de Conférences - HDR, INSA Lyon
Directeur:	François JACQUENET	-	Professeur, Université Jean Monnet de Saint-Étienne
Co-directeurs:	Elisa FROMONT	-	Maître de Conférences, Université Jean Monnet de Saint-Étienne
	Baptiste JEUDY	-	Maître de Conférences, Université Jean Monnet de Saint-Étienne
	Olivier MARTINOT	-	Directeur de Département, Alcatel-Lucent BellLabs
Invité:	Emmanuel MARILLY	-	Ingénieur de Recherche, Alcatel-Lucent BellLabs

---

Doctoral School "Science, Ingénierie, Santé" (ED 488)

PhD THESIS

to obtain the title of

Doctor in Computer Science

of the UNIVERSITY JEAN-MONNET

defended the 3rd of June 2014 by

Fabien DIOT

GRAPH MINING FOR OBJECT TRACKING IN VIDEOS

Jury:

Examiners:	Christophe DUCOTTET	-	Professor, Université Jean Monnet, Saint-Étienne
	Christine SOLNON	-	Professor, INSA Lyon
Reviewers:	Toon CALDERS	-	Professor, Université Libre de Bruxelles
	Christian WOLF	-	Assistant Professor - HDR, INSA Lyon
Supervisor:	François JACQUENET	-	Professor, Université Jean Monnet, Saint-Étienne
Co-supervisors:	Elisa FROMONT	-	Assistant Professor, Université Jean Monnet, Saint-Étienne
	Baptiste JEUDY	-	Assistant Professor, Université Jean Monnet, Saint-Étienne
	Olivier MARTINOT	-	Department Director, Alcatel-Lucent BellLabs
Invited:	Emmanuel MARILLY	-	Research Engineer, Alcatel-Lucent BellLabs

---

REMERCIEMENTS

Je voudrais commencer par remercier les Professeurs Toon Calders, de l'Université Libre de Bruxelle, et Christian Wolf, de l'Institut National des Sciences Appliquées de Lyon, d'avoir accepté de rapporter mon travail de thèse et permis d'améliorer ce manuscrit grâce à leurs remarques constructives. J'adresse également mes remerciements à Christine Solnon, Professeur à l'Institut National des Sciences Appliquées de Lyon, et Christophe Ducottet, Professeur l'Université Jean Monnet de Saint Etienne, pour leur implication en tant qu'examinateurs de ces travaux.

Je tiens ensuite à remercier François Jacquenet pour la confiance qu'il m'a accordé en acceptant d'associer son nom à mes travaux en tant que directeur de cette thèse, faisant fi de la piètre qualité du projet de compilation de 3ème année. Je remercie, tout particulièrement et chaleureusement, mes encadrants au laboratoire Hubert Curien. Baptiste Jeudy, pour son expertise, sa patience et sa capacité à toujours trouver une explication simple et intuitive à des problèmes compliqués. Elisa Fromont pour son éternelle bonne humeur, son implication et l'énergie dont elle a fait preuve, chaque jour et sur tous les fronts, dans l'accompagnement et l'organisation de cette thèse pour me recentrer, me motiver et me secouer à chaque fois que cela a été nécessaire. En dépit de conditions parfois difficiles, en raison de l'alternance entre Paris et Saint Etienne, vous avez tous les deux effectué un formidable travail d'encadrement sans lequel cette thèse ne se serait certainement pas aussi bien déroulée. Je remercie aussi mes encadrants aux Bell Labs, Emmanuel Marilly et Olivier Martinot qui on su me guider dans les méandres administratifs d'Alcatel-Lucent ainsi que pour toutes ces discussions qui m'ont permis de prendre du recul sur mes travaux.

Je salue les collègues du laboratoire Hubert Curien, Aurélien, Chahrazed, Christophe, David, Emilie M., Emilie S., Mathias, Mickaël, Jean-Philippe, Julien S., Laurent, Thomas et Tung, avec une mention spéciale pour Fabrice A. qui a toujours pris le temps de répondre à mes questions concernant linux. Je remercie aussi les collègues d'Alcatel-Lucent, Abdelkader, Alexandre, Arnaud, Corinne, Erwan, Fabrice P., Flo-

rentin, Gérard, Johann, Jerôme, Karim, Marwen, Myriam, Nicolas, Olivier D., Sylvain, Vincent H. et Vincent V. pour leur accueil chaleureux au sein des Bell Labs.

Je n'oublie pas les nombreux amis, toujours présents pour partager un canon (avec l'accent Stephanois). Je remercie ceux qui ont pu se libérer pour assister à ma soutenance, Antoine, Cécile, Charlotte, Gregory (dit Schnoble) avec qui j'ai partagé de nombreuses discussions constructives (ou pas...) pendant nos pauses déjeuné, Julien B. (dit Pepess) qui a répondu présent à chaque fois que j'ai eu besoin de lui, de son drone ou de sa voiture pour réaliser des vidéos et enfin Pierre-Emmanuel (dit Jhunior) pour son petit mot de félicitations qui m'a beaucoup touché. Remerciements sincères aux amis Parisiens qui m'ont hébergé pendant les débuts difficiles de cette thèse, Florent (dit Noeil), Eric et Julie qui nous on prêté leur appartement durant l'été 2011, Odile et Thomas (dit Roux), ainsi que Gilles et Anne-Flore et leurs amis qui m'ont accueilli comme l'un des leurs.

Enfin, je remercie ma famille, en me remémorant tous ces bons moments passés ensemble les Samedi et Dimanches après midi chez mes grand-parents. Mes deux frères, avec qui j'ai partagé une enfance heureuse. Mes parents qui ont admirablement réussi dans cette difficile épreuve qu'est celle d'élever trois garçons turbulents (en particulier en groupe ...). Pour terminer je remercie Delphine, à qui cette thèse et moi devons beaucoup. Tu m'a supporté quand moi même je n'y arrivais plus et ton soutien ainsi que tout amour mon énormément aidé à franchir les épreuves. Merci.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER 1

Introduction

L'omniprésence des moyens de capture vidéo, tels que les appareils photo ou encore les téléphones portables et la facilité de partage de ces contenus produits à travers des plate-formes comme Youtube, Dailymotion ou encore les réseaux sociaux, a conduit à une explosion du volume de documents multimédias disponibles sur internet. Par exemple en 2014, il a été estimé que 100 heures de vidéos sont "uploadées" chaque minute sur Youtube [1]. Cette masse de données multimédia soulève de nombreux problèmes, en particulier celui de leur indexation permettant aux moteurs de recherche (e.g. Google) de les retrouver facilement. Pour pouvoir répondre de manière efficace et pertinente aux requêtes des utilisateurs dans les moteurs de recherche, il est nécessaire d'associer à chaque contenu multimédia un ensemble de mots-clés. Ces derniers peuvent prendre une forme textuelle (ce qui est le plus souvent utilisé en indexation) mais peuvent aussi être constitués de résumés en images ou de courtes séquences vidéo permettant à un utilisateur de décider rapidement de l'intérêt d'un contenu retourné par un moteur de recherche. Dans la plupart des cas, les plate-formes de partage encouragent le producteur de contenu à fournir ces mots-clés. Cependant, ces informations peuvent être erronées ou simplement incomplètes. Par exemple, il n'est pas rare de voir des noms de jeux vidéos connus ou des personnes célèbres être associés à des vidéos simplement pour en augmenter le nombre de vues. De même il est courant qu'une vidéo soit mise en ligne sans aucun mot-clé associé. De nombreux travaux sont donc consacrés à l'extraction automatique (sans intervention de l'utilisateur) de l'information contenue dans les documents multimédias pour en permettre un usage plus pertinent par la suite.

De nombreuses pistes ont été explorées pour permettre d'associer automatiquement à une vidéo un ensemble de mots-clés (des catégories) ou un résumé visuel de celle-ci (qui pourra ensuite être classé pour permettre une indexation plus perti-

[1] source : www.youtube.com/yt/press/

nente). Dans cette thèse, nous nous intéressons particulièrement aux méthodes qui ne nécessitent pas d'apprendre des modèles à partir de données étiquetées, appelées *méthodes non supervisées*, qui sont plus réalistes pour traiter des vidéos au contenu très variable. Contrairement aux données textuelles, le contenu visuel n'est pas régi par un vocabulaire et une grammaire permettant de l'analyser efficacement et d'en déduire le sens ainsi que le sujet. Là où les caractères d'un alphabet sont regroupés en mots, phrases et paragraphes dans les documents textuels, la structuration de l'information contenue dans les images reste à déterminer. Par exemple, le problème de segmentation qui consiste à regrouper les pixels en ensembles cohérents représentant les différents objets d'une image sans avoir une idée du type d'objet recherché a priori, a fait l'objet de recherches durant des décennies et est toujours d'actualité. Pourtant, de la même manière que deux textes peuvent être comparés en mesurant la similitude de la distribution des mots qui les composent, deux images peuvent également être comparées en utilisant, par exemple, des histogrammes reflétant la distribution de la couleur des pixels de chacune d'elles. Cette approche permet de mesurer la similarité de deux images en se basant sur la distribution des couleurs de leurs pixels mais ignore complètement leur sens. Par exemple, deux images représentant la même scène, l'une de jour et l'autre de nuit, seront considérées très différentes par cette méthode, alors qu'à l'inverse, deux images représentant des objets différents mais de même couleur pourront être considérées similaires. Dans le cas de vidéos, les histogrammes de couleur sont utilisés pour détecter des frames clefs, c'est à dire, des frames considérées importantes car ayant une distribution de couleur très différente de celles des frames précédentes, indiquant que la scène a, par exemple, évolué. Ces frames clefs peuvent être utilisées pour fournir un résumé visuel de la vidéo, mais les histogrammes ne permettent pas de fournir d'informations sémantiques précises sur le contenu des séquences vidéo extraites. En particulier, ils ne permettent pas de se focaliser sur les objets d'intérêts. L'information utilisée dans ces histogrammes est de trop bas niveau pour décrire le contenu visuel. La structure d'une image, c'est à dire, la façon dont sont agencés les différents éléments visuels qui la composent, est donc une information capitale pour analyser celle-ci. Les mots visuels locaux ou les représentations pyramidales de ces mots tentent d'apporter une solution à ce problème en représentant la structure d'une image à l'échelle de petits groupes de pixels. Cette structuration reste toutefois limitée car ces mots visuels, au contraire des mots dans un texte, ne sont pas disposés les uns par rapport aux autres selon les règles d'une grammaire formelle bien définie ce qui limite leur capacité à décrire la relation entre les différents éléments visuels ainsi que la structure globale de l'image. Notre but dans cette thèse est d'extraire de l'information sémantiquement pertinente des vidéos de

manière non supervisée en prenant en compte les informations topologiques pouvant exister entre les éléments d'une image. Cette information pertinente prendra la forme de *tracks* représentant les objets d'intérêts suivis dans une vidéo.

Dans le cas des vidéos, il est possible d'utiliser l'information temporelle en plus de l'information spatiale pour identifier les objets. Lorsqu'un modèle de l'arrière plan de la scène est connu, comme c'est le cas dans beaucoup d'applications de vidéo surveillance, il est possible de détecter les objets se déplaçant dans le champ de vision de la caméra en soustrayant chaque image au modèle de l'arrière plan. Toutefois, même s'il est possible de maintenir à jour un modèle de l'arrière plan au fur et à mesure que de nouvelles images arrivent, cette méthode requiert que le mouvement de la caméra soit faible. Une autre stratégie couramment utilisée même lorsque l'arrière plan est instable, consiste à entraîner un classifieur sur un ensemble d'images représentant la classe d'objets souhaitée. Cette méthode requiert de connaître à l'avance le type d'objets intéressants et de bénéficier de suffisamment d'exemples d'apprentissage pour apprendre un modèle robuste. Elle est donc assez difficile à généraliser à tous les objets pouvant apparaître dans une vidéo dans une base comme Youtube.

Les approches actuelles ne sont applicables que dans des contextes particuliers et ne peuvent donc pas être utilisées pour traiter automatiquement un grand nombre de vidéos au contenu très différent. Le fait que la caméra soit immobile peut être facilement détecté (e.g., en utilisant des techniques de flux optique), auquel cas les méthodes de soustraction de fond peuvent permettre de détecter efficacement les objets. Dans le cadre de cette thèse nous nous intéressons à ce qui constitue un objet intéressant spécifiquement dans le cas de vidéos prises par une caméra en mouvement. Pour détecter les tracks représentant les objets d'intérêts, nous partons de l'hypothèse que les objets principaux apparaissent plus fréquemment que l'arrière plan puisque celui-ci change constamment. Par conséquent, nous considérons qu'un objet est un ensemble d'éléments visuels qui apparaissent fréquemment dans les images d'une vidéo. De plus, la relation de topologie entre les éléments visuels d'un même objet est supposée cohérente dans le temps ce qui nous pousse à rechercher des structures apparaissant fréquemment. Pour modéliser la structure globale d'une image, nous nous intéressons aux représentations à base de graphes attribués. Pour des raisons qui seront évoquées dans ce manuscrit, nous avons décidé de modéliser chaque frame d'une vidéo par un graphe d'adjacence de régions. Ce graphe s'appuie sur une segmentation au préalable de l'image en régions. Chaque région est représentée par un nœud et l'adjacence entre deux régions (et donc la topologie) est modélisée par un arc entre les deux nœuds correspondant aux régions. Le problème de recherche, non supervisée, d'ensembles d'éléments visuels fréquents à la topologie cohérente dans le temps (les

tracks) est analogue au problème de la fouille dynamique de sous graphes fréquents, c'est à dire à une recherche de sous graphes fréquents dans une base de données de graphes représentant une vidéo donnée. De plus, le processus de fouille de graphes étant non supervisé par essence, et le critère d'intérêt étant basé sur la fréquence, il n'est plus nécessaire qu'un utilisateur intervienne pour sélectionner les objets à suivre ou de connaître à l'avance le type d'objets présents dans la vidéo. Dans cette thèse nous nous focaliserons donc sur l'utilisation de ce type de méthodes pour extraire les tracks.

Ce document est organisé de la façon suivante. La première partie est consacrée à l'étude des deux domaines de recherche liés à cette thèse.

- Le chapitre 2 traite des techniques de traitement de l'image utilisées pour le suivi d'objets dans les vidéos afin de mettre en avant les limitations des approches actuelles.

- Le chapitre 3 se concentre sur le domaine de la fouille de graphes fréquents et en présente les principes ainsi que les principales approches développées dans la littérature.

La seconde partie de cette thèse est consacrée au développement de nos contributions.

- Le chapitre 4 est dédié aux contributions dans le domaine de la fouille de graphes fréquents. Notamment, nous présentons un algorithme de fouille de graphes plans efficace ainsi qu'une approche permettant d'exploiter l'information spatio-temporelle des vidéos pour limiter le nombre de graphes fréquents produits par notre algorithme et générer des motifs spatio-temporels. Plus précisément, ce chapitre décrit le fonctionnement de nos trois algorithmes, PLAGRAM (*Prado et al.* (2011)), DYPLAGRAM (*Prado et al.* (2013)) et DYPLAGRAM_ST (*Diot et al.* (2012)). Il présente les expérimentations conduites dans *Prado et al.* (2011) montrant la supériorité, en terme d'efficacité, de PLAGRAM en comparaison de l'algorithme générique GSPAN dans le cadre de la fouille de graphes plans. Ce chapitre expose aussi les expérimentations de *Prado et al.* (2013), comparant entre autre l'efficacité de PLAGRAM et DYPLAGRAM ainsi que celles de *Diot et al.* (2012), comparant l'efficacité de DYPLAGRAM et DYPLAGRAM_ST.

- Le chapitre 5 présente nos contributions dans le domaine du suivi d'objets. Il développe deux stratégies utilisant les sous graphes fréquents découverts par notre

algorithme de fouille. La première, présentée dans *Diot et al.* (2012), consiste à construire un graphe connectant les occurrences de sous graphes fréquents qui sont similaires. La deuxième méthode, appelée *TRAP*, utilise une technique de clustering hiérarchique pour regrouper les motifs spatio-temporels de DyPlagram_st ayant une trajectoire similaire. Ces clusters sont ensuite classés en fonction de leur taille et du nombre d'images de la vidéo qu'ils couvrent. Ce chapitre expose les expérimentations de *Prado et al.* (2011), *Prado et al.* (2013) et de *Diot et al.* (2012), visant à montrer que les motifs fréquents de Plagram, les motifs spatio-temporels de DyPlagram_st et ceux obtenus en post-traitement des motifs de DyPlagram ont un sens dans un contexte de suivi d'objets. Les expérimentations de *Diot et al.* (2012) permettent aussi de montrer que le chemin le plus court dans le graphe des occurrences permet de suivre l'objet principal d'une vidéo. Ce chapitre contient aussi une série d'expérimentations publiées dans *Diot et al.* (2014) montrant que les clusters les mieux classés, produits par *TRAP*, correspondent souvent aux objets principaux dans les vidéos testées. Enfin, dans la conclusion de ce chapitre, deux applications possibles de notre approche, qui ont donné lieu au dépôt de *deux brevets*, sont brièvement décrites. La première concerne le résumé automatique de vidéos et la deuxième présente une application permettant de construire un nuage d'étiquettes visuelles pour décrire une vidéo de la même manière que des nuages de mots sont utilisés pour décrire du texte.

Part I

Background

CHAPTER 2

A Review of Object Detection and Tracking in Videos

The visual tracking problem consists in determining the position of objects at each time step (frames) in a given video. Objects can be of very different types (e.g., rigid/deformable, variety of textures, different motion ...). The tracking context can also vary a lot (moving/non-moving camera, clutter, number of objects in the scene, frequency of occlusions...). Therefore, depending on the information known in advance about the tracked objects and about the tracking context, the problem of visual tracking can be addressed in very different ways. For example, in the context of video surveillance with non moving cameras, the fact that the background is fixed can be exploited to easily detect moving objects by subtracting consecutive frames and focus on what has changed. Visual tracking is a very active field of research in the computer vision domain which has grown quickly during the past 20 years. Indeed, the access to cheap cameras and the development of the web allow users to produce and share an increasing amount of multimedia content. Because of this, a lot of efforts have been put into developing processing tools to extract useful information from videos. In that regard, being capable of tracking visual features is important for a lot of applications. For instance, visual tracking is used in video surveillance applications to track pedestrians and detect suspicious activities and unlikely events. It is also used for video indexing purposes, by detecting and tracking the objects in a scene in order to describe it. Visual tracking is also effectively used for traffic monitoring and vehicle navigation or in the field of human-computer interactions through gesture recognition and eye gaze tracking.

To achieve visual tracking, a variety of visual features can be used. For example interest points are well suited to describe local features with high precision but fail at capturing more global features such as shape or topology. Regions and edges are better suited for the representation of shape and topology but are usually much more

influenced by image variations in images and are therefore harder to track robustly.

Because the literature about visual tracking is so vast, the purpose of this short review is not to study each approach that has been used in the past but rather to identify the current trends in the field. The interested reader can find a more broad review of the field up to 2006 in *Yilmaz et al.* (2006). A more recent technical report of York University also reviews the field of visual tracking in *Cannons* (2008). In section 2.2 we will introduce some basic tools that are commonly incorporated in video tracking frameworks. Section 2.3 will deal with the recent approaches in the visual tracking field and section 2.5 will conclude.

2.1 Feature Extraction

In order to track an object, one has to be able to find visual features representing it that are discriminant and robust enough. Those visual features can be regrouped in three major categories : points, extracted through interest point detection, regions, obtained with image segmentation techniques and edges, computed with edge detectors. This section will focus on giving some basic information about the techniques used in the literature to extract those 3 types of visual features.

2.1.1 Interest Point Detection

Interest points are small image patches, that exhibit texture properties. They are usually associated with a descriptor that tries to represent it in an invariant manner with respect to illumination changes and camera viewpoint. One of the first point detectors, called the Moravec point detector, was presented in *Moravec* (1979). The idea developed by the authors is to select image patches that have a low similarity, in term of intensities, with nearby overlapping larger image patches. The sum of square differences (SSD) is used to measure the similarity between two patches.

Another early point detector, named the Harris point detector was presented in *Harris and Stephens* (1988). Its principle is to use the image derivatives on x and y to evaluate if shifting a small window would reduce the SSD between the intensities of the pixels in the window. If shifting the window in any direction reduces the SSD then a local maxima has been found and the point is considered "interesting".

The most widely used point detector, which has been shown to be the most robust one (see *Mikolajczyk and Schmid* (2005)), is the more complex SIFT (Scale Invariant Feature Transform) detector of *Lowe* (2004). In its first step it computes several copies of the image by reducing its resolution. This produces a pyramid of scales. Neighboring images in this pyramid are subtracted to each other and extremums

are kept as interest points. Points with low contrast or that are along an edge are discarded. Then histograms of local gradient orientation are built. The major orientation is chosen as orientation of the point. At this point, each point has a location, a scale and an orientation. A descriptor is built for each one of them by placing 16 by 16 grid at the point's position, scale and orientation. The cells of this grid are regrouped in super-cells of 4 by 4 and an histogram of gradient of 8 bins, weighted with respect to the distance to the center of the grid, is computed in each one of those super-cells (see Figure 2.1). The value of the 8 bins of each of those histograms are stored in a vector of $8 \times 16 = 128$ dimensions and serves as descriptor for the interest points.

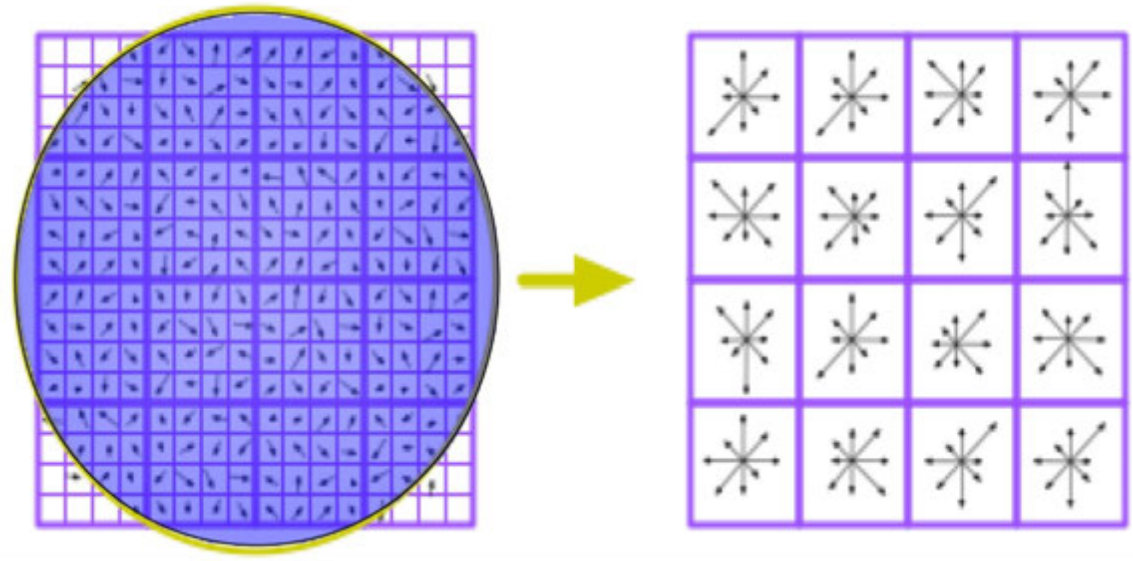

FIGURE 2.1: Histograms of gradient orientation computed for the SIFT descriptor of (*Lowe* (2004))

A comparative study of point detectors can be found in *Mikolajczyk and Schmid* (2005).

2.1.2 Image Segmentation

The aim of segmentation algorithms is to partition the image in visually similar regions.

Mean-Shift Clustering is a popular approach presented in *Comaniciu and Meer* (1999). This method clusters the multidimensional space $\mathcal{P}$ of pixel color and position. In this space each pixel is represented by a vector $[l, u, v, x, y] \in \mathcal{P}$ with $[l, u, v]$ for the color and $[x, y]$ for the position. Random cluster centers $c_i \in \mathcal{P}$ are selected in the image. A multidimensional ellipse is centered on each of this cluster centers in $\mathcal{P}$. Each

c_i is moved to the coordinares of the mean of the pixels inside the ellipse. The cluster centers are moved repeatedly until all of them stop moving. During this procedure cluster centers that are close can be merge. The Mean-Shift Clustering algorithm suffers from the fact that in some cases the mean is not a good representative of visually similar regions (e.g., regions for which the color changes gradually).

Another approach, developed in *Wu and Leahy* (1993), is the Graph-Cut segmentation. A graph $G = (V, E)$ is built with a node for each pixel. The weight on edges connecting the nodes typically represent the color, brightness or texture similarity of the nodes. A cut is a subset of the edges of the graph which once removed result in a partition of it into several disconnected subgraphs. The aim of Graph-Cut based segmentation is to find the partition that minimizes the sum of the weights of the edges in the cut. This minimum cut criterion tends to result in over-segmented images. To deal with this problem *Shi and Malik* (2000) proposed another segmentation algorithm based on graph partitioning. This approach proposed a global criterion, called the Normalized cut criterion, that captures the global properties of the regions instead of focusing on local properties like previous approaches. Here the cut depends both on the sum of the weights of its edges and the ratio between the weights of the nodes of each partition and all the nodes of the graph.

More recently *Felzenszwalb and Huttenlocher* (2004) proposed another graph based image segmentation algorithm[1]. In this approach each pixel is connected to its neighbors with edges weighted by the color distance between the pixels. Originally, each node is in its own region, and each region R_i has an internal difference $Int(R_i) = 0$. Once a region contains at least two nodes connected by an edge, its internal difference is defined as the maximum weight edge of its Minimum Spanning Tree (MST). To grow the regions, the edges are selected iteratively in increasing weight order and the regions they connect are merged. The process is stopped once the weight of the current edge is higher than $min(Int(R_1) + \tau(R_1), Int(R_2) + \tau(R_2))$ where R_1 and R_2 are the two candidate regions for the merging and $\tau(R_i) = k/|R_i|$ favors the merging of smaller regions. k is a parameter that allows to roughly control the size of the regions. At each iteration, the maximum edge weight of the MST of any particular region is always the last edge used to merge elements of that region. In *Grundmann et al.* (2010) the authors build upon the algorithm of *Felzenszwalb and Huttenlocher* (2004) to produce a video segmentation algorithm[2] that tries to segment the image in temporally consistent regions. Note that the main segmentation we used as a basis

[1]Efficient graph based segmentation source code available here: `http://cs.brown.edu/~pff/segment/`

[2]Video segmentation web service at this address: `http://videosegmentation.com/`

to build the graphs representing the videos is the one presented in *Felzenszwalb and Huttenlocher* (2004). We also used the algorithm of *Grundmann et al.* (2010) for some experiments.

2.1.3 Edge Detection

Edges are composed of pixels that lie at the boundary between two regions with their own intensity or color. Those are useful to detect object boundaries. The most standard edge detector is Canny's edge detector *Canny* (1986). This detector first slightly blurs the image to prevent noisy pixels to have significant influence on the result. Then the image gradients on the x and y axis are computed to detect parts of the image with high spatial derivative. Edges in the original image are characterized by ridges in the gradient magnitude image. Those ridges are transformed into finer edges by applying a non-maxima suppression step. This consists in tracking pixels at the top of the ridges and set to zero the other ones. This tracking is controlled by two thresholds T_1 and T_2 with $T_1 < T_2$. The tracking of pixels on the ridges starts by a pixel higher than T_1 and continues in both directions until a pixel lower than T_2 is met. Figure 2.2 gives an example of edges extracted from an image with the Canny edge detector.

There exists many other methods to detect edges such as the Nalwa (*Nalwa and Binford* (1986)), Iverson (*Iverson and Zucker* (1995)) and Bergholm (*Bergholm* (1987)) algorithms. The quality of the different methods used to detect edges is diffifult to compare. The choice of a particular edge detector is dependant on the problem at hand. For example some applications require few false positive while other might perform better when fewer false negative are detected. For more informations on the subject, we refer the reader to the recent study of the different edge detection methods presented in *Oskoei and Hu* (2010).

2.2 Tools for Visual Tracking

Object detection and tracking are usually achieved through complex frameworks that make use of several tools to represent the objects, to estimate the motion of visual features or to locate candidate patches of images that could correspond to the target. This section presents the most common of those tools.

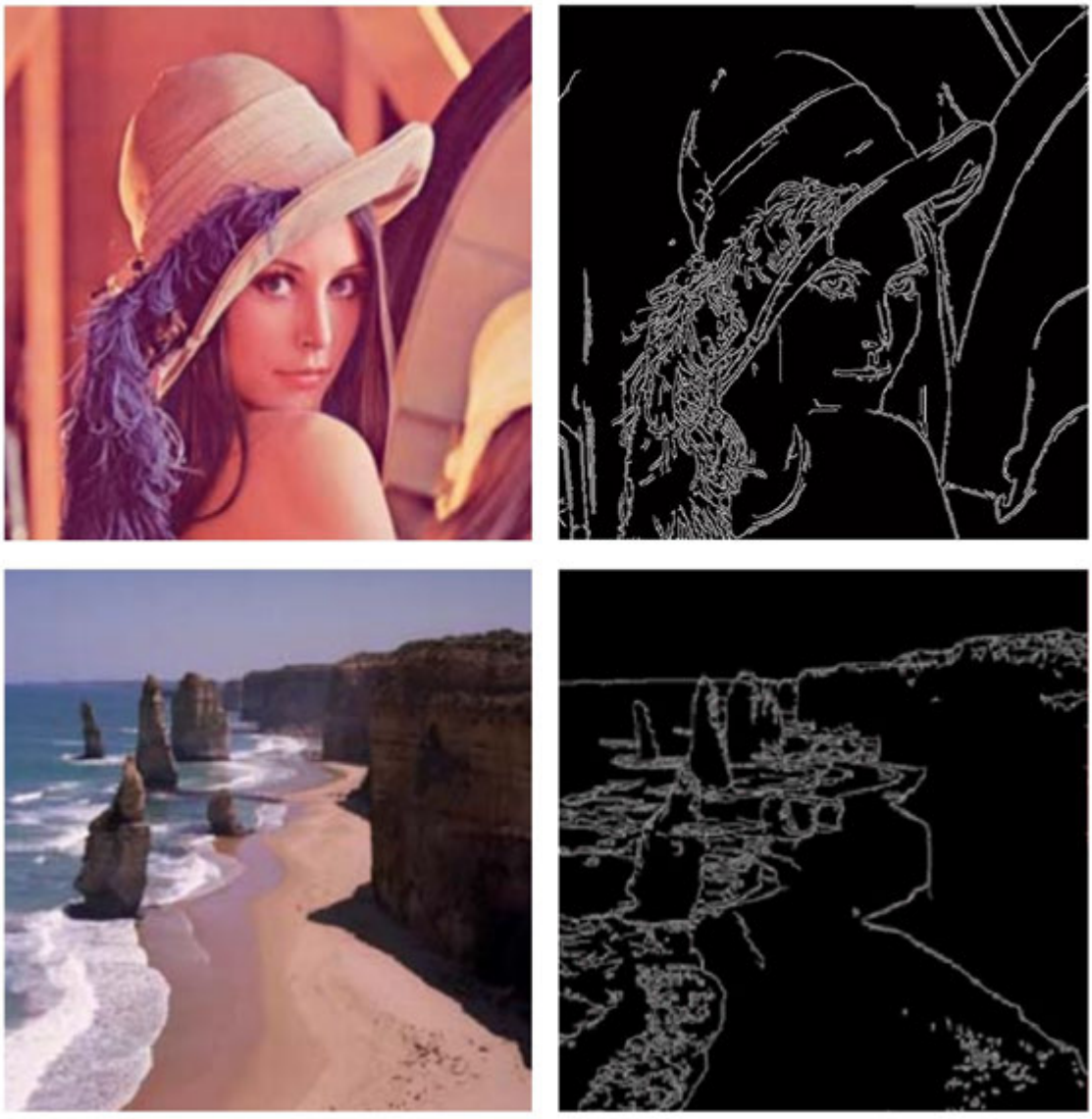

FIGURE 2.2: Edges detected with the Canny edge detector (*Canny* (1986))

2.2.1 Sliding Window

The most straightforward way of locating a set of features in an image is to use a sliding window. It simply consists in sliding a window over an image, possibly at different scales, and look for the position and scale that best fits the tracked set of features. This approach is used by object detectors such as face detectors or pedestrian detectors. Although it might seem that searching through the entire image is not computationally efficient, this strategy has been successfully used in real-time tracking applications, mainly by trackers using classifiers do detect the possible locations of the target, such as in *Kalal et al.* (2012).

2.2.2 Background Subtraction

Background subtraction techniques are commonly used to detect objects in videos filmed with a non moving camera. In those cases, since the background is stable, those techniques can try to learn a model of the background. Each incoming frame is compared to the background model and its pixels that do not fit the model are marked as foreground. Usually an algorithm is applied to find connected patches of foreground pixels, which can be used as input by various tracking approaches.

The simplest approach would be to keep an image of the background when no object is in the field of view and then subtract it to incoming frames. Each new object entering the field of view would be detected, but with no adaptation of the background model, if an object stops for a long time (e.g., a parked car), it will still be detected as foreground object. This could be solved by subtracting consecutive frames, but in this case an object would be lost as soon as it stops. It is clear that to be efficient, background subtraction techniques need more refined ways of modeling the background.

The work of *Wren et al.* (1997), which popularized background subtraction, presented a technique to learn gradual changes in time. The color of each pixel is modeled by a single Gaussian for which the mean and covariance are learnt over several consecutive frames. When a new frame is input, the likelihood of the color of each of its pixels with respect to the background model is computed. The ones with low likelihood are marked as foreground. This approach has been improved in *Stauffer and Grimson* (2000) by using a mixture of Gaussians to model the color of the pixels of the background.

More recent approaches such as *White and Shah* (2007) and *Hu and Su* (2007), try to deal with minor camera motion and dynamic background, e.g., leafs of a tree in the wind. In particular, the ViBe *Barnich and Van Droogenbroeck* (2011) gives very impressive real time results. The authors state that their algorithm could be applied on moving background by taking into account the movement of the camera, using embedded sensors or algorithmic techniques on the video stream, such as optical flow for example.

There exist a lot of other methods developed in the literature, and it would be impractical to study them in this short introduction to background subtraction. More details about background subtraction techniques can be found *Piccardi* (2004), *Benezeth et al.* (2008) or *Brutzer et al.* (2011).

FIGURE 2.3: Example of optical flow (*Sebesta and Baillieul* (2012))

2.2.3 Optical Flow

Optical flow estimation consists in estimating the motion of the different parts of a filmed scene (see Figure 2.3). This motion can arise from camera movement or object motion. Optical flow estimation techniques all make the assumption that the brightness and color of pixels stay constant when they flow between consecutive frames. This is called the *Brightness Constancy Constraint* and is characterized by the following equation :

$$I(x, y, t) = I(x + u, y + v, t + 1), \tag{2.1}$$

where $I(x, y, t)$ corresponds to the intensity of pixel (x, y) in frame t and $[u, v]$ is the motion vector relating the position of a pixel between the two consecutive frames.

Assuming the motion is small between t and $t + 1$, a first-order Taylor expansion can be applied on the right-hand side of equation 2.1:

$$I(x, y, t) = I(x, y, t) + u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t},$$

which simplifies to the *Optical Flow Constraint*:

$$u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0. \tag{2.2}$$

Unfortunately the *Brightness Constancy Constraint* of equation 2.1 and the *Optical Flow Constraint* of equation 2.2 only provide one constraint for the two unknowns u and v, this is called the *aperture problem*. In practice, optical flow techniques will be based either on the *Brightness Constancy Constraint* or *Optical Flow Constraint*

and turn the equation into an error per pixel, adding additional constraints to deal with the *aperture problem*.

For example, the original Horn-Schunck formulation of the problem (*Horn and Schunck* (1981)) combined the *Optical Flow Constraint* with a motion smoothness constraint favoring a small first order derivative of the flow field. The error per pixel of the equation 2.2 and the motion smoothness are summed over a sampling window using the L2 Norm. In this case, the resulting energy function to minimize is:

$$E = \sum_{x,y} \underbrace{\left[u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t}\right]^2}_{\text{pixel error}} + \sum_{x,y} \underbrace{\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right]}_{\text{motion smoothness}}.$$

The authors of *Sun et al.* (2010) showed that this classical Horn-Schunck formulation could give state of the art results when applied in conjunction with modern optimization techniques.

Another early approach, called the Lucas-Kanade algorithm (*Lucas et al.* (1981)), dealt with the *aperture problem* by considering the motion to be equal for pixels in a small neighborhood. This allowed the authors to use one equation per pixel of the neighborhood, resulting in an over determined system of equations that can be solved efficiently. More details about this technique are given in section 2.2.4.

One of the leading methods[3] was presented in *Sun et al.* (2010). The authors formulate a new objective function by adding a non-local term to the original objective of the Horn-Shunck method to integrate information robustness over large spatial neighborhoods.

During the 30 years of development of optical flow techniques, a large number of penalty functions and optimization have been described in the literature. Discussing all those methods would be out of the scope of this discussion. The interested reader can find an overview of the field in *Baker et al.* (2011).

2.2.4 Lucas-Kanade Tracker

First presented in *Lucas et al.* (1981), the Lucas-Kanade tracker is a popular short-term tracker (i.e., tracks features over a small number of frames). It is used to estimate the motion of pixels between two consecutive frames. While it was originally developed to compute optical flows, this approach is commonly used by tracking

[3]Optical flow source code of *Sun et al.* (2010) available here: `http://cs.brown.edu/people/dqsun/research/software.html`

algorithms to track sets of visual features representing a target between consecutive frames, providing an estimate of the motion of the target. It assumes that if the frames are separated by a small time increment δt, the motion is small and constant in a small window (usually between 2 to 7 pixels wide) around the tracked pixel. The idea is to find how much the window has to be moved so that intensities within it in a first frame are equivalent to the ones in the moved window in the next frame.

Formally, given two grey scale images A and B and taking for each tracked pixel a window W centered on its coordinates, we want to find the motion vector $\mathbf{v} = [v_x \ v_y]$ that minimizes the following error :

$$\varepsilon(\mathbf{v}) = \sum_{(x,y)\in W} (A(x,y) - B(x+v_x, y+v_y))^2, \tag{2.3}$$

where $A(x,y)$ corresponds to the intensity of image A at coordinates x, y.

At the optimum, the derivative of ε in function of $\mathbf{v}$ equals 0. Deriving 2.3 in function of $\mathbf{v}$ we obtain:

$$\frac{\partial \varepsilon(\mathbf{v})}{\partial \mathbf{v}} = -2 \sum_{(x,y)\in W} (A(x,y) - B(x+v_x, y+v_y)) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial x} \end{bmatrix}.$$

Since it is assumed that the displacement is small, $B(x+v_x, y+v_y)$ can be approximated by its first order Taylor expansion at the point (x,y) giving:

$$\frac{\partial \varepsilon(\mathbf{v})}{\partial \mathbf{v}} \approx -2 \sum_{(x,y)\in W} \left(A(x,y) - B(x,y) - \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial x} \end{bmatrix} \mathbf{v} \right) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial x} \end{bmatrix}. \tag{2.4}$$

$I_t(x,y) = A(x,y) - B(x,y)$ is the temporal derivative of the image and $\begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial x} \end{bmatrix} = [I_x \ I_y]$ the gradient composed of the two spatial derivatives on x and y, I_x and I_y. Using the assumption that there is only a small change between A and B, we can consider the spatial derivatives to be equal in both images, therefore:

$$I_x(x,y) = \frac{\partial A(x,y)}{\partial x} = \frac{A(x+1,y) - A(x-1,y)}{2}. \tag{2.5}$$

$$I_x(x,y) = \frac{\partial A(x,y)}{\partial y} = \frac{A(x,y+1) - A(x,y-1)}{2}. \tag{2.6}$$

Plugging those notations in equation 2.4 we obtain:

$$\frac{\partial \varepsilon(\mathbf{v})}{\partial \mathbf{v}} \approx -2 \sum_{((x,y)\in W} I_t(x,y) - [I_x(x,y) \ I_y] \mathbf{v}) \cdot [I_x(x,y) \ I_y(x,y)].$$

A slight transformation gives:

$$\frac{1}{2}\left[\frac{\partial\varepsilon(\mathbf{v})}{\partial\mathbf{v}}\right]^T \approx \sum_{(x,y)\in W}\left(\underbrace{\begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_xI_y(x,y) & I_y^2(x,y)\end{bmatrix}}_{G}\mathbf{v} - \underbrace{\begin{bmatrix} I_t(x,y)I_x(x,y) \\ I_t(x,y)I_y(x,y)\end{bmatrix}}_{b}\right).$$

The optimum displacement can be found by solving:

$$\mathbf{v}_{opt} = G^{-1}b. \tag{2.7}$$

G is invertible if the gradient of A is defined and non nul, i.e, the window is contained in the picture and there is enough contrast inside it. To ensure there is enough contrast in the window, the tracked pixels are usually interest points since those are detected based on their contrast.

The *aperture problem*, encountered by optical flow methods, is dealt with by assuming that pixels in a small neighborhood have identical motion. In this case, the system of equations 2.7 is overdetermined. Indeed using a 5 by 5 window results in 25 equations for two unknowns.

Note that the result can be improved by multiple iterations of the above computations by warping B using $\mathbf{v}$, i.e., translating the image according to $\mathbf{v}$, and recomputing a new optimal velocity vector. Let $\mathbf{v}^{(n)}$ be the optimal velocity vector obtained at the *nth* iteration and $B^{(n+1)}$ the image obtained by warping $B^{(n)}$ with $\mathbf{v}^{(n)}$. Here we can see why it is useful to compute the gradient of A instead of the one of B in equations 2.5 and 2.6. Indeed, in this way, at each iteration, only the time derivative changes. Therefore, only the matrix b needs to be recomputed by calculating the new time derivative $I_t^{(n)} = A(x,y) - B^{(n)}(x,y)$. For n iterations the optimal velocity vector is equal to $\mathbf{v}_{opt} = \sum_n \mathbf{v}^{(i)}$.

In order for the Taylor expansion in equation 2.4 to be a correct approximation, the small motion assumption has to hold. To deal with cases when the motion is too large, a pyramidal implementation can be used (see *Bouguet* (2001)). The Lucas-Kanade method is applied at different resolutions of the images starting with low resolutions and refining the results with higher resolutions.

The Lucas-Kanade approach can be used to compute the optical flow of frames giving information about the motion of each part of a frame. In *Kalal et al.* (2012), Lucas-Kanade tracker is used to estimate the position of the tracked object at the next time step. Features within the object bounding box are sampled and tracked to

estimate the displacement and scale change of the bounding box in the next frame.

2.2.5 Mean-Shift Tracker

First presented in *Fukunaga and Hostetler* (1975), the Mean-Shift procedure is used to locate the maxima of a probability density function given discrete data sampled from that function. Given an initial estimate x of where the maxima of the probability density function is, this method iteratively shifts x to the weighted average of nearby data samples (see Figure 2.4).

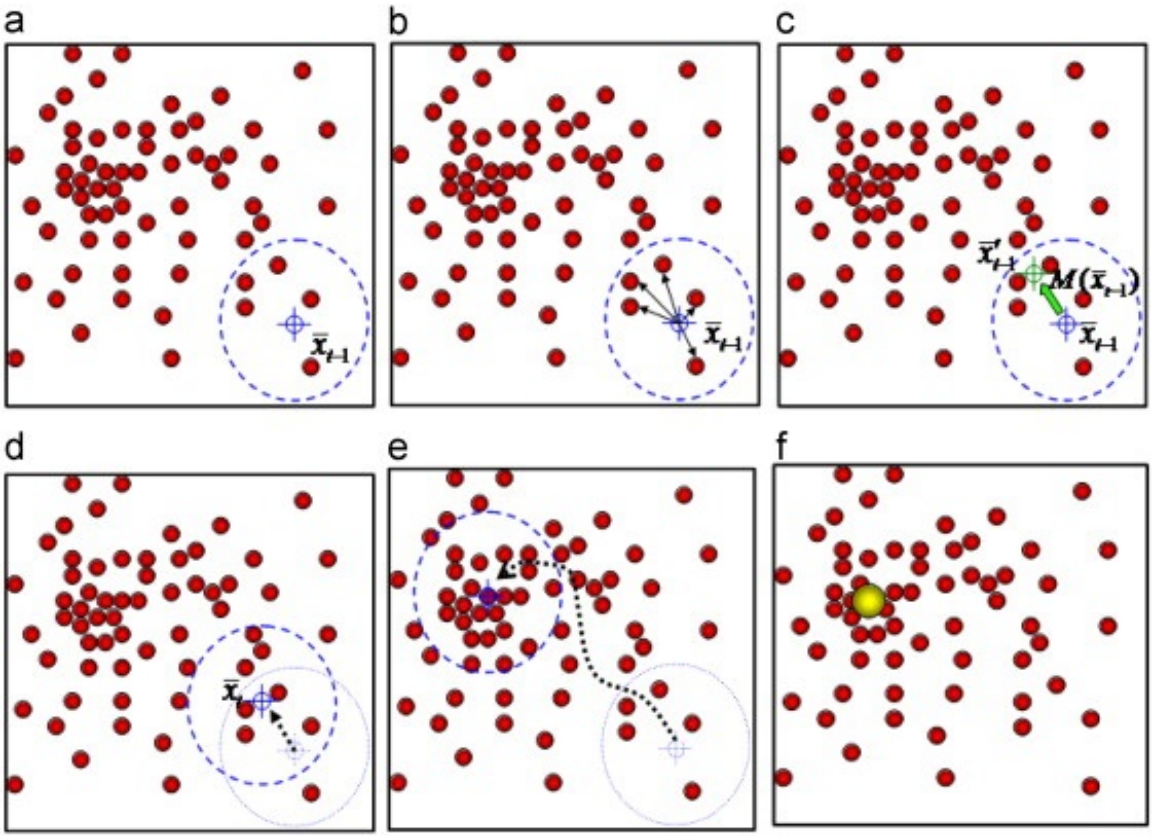

FIGURE 2.4: Graphical example of several iterations of the Mean-Shift procedure given noisy measurements observed through time (*Chang et al.* (2010)).

A kernel function $K(x_i - x)$ is used to determine the weight of the neighboring data samples x_i in the neighborhood $N(x)$ of x. The position of the weighted mean is

$$m(x) = \frac{\sum x_i \in N(x) K(x_i - x) x_i}{\sum x_i \in N(x) K(x_i - x)}.$$

This computation is iteratively repeated until $m(x)$ converges.

In the context of visual tracking, the Mean-Shift tracker is used in *Comaniciu et al.* (2003) to maximize the appearance similarity iteratively by comparing a weighted his-

togram representing the object and another one computed around the hypothesized object location. The similarity between histograms is computed using the Bhattacharya coefficient.

Another possibility is to use a classifier to produce a confidence map reflecting the probability for each pixel to correspond to the appearance model of the tracked object. The Mean-Shift procedure can be used to find the peak of this confidence map around the initial estimate of the position of the target such as in *Avidan* (2007).

2.2.6 Kalman Filter

The Kalman filter is a Bayesian filtering approach first presented in *Kalman et al.* (1960). Its purpose is to estimate the state of a system (e.g., position of a target, its motion, its size etc ...) given noisy measurements observed through time. For example, given a series of noisy radar measurements corresponding to a single target, a Kalman filter can be applied to filter those noisy measurements and estimate a trajectory closer to the real trajectory of the tracked object than the one that would be obtained by simply linking the observations together(see Figure 2.5). The kalman filter provides an optimal estimate of the true state of a system when the equations modeling the relationship between the variables are linear and when all the uncertainty can be modeled by Gaussians. Note that most systems are usually impossible to model exactly with a linear model, but if a good approximation can be found with linear equations, and if uncertainties are properly estimated with respect to the underlying system, a Kalman Filter will be able to handle inaccuracies and find good estimates.

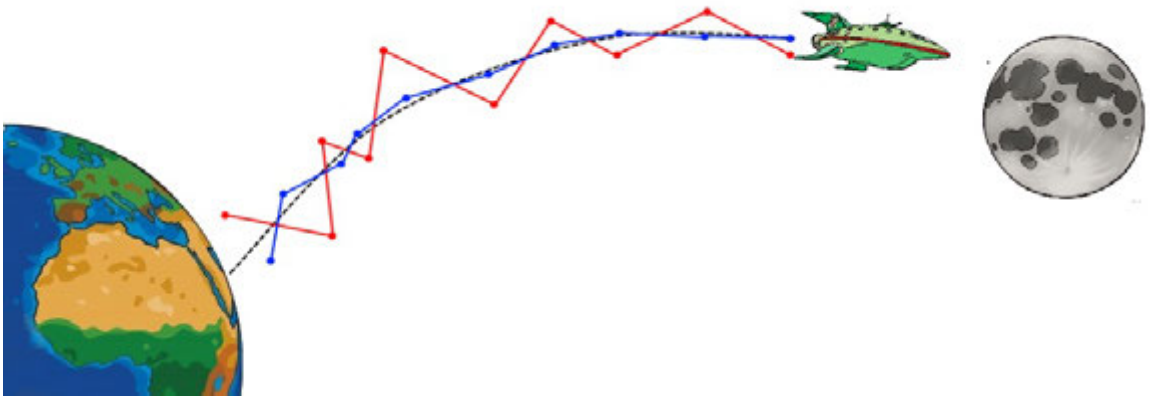

FIGURE 2.5: Kalman filtering example with the ground-truth trajectory of the spaceship in dashed black, the noisy measurements in red and the filtered trajectory in blue.

Formally, given the *posterior* state estimate $\hat{\mathbf{x}}_t$ of the system obtained at time t,

the *prior* estimate, $\hat{\mathbf{x}}_{t+1}^{-}$, of the state at time $t+1$ can be computed as follows :

$$\hat{\mathbf{x}}_{t+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_t + \mathbf{B}\hat{\mathbf{u}}_t, \tag{2.8}$$

where $\mathbf{A}$ is the state transition matrix, $\mathbf{u}$ the control inputs (e.g., human input provided by the steering wheel, break pedal, gas pedal...) and $\mathbf{B}$ the system of equations relating the control inputs to the system's state. In the visual tracking context, there usually is no control input, therefore $\mathbf{B}$ and $\mathbf{u}$ can be ignored. For example the Kalman filter state could be defined as the position of the target and its velocity ($\mathbf{x}_t = (p_x, p_y, v_x, v_y)^T$). In this case the state transition matrix, relating the position to the velocity, could be :

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ignoring $\mathbf{B}$ and $\mathbf{u}$; plugging this simple transition matrix A in equation 2.8 simply adds v_x to p_x and v_y to p_y to compute the transition from $\hat{\mathbf{x}}_t$ to $\hat{\mathbf{x}}_{t+1}^{-}$. In addition of the *a priori* estimation of the state, the *prior* state covariance $\mathbf{P}_t$, which measures the accuracy of the prediction step, must also be computed:

$$\mathbf{P}_{t+1}^{-} = \mathbf{A}\mathbf{P_t}\mathbf{A}^T + \mathbf{Q}, \tag{2.9}$$

where $\mathbf{Q}$ is the covariance of the noise associated with the prediction process.

After this prediction step, the second half of the Kalman filtering procedure, the correction step, consists in improving the estimate $\hat{\mathbf{x}}_{t+1}^{-}$ by combining it with the new measurement $\mathbf{z_{t+1}}$.

We first compute the Kalman gain which models the confidence we have in the estimate versus the measurement.

$$\mathbf{K}_{t+1} = \mathbf{P}_{t+1}^{-}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{t+1}^{-}\mathbf{H}^T + \mathbf{R})^{-1}, \tag{2.10}$$

where $\mathbf{R}$ is the noise associated with the measurement process and $\mathbf{H}$ the system of equations that converts between state and measurement (if the state and measurement are the same type of variable then $\mathbf{H}$ is the identity matrix). Finally we can compute the *posterior* estimate of $\mathbf{x}_{t+1}$ and the *posterior* state covariance $\mathbf{P}_{t+1}$ using the two following formulas :

$$\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_{t+1}^{-} + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\hat{\mathbf{x}}_{t+1}^{-}). \tag{2.11}$$

$$\mathbf{P}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})\mathbf{P}^{-}_{t+1}. \tag{2.12}$$

In equation 2.12 the posterior state covariance is updated according to the confidence in the prediction step encoded in the Kalman gain.

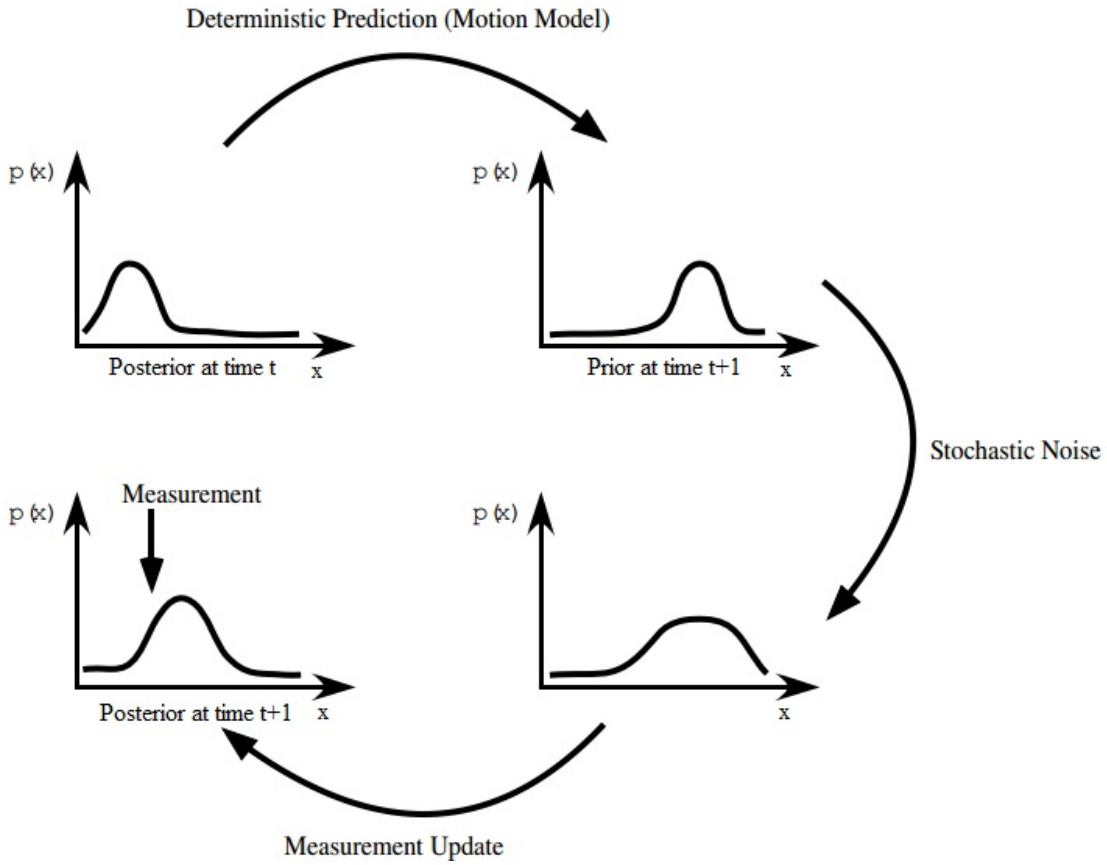

FIGURE 2.6: Main steps of the Kalman filter (*Cannons* (2008))

Repeating this process for each measurement will produce a trajectory that will be closer to the true trajectory of the object that emitted the noisy measurements. Note that the estimation and measurement covariances $\mathbf{Q}$ and $\mathbf{R}$ are parameters and the initial estimate of the state $\hat{\mathbf{x}}_0$ is also given as input.

Figure 2.6 summarizes the Kalman filtering steps. The prediction step, which corresponds to the computation in equation 2.8, influences only the mean of the Gaussian. After computing the Kalman gain in equation 2.10, the stochastic noise and measurement update steps are both applied simultaneously in equation 2.11 to obtain the *posterior* state estimate at time $t + 1$.

Two drawbacks can limit the usage of Kalman filters. First, if uncertainty cannot be modeled by Gaussians the error covariance $\mathbf{P_t}$ might not converge resulting in bad

estimations. Secondly Kalman filters provide optimal estimates only if the underlying system is linear. The Extended Kalman Filter (*Bar-Shalom* (1987)) has been designed to deal with non linear system but requires more complex computations.

In case of multiple target tracking, each target can be tracked by its own Kalman filter, but if the correspondence between the measurements and the targets is not known, a mechanism must make a choice to pick which measurement will be used to update each filter.

In the context of tracking, Kalman filters are not always used to estimate the sate of a target. For example, in *Duan et al.* (2012), the authors track groups of objects and use a Kalman filtering approach to compute the changes in relative location between objects in order to build a mutual relation model reflecting the strength of their relation.

2.2.7 Particle Filter

First presented in *Isard and Blake* (1996), the particle filter is another type of Bayesian filter. Based on sequential Monte Carlo sampling methods, particle filters estimate the *posterior* density of the state variables (e.g., position of a target, its motion, its size etc ...) with random samples. This removes the major restrictions of the Kalman filter by allowing the use of non linear equations for the prediction and update as well as multi-modal distributions instead of Gaussian ones to model the errors. As shown in Figure 2.7, the particle filtering procedure is conceptually very similar to the Kalman filter.

Each individual particle corresponds to one hypothesised state. In the context of visual tracking, each particle could be a vector $\mathbf{s}_t^n = (p_x, p_y)$ representing an hypothetical position of the target. Each particle is assigned an importance weight π_t^n reflecting the quality of the hypothesis.

Given a set of particles $S_t = (\mathbf{s}_t^1, ..., \mathbf{s}_t^n)$ and their importance weights $\Pi_t = (\pi_t^1, ..., \pi_t^n)$ at time t, the first step of the particle filter procedure consists in sampling a new set of particles $\Pi_{t+1} = (\pi_{t+1}^1, ..., \pi_{t+1}^n)$ at time $t+1$ with respect to their importance weights. This is done by picking particles in S_t with replacement (the same particle can be picked several times) in function of their weight. Then, the deterministic prediction step is performed by computing the new position of each particle at time $t+1$. This corresponds to the computation of equation 2.8 of the Kalman filtering procedure. The difference here is that any motion model can be used whereas the transition matrix of Kalman filters has to be a linear system of equations.

Because the prediction step is deterministic and the particles in S_t can be picked several times, especially if they have a high importance weight, it is likely for some

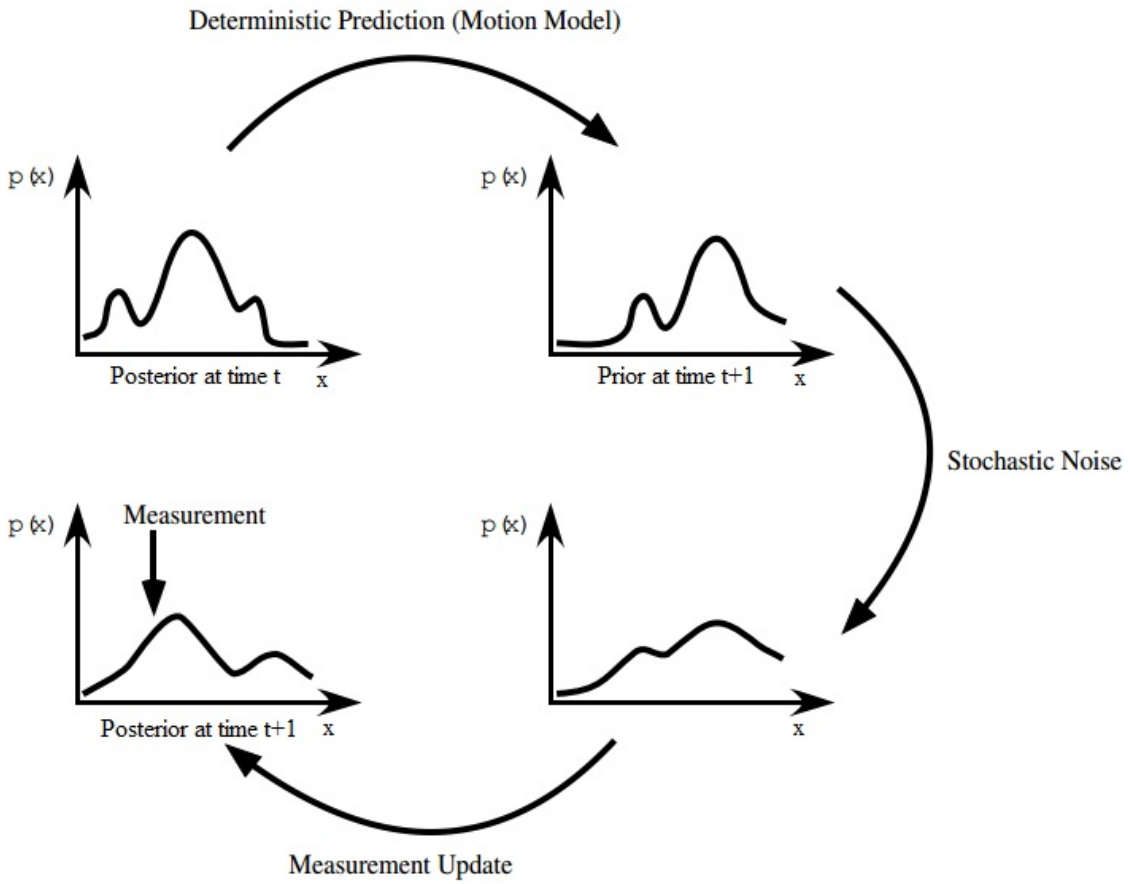

FIGURE 2.7: Main steps of the particle filter (*Cannons* (2008))

of the resulting particles to be identical. Therefore, the particles are diffused by taking into account the estimation noise to produce a more diverse set of samples. This corresponds to the stochastic diffusion step. In the case of particle filters, the uncertainty of the prediction can be modeled by non Gaussian distributions. Note that, although a motion model can help to perform more accurate predictions, this stochastic diffusion step can be enough by itself to predict the new set of particles at time $t+1$ from the set of particles at time t and track a target. For example, in *Erdem et al.* (2012), no motion model is used to predict the new set of particles, the set of particles at time $t+1$ is obtained solely by diffusing the particles obtained at time t with a Gaussian.

Next, the measurement update step is applied. It consists in computing new importance weights. This can be done by measuring the similarity of the data observed at the particle state (e.g., color histograms around the position of the particle, optical flow, etc...) with an object model. For example, if the target's appearance is modeled by a color histogram, the new importance weights could correspond to the similarity

between the color histogram measured at the position of each particle and the color histogram of the object model. Note that while Kalman filters require measurements to be convertible into state vectors with the linear system of equations H to compute the *posterior state estimate* a time $t+1$ in equation 2.11, particle filters do not have such restrictions. They only need a mechanism to compute how much each hypothesis modeled by each particle is in agreement with the object model.

Once the new particles have been assigned their corresponding importance weight, the particle filter procedure is over and can be reiterated for the next time steps. At each step, the resulting state of the target can be obtained by applying the meanshift procedure described in section 2.2.5 or simply by computing the mean of the particles or by selecting the particle with highest probability (i.e., highest importance weight). Figure 2.8 shows the evolution of the samples and their weights when tracking a target for one time step.

In *Erdem et al.* (2012) the authors represent the object with two fixed grids of rectangular image patches (Figure 2.9). They use a particle filter to track the object. The importance weight of each particle is computed by measuring the similarity of the image patches in the original grid in the first frame with the patches obtained by positioning the same grid at the particle coordinates.

2.3 State-of-the-Art Trackers

In this section we will discuss recent approaches that have contributed to the development of the field of visual tracking.

2.3.1 Part-Based Tracking

Part-based trackers try to preserve the spatial relationship between pixels by representing the target by multiple parts. They are often used to track humans by representing them with parts for the limbs, torso and head as in *Nejhum et al.* (2010). Representing an object with parts has the advantage of keeping some information about the relative spatial arrangement of its different pieces. They usually need the model of the object to be known a priori.

In this section we will discuss two different part-based trackers that only require the target to be tagged in the first frame.

A first approach, called FragTrack presented in *Erdem et al.* (2012) uses two fixed rectangular grids applied at the initial user selected region. The color histograms of each cell is computed and serves as appearance model T to represent the object (overall 36 patches; 18 vertical and 18 horizontal; see Figure 2.9).

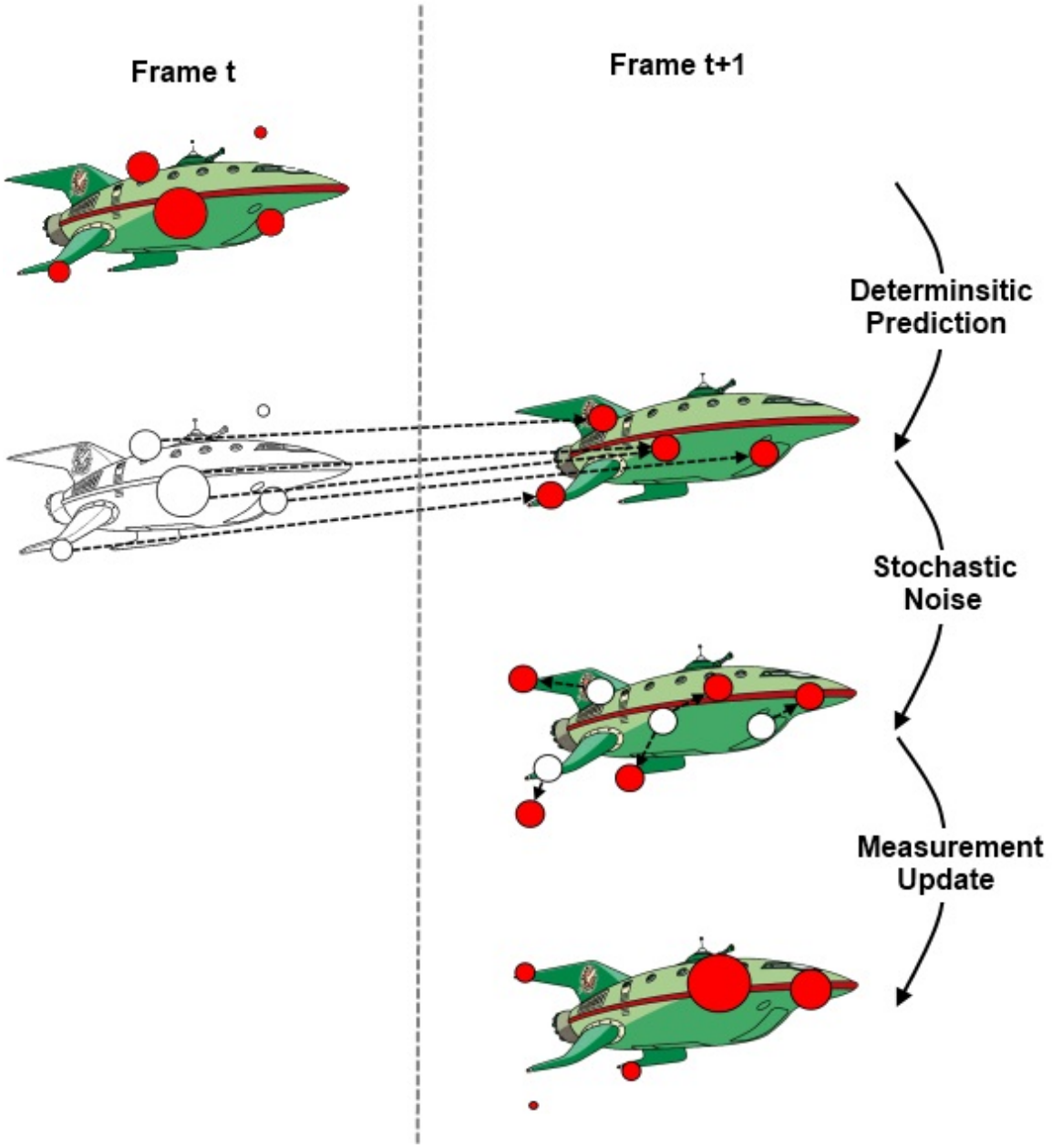

FIGURE 2.8: Evolution of the particles (in red) for one time step. The size of each particle corresponds to its importance weight. First, particles at time t are selected according to their importance weight and their new state vectors in frame $t+1$ are predicted according to some motion mode. Then the state vector of each particle is randomly shifted to account for the prediction uncertainty. Finally, importance weights are recomputed based on the agreement between the object model and the image patch measured at the position of each particle.

To track the object the authors use a particle filter in which each particle $\mathbf{s}_i^t = (x, y, s_x, s_y)$ corresponds to an hypothesized location and scale at time t. Note that no motion model is used in this approach, the prediction of the new state vector of each particle at time $t+1$ is achieved by only diffusing the particles using a Gaussian

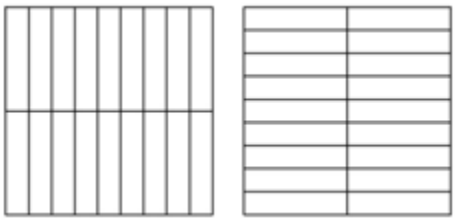

FIGURE 2.9: FragTrack's grids of image patches used to represent the target in *Erdem et al.* (2012)

$\mathcal{N}(\mathbf{s}_i^t, \Lambda)$ with mean $\mathbf{s}_i^t$, and covariance $\Lambda = diag(\sigma_x^2, \sigma_y^2, \sigma_{s_x}^2, \sigma_{s_y}^2)$.

After predicting the new state vector of each particle, two grids, with the same layout as the appearance model, are positioned at the location and scale of each particle to compute their importance weights. The color histograms of the cells of the grids of each particle are compared to the one of the appearance model.

Each cell of the appearance model is weighted to reflect its reliability. Those weights determine the contribution of each cell to the joint result computed when comparing the cells of the grids of each particle to the ones of the appearance model. The weights are updated after each time step in function of the agreement between each cell and the joint result of all cells. Thanks to this adaptation process the tracker is capable of dealing with changes in appearance and partial occlusions of the target.

This approach can be performed in nearly real time (≈10 fps) and gives very good qualitative results. Although it is important to note that only the weights of the different parts of the template are updated, the color histogram of each image patch stays fixed. Therefore, the algorithm is very likely to loose the target if its appearance changes too much.

In the recent article from *He et al.* (2013), the authors build upon this method to achieve very good results both in term of quality of the results and in term of computational efficiency. The authors use the same kind of grid as in *Erdem et al.* (2012) but with around 400 cells. They also use weighted histograms to model the appearance of each cell of the grid, i.e., histograms in which the contribution of each pixel of the image decreases exponentially with its distance from the center of the cell.

In *Chockalingam et al.* (2009) the authors present a very different approach. All the images are individually segmented using a region growing method. This consists in selecting a pixel at random and putting it in its own image region. Then, neighboring pixels are added if they are within some τ (provided as parameter) standard deviations of the Gaussian model representing the region. The constraint is relaxed for small regions that do not have enough pixels for their Gaussian model to be reliable. This

process is repeated until all pixels are in a region. The contour of the object is modeled using a level sets formulation (*Brox et al.* (2006); *Osher and Sethian* (1988)). The final object model is composed of the contour and the regions R^- inside it. The background is modeled using all the regions R^+ outside the contour. Also the motion of each fragment is computed by optical flow using *Birchfield and Pundlik* (2008). At each time step, the position of the object is estimated by using the average motion of the regions composing it. Then the contour evolves to maximize the likelihood of the pixels inside it to be part of the object, and minimize their likelihood to be part of the background. The likelihoods are given by a Gaussian mixture model (one Gaussian per region). According to the authors this algorithm manages to track targets undergoing strong deformation, unpredictable motion and complete occlusion.

Part-based algorithms can track targets with unstable appearance but they are difficult to update to take into account appearance changes without risking that the tracker drifts from the target. Besides they require domain knowledge if the layout of the parts in the appearance model is to reflect the target's topology.

2.3.2 Segmentation Based Approaches

Some applications require a precise segmentation of the target. To achieve this, segmentation based approach use the segmentation obtained at time t (the initial segmentation at t=0 is given as input) to guide the segmentation at time t+1. A segmentation of the background and N objects can be obtained at each time t by minimizing the following energy function:

$$J(\lambda) = \varepsilon_D(\lambda) + \varepsilon_R(\lambda), \tag{2.13}$$

where λ is a labeling function that assigns a label $l \in 0...N$ to each pixel (one label l corresponds to one object, with label 0 for the background), and ε_D is the data term measuring the likelihood of a pixel being assigned a certain label and is calculated as follows:

$$\varepsilon_D(\lambda) = -\sum_{x \in \Omega^t} \sum_{l=0}^{N} ln(P_l(x))\delta(\lambda(x), l), \tag{2.14}$$

where Ω^t is the set of all pixels of the frame number t, $P_l(x)$ the likelihood of pixel x corresponding to object l, and $\delta(l_1, l_2) = 1$ if $l_1 = l_2$. The regularization term ε_R of equation 2.13 is:

$$\varepsilon_R(\lambda) = R_\Omega \sum_{x \in \Omega^t} \sum_{z \in \mathcal{N}} F(x, z)[1 - \delta(\lambda(x), \lambda(z))], \tag{2.15}$$

with $R_\Omega > 0$ the regularization parameter and $F(x_1, x_2)$ a similarity function between pixels. This formulation penalizes similar pixels in the same neighborhood having different labels.

Usually the minimization of the energy function of equation 2.13 is performed through a Graph-Cut approach such as in *Boykov and Funka-Lea* (2006). It consists in building a graph $G = (V, E)$ with nodes corresponding to the pixels of the image. One terminal node per label is added. Edges connect each pixel with pixels of its neighborhood and each terminal node to each pixel. A cut C is a subset of all edges that meets the following requirements:

- Each node is connected to at most one terminal node.
- Two nodes connected to two different terminal nodes cannot be connected together.

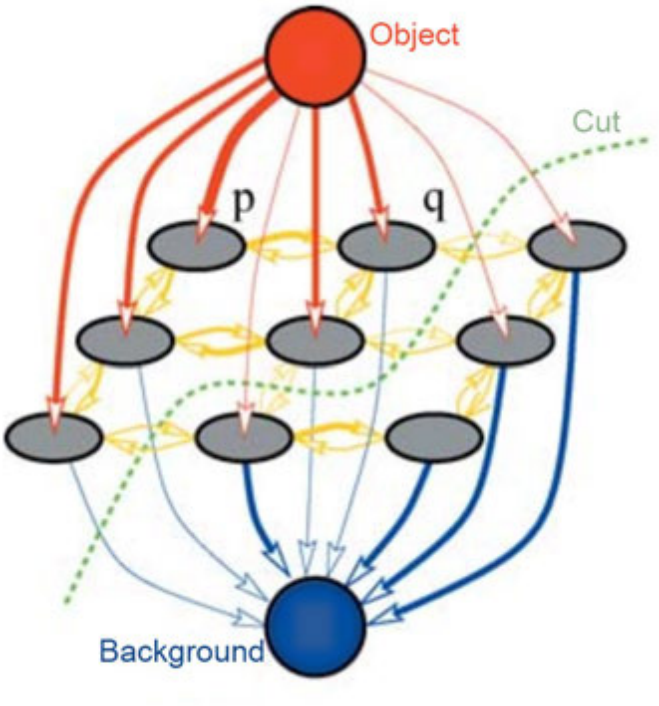

FIGURE 2.10: Example of graph cut with one object to segment from the background (*Ramírez et al.* (2012))

The aim is to find the cut that minimizes the energy function $J(\lambda)$. (see Figure 2.10 for a graphical representation of a cut). In the context of visual tracking, this

simple energy minimization framework is not sufficient. Indeed, the temporal dimension has to be taken into account to achieve tracking. In *Malcolm et al.* (2007) the authors use the mean velocity of the object i in past frames to predict the set of pixels $V_i^{t+1|t}$ that is likely to correspond to the object in frame t+1 and that should be labeled l_i. The authors add a third term ε_γ to the energy function that penalizes assigning a different label than l_i to the pixels in the predicted set.

This simple approach suffers from the lack of mechanism to recover from partial occlusions and therefore quickly loses the target when it is occluded.

To deal with this problem *Papadakis et al.* (2011), represents the tracked objects with two sets of pixels, the visible part and the occluded part. At each time step t, the Lucas-Kanade algorithm is used to estimate the position of the pixels representing the object in frame t+1. New terms are added to the energy function to handle the occluded part. Again the energy function is minimized using a graph cut algorithm. The results presented show that this approach, unlike *Malcolm et al.* (2007), can deal with partial occlusions. In one sequence a pedestrian is almost completely occluded by a vehicle during around 30 frames and is still well recovered.

A different approach is presented in *Tsai et al.* (2012). In this paper a multi-label Markov Random Field (MRF) is used to automatically segment and track the targets. The authors minimize the energy function $J(\lambda) = \varepsilon_D(\lambda) + \varepsilon_R(\lambda)$. The difference here comes from the fact that the temporal consistency is not represented by a term in the energy function. Instead pixels in frame t are connected to their neighbors in frame t+1. The temporal neighborhood of a pixel in frame t consists in all pixels in frame $t+1$ that are spatially close enough (closer than a maximum displacement threshold). With each label is associated a motion field $d^1, ..., d^i$ such that assigning label lp to pixel p means it will be be displaced by d^p. If the maximum displacement on x or y is m there is $(2m+1)^2$ possible labels and since each pixel can also be labeled background or foreground the total number of possible labels for each pixels is $2 * (2m+1)^2$. Associating labels with a displacement enforces motion coherence through the regularization term. Indeed the regularization term encourages neighboring pixels in the same frame to have the same label which in this case also means that they should have the same displacement. This approach gives good results in term of precision but no occlusion test is conducted and since no mechanism is present to deal with occlusions, this approach is unlikely to perform very well when targets are occluded.

Segmentation based techniques can accurately follow deformable objects with high precision. At each time step, they output a foreground region of pixels corresponding to the target. They cannot be applied to entire frames, hence they rely on an accurate

motion model to predict the position of the target in subsequent frames. Thus, they usually deal poorly with long term occlusions and interactions between objects. It also requires more effort from the user to provide the initial segmentation than a simple bounding box.

2.3.3 Tracking by Detection

Tracking by detection consists in iteratively training a classifier to separate the object from the background of each frames. At each time step only the previous frames of the video are used for training. The main challenge of those approaches is to update the classifier as new information is provided by the video frames.

In *Zhang et al.* (2012) the authors present a simple technique called CT[4] (for *Compressive Tracking*). The tracking from a frame t to a frame $t+1$ is achieved by first sampling positive samples around the object location at time t and negative samples far away from the object and use both negative and positive samples to build a Bayesian classifier. Then locations in frame $t+1$ around the position of the object at time t are rated using the classifier and the one with the highest score is selected as position in the frame $t+1$. The originality of CT comes from the fact that image patches are represented by a sparse feature vector obtained through random projections of the image features. More precisely, for each sample, its different representations at multiple scales are concatenated into a high dimension vector. Using a sparse random matrix, this high dimension vector is projected onto a lower dimension vector which is used as final representation of each sample. Such a compression of the representation of the samples preserves salient information while being robust to noise. It allows the algorithm to handle changes in pose, illumination and scale as well as partial occlusions. Results show that this algorithm is very fast. Indeed, in their experiments the authors demonstrate that it can process around 35 images per second, while average state of the art tracking by detection algorithms handle around 10 images per second. This approach also gives very good precision results. However, it is expected to perform poorly when multiple similar targets interact with each others.

A different method, called TLD[5] (for *Tracking Learning Detection*), is presented in *Kalal et al.* (2012). Here, the object is represented by a collection of positive and negative image patches M. Positive patches are ordered in function of their time of

[4]source code of CT available here: http://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm

[5]source code of TLD available here under the name OpenTLD: http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html

addition to M. At each time step the image patch representing the target is tracked by a Lucas-Kanade tracker. The algorithm also performs a window search on the frame, sampling windows of the size of the initial bounding box at multiple locations and scales. Those image patches are compared to the object model using a Nearest-Neighbor classifier to detect possible locations of the target in the whole frame. The image patches obtained by both the tracker and the detector are compared to the earliest 50% of the positive image patches in M. The one with maximum confidence is returned as the current position. The collection M is updated with the following strategy:

- If the new location estimated by the tracker is labeled as negative by the Nearest-Neighbor classifier, it is added to M as a positive image patch. This increases the generalization capability of the object model.

- All responses returned by the detector and the response produced by the tracker are analyzed. The image patches that do not overlap with the maximally confident patch are added to M has negative image patches. This increases the discriminative power of the object model.

This real-time approach gives very accurate results, even under strong and unpredictable motion. Thanks to the detector, this algorithm can recover from short disappearances of the target.

Most tracking by detection approaches only use the current frame to update the object model at each step. In *Supancic III and Ramanan* (2013), the authors show that the appearance model is more important than the motion model for long term tracking and that performances can be improved by learning from the "right frames". Possible locations of the target in the current frame are detected by a linear SVM. The detections up to the current time step are structured into a graph, with one node representing one detection. Between each pair of frames, edges connect the nodes and represent the cost of transitioning from on location to the other based on the negative SVM score of the destination image patch and the agreement between the optical flows of both locations. After each time step, the SVM is retrained from previously tracked frames by selecting the ones that minimize the SVM objective. In each of the selected frames, the patch in the bounding box of the tracked object is used as positive sample and patches around the bounding box as negative samples.

Tracking by detection method are really effective real-time single target tracking methods. They can cope with strong and unpredictable motion and adapt to appearance changes. However this adaptation capability makes them prone to drift-

ing problems. Also, in the case of multiple objects with similar appearance, those methods can easily switch between different targets.

2.3.4 Trackers Exploiting Context

A problem encountered by a lot of single target trackers is to distinguish the target with other similar objects in the scene, called *distracters*. For example, if two pedestrians are walking together, single target trackers can easily confuse them, especially if they wear similar clothings. In recent years, multiple approaches have tried to deal with the presence of multiple similar targets by keeping track of *distracters* with which the target should not be confused.

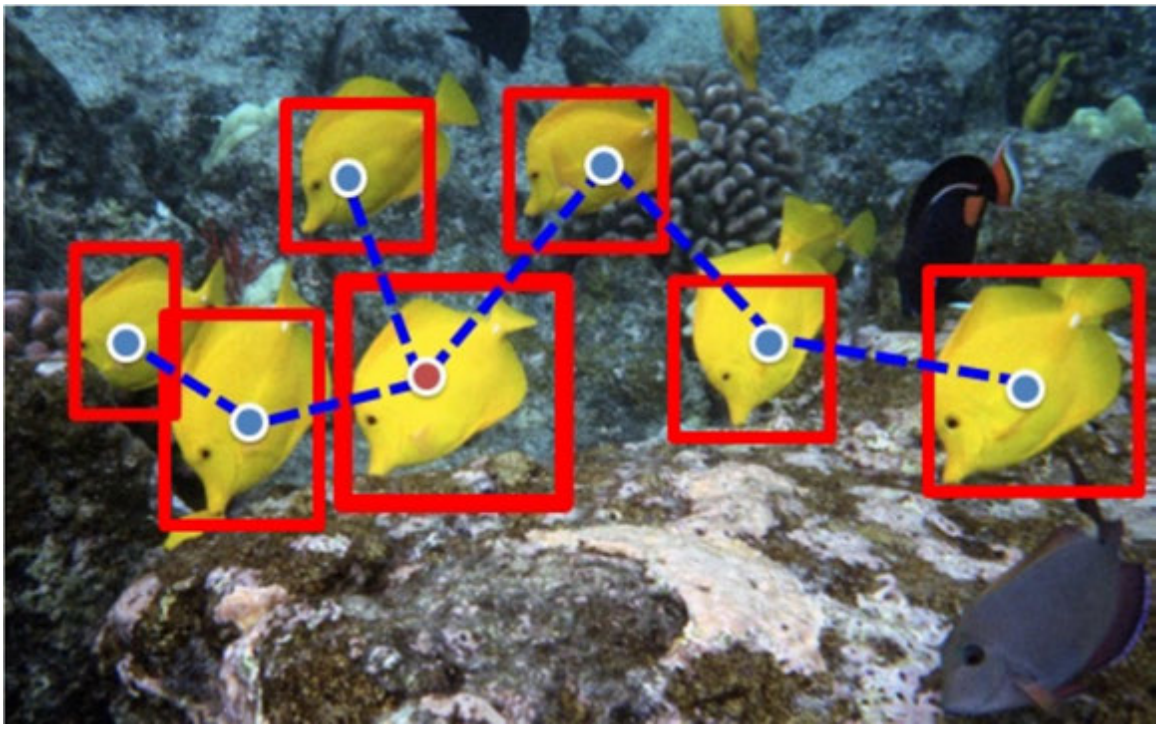

FIGURE 2.11: Example branching tree used in *Gu and Tomasi* (2011). Each node represents a tracker and an edge connects two trackers if one has been branched from the other in a previous frame

In *Gu and Tomasi* (2011), the authors presented an approach that deals with the presence of similar objects by branching the tracker when *distracters* are detected. This tracker builds upon the tracking by detection algorithm of *Gu et al.* (2011), which uses a nearest neighbor classifier to distinguish between SIFT interest points belonging to the target and to the background. This initial tracker is used to determine the optimal location of the target with respect to visual and motion consistency. It is branched, i.e, a new tracker is spawned, to track each other possible locations that

are similar to the target and do not overlap too much with the best location. This branching operation defines a parent relationship between trackers in each frame k that is modeled in a branching tree $G_k =< V, E >$ where nodes in V represent trackers and the edges in E model the fact that a tracker has been branched from another one in a previous frame (see Figure 2.11). The optimal branching tree is built in each frame by finding the one that minimizes the following equation:

$$\hat{G}_k = \underset{G_k}{argmin} \left\{ \underbrace{E(G_k)}_{\text{visual consistency}} + \lambda \underbrace{\kappa(G_k, G_{k-1})}_{\text{structural difference}} \right\}, \tag{2.16}$$

where G_k is a branching tree in frame k and λ is a regularization parameter between the visual consistency and the structural difference. The appearance model used by the tracker is composed of an object model and a background model. The visual consistency of each tracker is measured by summing the costs of each feature in the window with respect to its closeness to the object and background models. The visual consistency term in equation 2.16 $E(G_k)$ corresponds to the sum of the visual consistency of each tracker in the branch tree. The structural difference compares the spatial geometry of the branching trees of consecutive frames. It is obtained with the following equation :

$$\kappa(G_k, G_{k-1}) = \sum_{(i,j) \in E_{k-1}} ||(W_k^i - W_k^j) - (W_{k-1}^i - W_{k-1}^j)||^2, \tag{2.17}$$

with (i, j) an edge in G_{k-1} between the trackers i and j, and W_k^i the spatial coordinates (x, y) of the window corresponding to the ith tracker in frame k. The structural difference computation in equation 2.17 favors low difference between the relative spatial distances separating pairs of trackers in consecutive frames. This encourages the optimal branching tree $\hat{G}_k$ to be geometrically similar to G_{k-1}. Once the optimal branching tree has been found, branching operations are performed on trackers in $\hat{G}_k$ that meet the branching constraints. Then the object and the background models are updated before extracting candidate locations in the next frame. Experiments show that this strategy can successfully follow the correct target in the presence of 2 to 20 similar objects. Its main limitations are the absence of mechanism to deal with changes in scale and that the branching tree never shrinks to take into account trackers losing their target.

Another way of exploiting the context to track a target is presented in *Grabner et al.* (2010). Instead of exploiting *distracters*, this approach uses *supporters*, i.e., features that have a correlated motion with the target. *Supporters* can belong to the

target, such as a watch on the wrist of a pedestrian for example, but they can also belong to the rest of the scene. The idea of this technique is to use few reliable target locations, provided by a human or a robust tracker, to learn a model, which in this case is a set of *supporters*, to predict the position of the target. In this work the *supporters* are SIFT interest points. At each iteration, features are extracted and the ones that match the model are used to vote on the position of the target based on the correlation of their motion with the motion of the target. If the confidence with the current estimate of the target location is high enough, a second set of *supporters* is extracted. This second set of *supporters* helps estimating the position of the target but is not used to assess the confidence of the estimate. This means they cannot trigger the extraction of more *supporters* which avoids a feedback loop that can cause the tracker to drift from the target. The authors use this strategy in combination with the tracker presented in *Stalder et al.* (2009). When this tracker loses the target the trained model of supporters is used to infer the position of the object, even if it is completely occluded or out of the field of view. As shown in the experiments, this approach allowed to significantly improve the recall of the target.

Supporters are also exploited in *Yang et al.* (2009) to track a target using context. Here the *supporters* are obtained through data mining. Each image of the video is segmented individually. Then, the resulting segmented regions are clustered to form a vocabulary. This is done incrementally by adding to the vocabulary the image segments of each incoming frame. Each element of the vocabulary is treated as an item, and transactions are formed by the items and the other items in their neighborhood. In each frame the items in the region of interest are used to build transactions, and a transaction database is formed by the transactions constructed from all frames in a sliding window. Items with high co-occurrence are then selected as candidate *supporters*. A mean-shift tracker is used to find the correspondence of each candidate *supporter* in successive frames. This is used to estimate the motion of the candidate and test if it is correlated with the motion of the target, in which case it is validated as *supporter*. Then, a message passing mechanism is used to refine the motion estimate of the target. First the *supporters* send their motion estimate to the target which updates its own estimate, and then the target propagates the messages back to them. Those messages correspond to the motion estimate of the *supporters* weighted according to their correlation with the target, which results from the co-occurrence measured during the mining step.

Finally, in *Dinh et al.* (2011), the authors presented an approach exploiting both *distracters* and *supporters*. Their technique uses an extended version of the tracker presented in *Kalal et al.* (2010). The object model of *Kalal et al.* (2009) is used to

represent the target. It consists in a template of image patches representing the target in previous frames and organized in a binary tree. At each iteration multiple candidate locations are detected. To distinguish between them, *supporters*, in this case SURF interest points (*Bay et al.* (2006)), are extracted around those candidate locations. The *supporters* are then matched with the ones extracted in the 4 previous frames. If the number of matched *supporters* around a candidate location is high enough it is placed in a list of strong candidates. The candidate with highest confidence in this list is selected as the target and other candidates are greedily associated to the *distracter* trackers. The remaining candidate locations that have high confidence and are not already tracked trigger new *distracter* trackers. The object model of each tracker is updated and the process starts over with the next frame. The experiments conducted by the authors show that this technique can follow a target in difficult scenarios where other state of the art trackers tend to lose it.

Those trackers exploiting *supporters* and *distracters* require more computations than simpler ones that ignore context but can still manage to process a video in nearly real time. However, they possess the advantage of considering multiple targets, which helps to distinguish the correct one in ambiguous cases, without being as complex as conventional multi-target trackers.

2.3.5 Data Association for Multi-Target Tracking

A lot of multi-target tracking approaches use an object model learnt offline to detect locations of the objects in each video frame. Data association consists in partitioning the set of detections into tracks representing targets. Most of those approaches use object detections as a basis and assume a one to one mapping between object detections and targets. This means that each object detection can be claimed by at most one track, and only one object detection is selected by each track at each time step. Although some approaches like *Yu and Medioni* (2009) do not make the one to one mapping assumption, this section will focus on the former family of approaches that are more closely related.

A first method consists in casting the problem as the minimization of a network flow. In *Zhang et al.* (2008) a network between detections is created. Each detection is represented by two nodes, with an edge between them representing the probability of the observation to be the true detection (modeled by a Bernouilli distribution). The probability of two detections to be part of the same track is modeled by adding an edge between the two pairs of nodes representing the two detections. The weights on those edges are based on the distance between the color histograms of the two detections, their difference in size and the number of frames separating them. Each

edge has a maximum flow capacity of one to prevent single detections to be part of multiple tracks. Then the Min-cost flow is solved using a push-relabel approach (*Goldberg* (1997)). *Pirsiavash et al.* (2011) and *Berclaz et al.* (2011) use a k-shortest path optimization (*Ahuja et al.* (1993)) to solve the Min-cost flow problem much more efficiently. The idea is to find the k-shortest paths, starting from the source node of the network and ending at the sink node, such that the sum of their weights is minimal.

Another family of data association approaches uses iterative hierarchical methods to link tracklets together, i.e., small tracks composed of a few detections. In *Brendel et al.* (2011), a graph is built in which nodes represent every pair of object detections in consecutive frames. An edge is put between two nodes if they are in conflict, i.e., if they share one of their object detections. Nodes are weighted by measuring the similarity between the two object detections composing them. This similarity measure is learnt over time to make observations in a same track more similar between each others than with detections in other tracks. The produced graph is composed of several independent graphs, each one of them representing all possible associations between detections in one frame and the detections in the next one. In this setup finding the best tracks corresponds to solving the Maximum Weight Independent Set (MWIS) problem. Indeed, since conflicting nodes are connected by an edge, finding the heaviest subset of nodes that are not connected corresponds to finding the best association between detections in one frame and detections in the next one, producing tracks of size 2. This procedure is repeated iteratively by building a new graph in which nodes are composed of pair of tracks, with a weight corresponding to the average similarity of the object detections composing the tracks. An edge connects two nodes if the corresponding 4 tracks (2 per node) share one detection. The MWIS of this new graph is computed to produce longer tracks that are used as basis in the next iteration. This iterative process is stopped when the weight of the MWIS stops increasing.

While the data association problem can be solved efficiently by network flow or iterative hierarchical approaches, those methods have a strong weakness. They can only use simple motion models such as the pairwise distance between consecutive observations. Indeed the cost on the edges can only represent short-term motion (between two consecutive frames). The work presented in *Collins* (2012) addresses this issue by representing the data association problem by a MultiDimensional Assignment (MDA). The MDA is a specialization of the Set Partition Problem (SPP) to the case of k-partite graphs. It consists in partitioning the observations in sets (tracks), picking at most one observation per frame for each set and minimizing the cost of

each track. To solve the MDA, the best assignments are iteratively computed locally by considering pairs of adjacent frames. The Kuhn-Munkres Hungarian algorithm (*Burkard et al.* (2009)) is applied to find such an assignment between the observations in every pair of consecutive frames. The authors state that around 5 iterations of this process are necessary to reach convergence. The strength of this approach is the use of a global cost function for each track $cost(t) = E_{dist} + E_{curv}$. With E_{dist} the average distance between successive pairs of points and E_{curv} a sum of curvature terms over the length of the trajectory. This global cost function allows to take into account all the temporal information to obtain more coherent tracks.

Another recent approach, presented in *Butt and Collins* (2013), also uses a higher-order motion model. Candidate matches between pair of detections in consecutive frames are formed based on their appearance similarity and their spatial proximity. A graph $G = (V, E)$ is built in which each candidate match is represented by 2 nodes, an *incoming node* and an *outgoing* one, connected by an edge weighted by the cost of matching the two detections. The *incoming node* represents the detection appearing in the earliest frame of the candidate match and the *outgoing node* represents the detection appearing in the next frame. Note that with this formulation, multiple nodes represent the same observation (1 node per candidate match involving this detection). *Outgoing* and *incoming nodes* are connected by an edge if they represent the same observation. In this way, 3 observations are connected together: one in frame t, one in frame $t + 1$, which is shared by the two matches, and one in frame $t + 2$. Those edges between the *incoming nodes* and the *incoming nodes* are weighted using a higher-order motion model favoring a constant velocity and direction along the three nodes connected by the linkage of the two candidate matches. Two special nodes are added: a *source node* which is connected to each *incoming node*, and a *sink node*, connected to each *outgoing node*. This formulation is a network flow but is not sufficient. To prevent the selection of nodes representing the same detection, hard constraints are added, but because of this the problem is not a network flow anymore. To remove those constraints and treat the problem as a network flow problem, a Lagrangian relaxation is performed. The hard constraints are turned into soft constraints which are incorporated into the cost function of the network flow problem, and weighted by Lagrangian multipliers. This formulation only provides a lower bound for the original constrained cost function but is convex (*Boyd and Vandenberghe* (2004)). Several iterations of the network flow algorithm are performed, updating the weights of the soft constraints with a subgradient method, until the change in weights becomes too small, or if a maximum number of iterations is reached.

The major drawback of these approaches is the use of specific object detectors

which require to know in advance the type of object to be followed. However no other type of approach can deal with the same amount of interacting targets, making data association methods the most suitable way to analyze scenes with lots of objects.

2.3.6 Arbitrary Object Detection and Tracking

Some approaches try to not make any assumptions about the kind of targets to track. Since in this case, observations usually do not correspond to entire objects but instead to parts of objects, no one to one mapping assumption is made. Because of this, a higher proportion of observations has to be managed to track the same number of targets than standard data association approaches. Without the one to one mapping assumption the number of possible associations between observation increases. Therefore, the computations those methods need to perform are more complex. For those reasons, techniques that try to detect and track arbitrary objects are usually applied to problems involving fewer targets than the common problem of pedestrian detection and tracking in crowded scenes.

In *Yu and Medioni* (2009), no pre-trained detector is used, but instead observations are gathered by background subtraction. This method allows observations to be claimed by several objects, and single object can be represented by several observations in the same frame. The aim is to find the best spatial (inside a given frame) and temporal (across different frames) association between observations. Noisy observations are regrouped in a special track. Each solution, i.e., set of tracks, is rated, favoring small number of long tracks with little overlap with other tracks. A Kalman filtering approach is used to estimate the motion likelihood of each tracks and the appearance likelihood is computed by comparing the histogram descriptors of the observations in the same track in consecutive frames. To find the best solution a Monte Carlo Markov Chain approach is used. This method gives good results on several pedestrian or vehicle videos but the use of background subtraction makes this method inefficient when the camera is moving.

In *Endres and Hoiem* (2010) the authors have presented an approach to extract image segments, called object proposals, that better correspond to foreground objects than regions of traditional segmentations. To do so they learn a category independent object model from the Berkeley Segmentation Dataset (*Martin et al.* (2001)). Their approach produces image segments so that each object of the video is well represented by at least one segment. Beside, each segment is given an "objectness" score reflecting how likely the region is to represent a foreground object. Those scores are computed so that each object has at least one highly ranked corresponding region.

Based on the object proposals produced by *Endres and Hoiem* (2010) the authors

of *Lee et al.* (2011) have designed an algorithm to detect and track the main objects of a video. They first rank each proposal using the following formula : $S(r) = O(r) + M(r)$. $O(r)$ is the score computed by *Endres and Hoiem* (2010). $M(r)$ measures the confidence that the region r corresponds to a coherently moving object and is computed by measuring the difference between the optical flow histograms of r and the pixels $\bar{r}$ around it within a loosely fit bounding box. Then the top ranked segments (with respect to $S(r)$) are selected and the similarity between their unnormalized color histograms is computed, giving high affinity to regions with similar color and size. The pairwise affinity is computed for each pair of regions to obtain an affinity matrix K_c. The next step consists in applying a form of spectral clustering (see for example *Perona and Freeman* (1998)) that produces partitions of regions with high affinities. For each cluster c of regions, Gaussian Mixture Models are estimated to model the foreground pixels' color (pixels in the regions) and the background pixels' color (pixels in the complement of c in all images). The authors then use the background model, foreground model and segments to guide a foreground object segmentation. The experiments presented in *Lee et al.* (2011) only show results for very short videos ($\approx$50 frames) with no occlusion. Besides most results presented show the performances on single target tracking, in the only video with several objects, figuring a dense group of penguin, the whole group is segmented as one object.

A second approach exploiting the object proposals produced by *Endres and Hoiem* (2010) is presented in *Ma and Latecki* (2012). This time, the authors focus on identifying and segmenting the main object of a video in each frames. To do so they build a graph $G = (V, E)$ in which nodes represent the segments. Edges $(u, u) \in E$ represent the "objectness" score of region u using the same formula as *Lee et al.* (2011). Edges $(u, v) \in E$ represent the similarity between regions u and v. They also introduce constraints specifying that regions in the same frame cannot be part of the same solution as well as regions that are too far apart spatially. Once the graph is built, the next step consists in finding the Maximum Weight Clique (MWC). This MCW corresponds to a sequence of object proposals (1 per frame) that are likely to represent the same object. Finally, as in *Lee et al.* (2011), the appearance model of the foreground and background is estimated and used in combination with the segments to guide a foreground segmentation of each images. The algorithm successfully identifies the main object in the videos and the results obtained are a bit better than the ones of *Lee et al.* (2011) in term of precision. Again no occlusion test has been performed for this approach.

Zhang et al. (2013) also uses the object proposals of *Endres and Hoiem* (2010) and tries to detect and track the main object of a video. First they use the optical flow to

warp object proposals in frame t to the frame $t+1$ and merge them with the image segments of frame $t+1$ if they have at least 50% overlap. Those merges are added to the original pool of object proposals. This process is carried out both forward and backward in time. Then a graph is built using two nodes to represent each proposal, an *incoming node* and an *outgoing one*. Edges between those nodes are weighted according to the "objectness" score of *Endres and Hoiem* (2010). *Outgoing nodes* are connected to the *incoming node* of the next three frames with an edges weighted according to appearance and shape similarity scores. Next, a track is returned by looking for the highest weight path in the graph. Using the image segments of the proposals linked by this track, two Gaussian mixture models are built, one modeling the object's appearance and one the background. Those Gaussian mixture models are then used to guide a segmentation of the object represented by the track in each frame.

While those three last state-of-the art approaches are interesting, they still are very dependant on the quality of the results of detector of *Endres and Hoiem* (2010), which does not perform equally on different object classes (see Figure 2.12). They also don't have any mechanism to deal with occlusions, and have difficulties distinguishing between objects in crowded scenes, which makes them unlikely to be able to follow an object in a complex video.

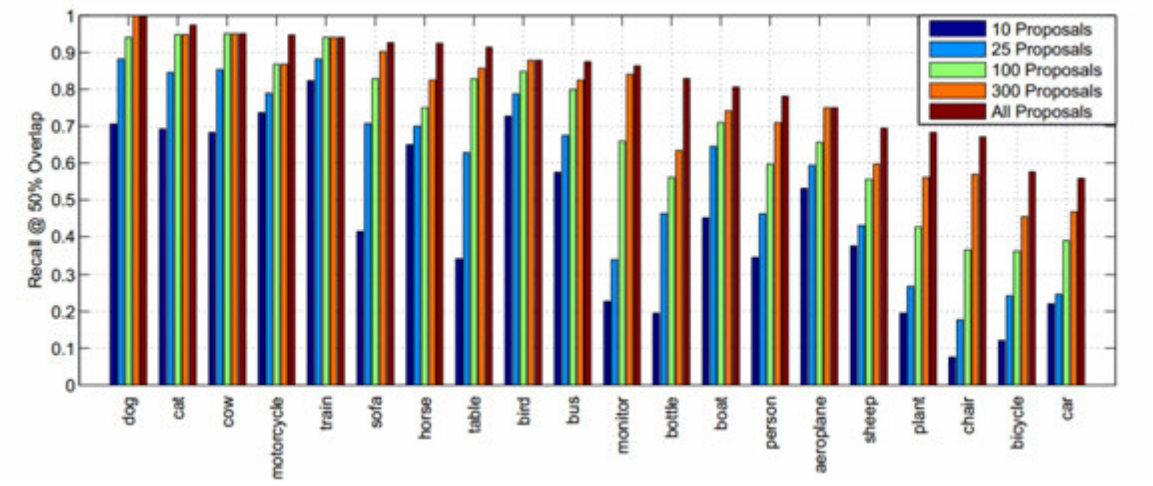

FIGURE 2.12: Recall results of object proposals obtained with *Endres and Hoiem* (2010) for each object categories in PASCAL VOC 2008 (*Everingham et al.* (2008)). The number of proposals corresponds to the number of highest ranked segments sectioned

Finally, *Fragkiadaki and Shi* (2011) presents a motion segmentation approach that tries to achieve detection free tracking. To do so, they first extract dense point trajectories using *Sundaram et al.* (2010). This dense optical flow is used to segment the image into foreground regions with different motion. Using tracks instead of pixels

to conduct the segmentation allows to segment pixels even if they stop moving for some time. Then, a graph is built in which nodes represent tracks and edges represent attractive and repulsive forces between them. Tracks with similar motions are attracted while tracks that belong to different segments of the motion segmentation in at least one frame repulse themselves. Following the solution proposed by *Yu and Shi* (2001), a form of spectral clustering is performed on the tracks. At first, only long tracks are clustered, then smaller tracks are added to the clusters depending on their affinity with them. Experiments have been conducted on videos of 50 frames. They show that multiple basket ball players can be detected and tracked with this approach. Overall 30% of the pixels representing the players are recalled and in 75% of the video frames the recall of the players is above 20%.

2.4 Datasets

There exists a lot of datasets used by tracking approaches to assess the efficiency of their algorithm. Most of datasets are specialised in one type of setup (e.g., moving/non moving camera, sport event, video surveillance ...).

Video surveillance is the domain for which there is the most choice, a lot of video datasets involve pedestrians walking in the field of view of a fixed camera. The CAVIAR[6] (*Caviar* (2004), see Figure 2.13), PETS 2009[7] (*Ferryman et al.* (2009), see Figure 2.14) and TUD (*Andriluka et al.* (2008), see Figure 2.15) datasets are among the most commonly used in fixed camera setups. The Caltech[8] (*Dollar et al.* (2012), see Figure 2.20), filmed from a car in traffic, and ETHMS[9] datasets (*Ess et al.* (2007), see Figure 2.19), taken from a moving stroller, are designed for video surveillance applications under moving camera.

Arbitrary object tracking approaches often reference the Gatech SegTrack[10][11] (*Tsai et al.* (2010),*Li et al.* (2013), see Figure 2.18). Those two datasets are composed of short video segments (usually < 100 frames) in which there is usually a single object undergoing strong appearance changes.

Single target tracking approaches often perform their experiments on the dataset[12] provided by *Kalal et al.* (2012) (see Figure 2.17). This dataset contains a comprehensive range of videos. Their length is comprised between 150 and 10000 frames

[6]CAVIAR: http://homepages.inf.ed.ac.uk/rbf/CAVIAR/
[7]PETS 2009: http://www.cvg.rdg.ac.uk/PETS2009/a.html
[8]Caltech: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/
[9]ETHMS: http://www.vision.ee.ethz.ch/~aess/dataset/
[10]SegTrack: http://cpl.cc.gatech.edu/projects/SegTrack/
[11]SegTrack v2: http://www.cc.gatech.edu/~fli/SegTrack2/dataset.html
[12]TLD: http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/index.html

with various objects (cars, pedestrians, animals etc ...). The quality of those videos is quite low, which is used to assess the robustness of the algorithms. This can be problematic for approaches that require a minimum of details to properly describe the targets. Another dataset widely used by single target tracking approaches is the Babenko[13] dataset composed of 12 videos(*Babenko et al.* (2011), see Figure 2.16).

As we can see there is no common dataset for all tracking methods, most of them are composed of at most a dozen of videos and are designed to assess the efficiency of a particular type of approach. Moreover, it is common practice to compose new datasets by picking videos from other datasets. This can make it hard to compare the different approaches.

FIGURE 2.13: Sample frames from the CAVIAR dataset (*Caviar* (2004)) with ground truth for pedestrians with body parts (in yellow), groups of people (green boxes), and ground plane detection in the last image on the right.

FIGURE 2.14: Pets 2009 dataset (*Ferryman et al.* (2009)). *Top images (from left to right):* multiple cameras setup. Ground truth for people count and density estimation in different regions of the field of view. Ground truth for the number of people entering through the brown line and number of people exiting through the red and purple lines. *Bottom images* show the ground truth for tracking pedestrians tagged with the marks A and B.

[13]Babenko: http://vision.ucsd.edu/~bbabenko/project_miltrack.shtml

FIGURE 2.15: Sample frames from the TUD dataset (*Andriluka et al.* (2008)). *Left:* Ground truth for pedestrian tracking with temporal their temporal identification. *Middle:* Ground truth for articulated body pose estimation. *Right:* 8 viewpoint annotations.

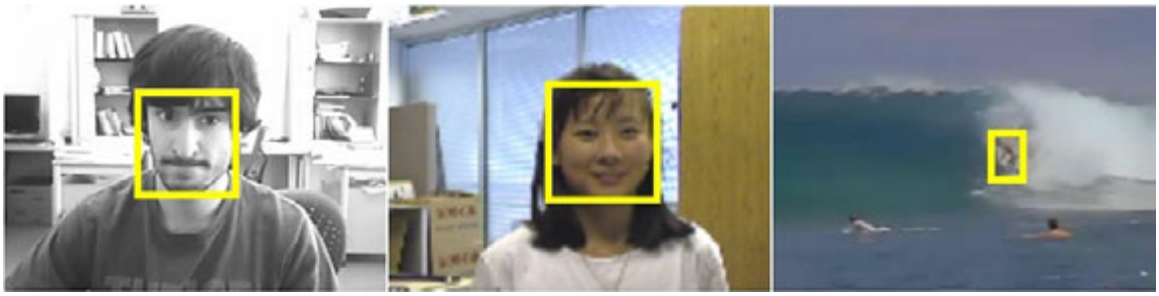

FIGURE 2.16: Sample frames from some of the 12 videos of the Babenko dataset *Babenko et al.* (2011)

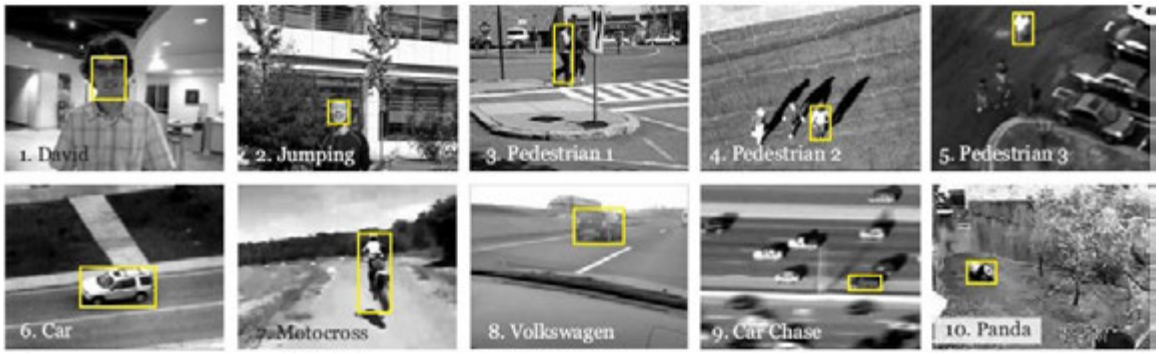

FIGURE 2.17: Sample frames of each sequence in the dataset used by TLD (*Kalal et al.* (2012))

FIGURE 2.18: Segtrack v2 dataset (*Li et al.* (2013)) with precise ground truth for the main objects

FIGURE 2.19: Example sequence of the ETHMS dataset (*Ess et al.* (2007)), taken from a moving stroller, with ground truth for pedestrians as well as their temporal identification

FIGURE 2.20: Frames from the Caltech dataset (*Dollar et al.* (2012)), taken from a car driving in traffic, with ground truth for pedestrians with occlusion labels (in dashed yellow)

2.5 Conclusion

Visual tracking is a very complex problem that is usually reduced to a simpler one by considering specific tracking cases. The ideal goal would be to automatically find the objects that should be tracked in the videos, but, like for clustering and image segmentation, defining exactly what are the expected results is difficult.

If someone wants to be able to track any visual feature, from small details to entire objects, tracking by detections achieves very good real time results under a variety of motions, dealing with occlusions and appearance changes. Approaches in *Kalal et al.* (2012), *Zhang et al.* (2012) or *Supancic III and Ramanan* (2013) are good example of state-of-the-art trackers in this category.

Segmentation-based approaches can track targets with high precision but they also require a precise initial segmentation of the targets. Tracking the target with high precision requires complex computations that cannot be applied on the entire frame in reasonable time. This reduces the capacity of those approaches to find the target if its motion cannot be predicted accurately. Hence segmentation based trackers usually have problems recovering from long term complete occlusions. Under those conditions *Papadakis et al.* (2011) provides excellent results.

A common limitation of most single target trackers is that they tend to perform poorly when multiple similar targets are present. Approaches exploiting the context

such as *Gu and Tomasi* (2011) or *Dinh et al.* (2011) can overcome this limitation by tracking *distracters*, that should not be confused with the target, and by using *supporters* to recover the target when it disappears.

When there are a lot of interacting targets to track, data association approaches are better suited. They also do not necessarily require the number of objects and their position in the first frame to be specified. Instead they restrict themselves to tracking particular sets of objects or to specific scenes with a non moving camera. This allows them to ignore the background and focus on interesting observations such as those returned by a specific object detector. They directly try to make sense of the scene by modelling the interactions between objects. If the type of target can be known in advance, approaches like *Collins* (2012) and *Butt and Collins* (2013) give very good results by exploiting a maximum of the temporal information.

Without any knowledge on the objects to track, *Lee et al.* (2011), *Ma and Latecki* (2012) and *Zhang et al.* (2013) are good choices in setups with few targets. But those approaches have only been tested on the short video segments of the SegTrack dataset (*Tsai et al.* (2010)), it is therefore unclear how they would perform in long term tracking conditions. Moreover, those approaches have difficulties distinguishing the individual objects in a crowded scene.

To conclude, there is still no perfect approach requiring no human input to detect and track interesting objects in the general case. This is due to the fact that interestingness is a subjective measure. Therefore all approaches have to reduce the number of scenarios they intend to deal with, either by asking a human to choose what is interesting or by making assumptions about the type of objects that are interesting. In this thesis we address the problem of arbitrary object detection. More precisely we investigate how the interesting objects in a video could be detected and tracked without requiring any user intervention or supervised information to train a classifier. The main assumption we make is that the main objects of a video should appear more frequently than the background if the camera is moving. Therefore, visual features appearing frequently in those videos should be useful to achieve arbitrary object detection. The next chapter will discuss the topic of Frequent Pattern Mining which is the field of research that focuses on extracting frequent patterns from the data.

CHAPTER 3

Graph Mining

3.1 Introduction

Assuming that the background of a video is changing, for example if the camera is moving, the main objects should appear more frequently than the elements of the rest of the scene. However, when the camera or an object moves, or if the illumination of the scene changes, the visual features composing it are also affected. Therefore, visual features of an object might not be consistent enough through time to constitute a robust representation. Nevertheless and regardless of its orientation or its scale, the topology of the different visual parts of an object, i.e, the topological relationships between parts of the same object, should be more consistent than the ones between parts of different objects. Based on this observation, the main hypothesis of this thesis is that if an object frequently appears in the frames of a video, the visual parts composing it should appear frequently and the topological relationships between those parts should be frequent as well. By representing the frames of a video with graphs, the problem of detecting and tracking the main objects of the video can be related to the problem of discovering *frequent subgraphs* in the database of graphs representing the video frames. Such an approach presents the advantage of not requiring any user intervention to determine what is interesting in a video. Instead, it relies on the frequency of substructures in the frames representation. Moreover, it is not restricted to a particular type of object such as pedestrians or cars for example.

This chapter gives an overview of *frequent graph mining*. The next section deals with the more general problem of *pattern mining*. Then we provide important definitions related to the field of *graph mining*. In section 3.4 we discuss in more depth the different problems faced by *graph mining* algorithms and how they have been tackled in the literature. Section 3.5 presents a review of popular *graph mining* methods. Finally, Section 3.6 briefly discusses how to represent videos with graphs before con-

cluding.

3.2 Generalities on Frequent Pattern Mining

Pattern mining is a subfield of data mining that focuses on extracting interesting patterns and relationships from large datasets or databases. Frequent pattern mining has been popularized by the Apriori algorithm (*Agrawal et al.* (1993)) in the context of the *market-basket analysis*. Given a set of elements called *items* $\mathcal{I}$ (e.g., eggs, milk, chocolate ...) and a database $\mathcal{D}$, containing *transactions*, i.e. sets of items (e.g., the set of all the goods that a given customer bought in one transaction), a frequent *itemset* mining algorithm such as Apriori mines sets of items that frequently appear together in the transactions of the database.

As pointed out in *Frawley et al.* (1992), given a reasonably large dataset, a very large number of potentially interesting patterns could be considered. In fact this number grows exponentially with the size of the database. It quickly became clear that measures of *interestingness* had to be developed to reduce the number of output patterns. Two types of measures can be distinguished, *objective measures*, that depend only on the structure of the pattern and the data, and *subjective measures*, that can take into account user preferences to return more specific types of patterns. Popular *objective measures* are the *support* of an *itemset* and *confidence* of an *association rule* (*Agrawal et al.* (1993)). The *support* of an *itemset* corresponds to the number of transactions in the database that contain it. The *confidence* of an *association rule* "$A \Rightarrow B$" (where A and B are itemsets) is the number of transactions containing both A and B divided by the number of transactions containing A. More explicitly, the confidence of "$A \Rightarrow B$" corresponds to the proportion of customers who bought B among the ones who bought A. *Piatetsky-Shapiro and Matheus* (1994) noted that, objective measures still output a lot of uninteresting patterns. In *Silberschatz and Tuzhilin* (1996), the authors studied a subjective measure called *actionability*, which considers a pattern interesting if the user can act on it. For example, consider a database of student evaluations of courses at the university. Evaluations can take the form of a vector $\mathbf{e} = (term, year, course, section, instructor, instructor\_rating, course\_rating)$. Knowing that a specific professor always gets an instructor ratings below its course rating ($instructor\_rating < course\_rating$) can be interesting for a chairperson of the department in which the professor is teaching because this person can act on it by advising the professor to improve his teaching. The authors of *Piatetsky-Shapiro and Matheus* (1994) also study another subjective measure called *unexpectedness*, which considers a pattern interesting if it is *unexpected* to the user with respect to his belief.

A survey on measures of *interestingness* can be found in *Geng and Hamilton* (2006).

In this thesis, we focus on the *frequency* interestingness measure. It is defined as the *support* of a pattern divided by the total number of transactions in the database. In this context, a pattern is considered *interesting* if its *frequency* is higher than a *minimum frequency constraint*. A large number of frequent pattern mining techniques are described in the literature. They depend on the type of the data, its representation, or the relationships between elements of a database. Most algorithms can be roughly summarized as follows:

- Given a frequent pattern p, find all its occurrences in the database.
- Extend patterns (e.g., by adding one item to them).
- Scan the database to count the number of occurrences of each extension. Repeat the process with the extended patterns.

Starting from frequent simple patterns of one *item*, this process iteratively builds all frequent patterns in the original database. A pattern p extended in a bigger pattern p' is called an extensions of p. A *sub-pattern* of p' is a pattern p that is included in p'. Conversely, p' is a *super-pattern* of p. To keep this problem tractable, the search space of possible candidate patterns has to be bounded in some way. An important property of the *minimum frequency constraint* is that it is anti-monotonic. This means that if a pattern is *infrequent*, none of its *super-patterns* can be *frequent*. Therefore, *infrequent patterns* do not need to be extended, i.e., *infrequent patterns* can be safely pruned from the search space without missing any *frequent pattern*. Often, not all *frequent patterns* are interesting and redundancies in the discovered knowledge can be reduced by focusing on particular classes of patterns, such as *closed* or *maximal patterns*. A pattern is *closed* if all its *super-patterns* have a frequency strictly lower than his, and a pattern is *maximal* if none of its *super-patterns* is frequent. In *Calders and Goethals* (2007), the authors show how to derive lower and upper bounds for the support of *itemsets*, based on the support of all subsets of an *itemset*. Those bounds can be used to avoid processing patterns for which their support can be derived from their *sub-patterns* and focus on *Non-Derivable Itemsets*. This allows to build a condensed representation that covers all the *frequent itemsets* to reduce the output of the mining algorithm while still discovering the important information.

Before being capable of extracting meaningful patterns, it is crucial to choose a data representation scheme that fits the problem at hand. *Itemsets* are one type of pattern. If the order among *items* of the *transactions* has an importance, patterns will take the form *subsequences*. Graphs are used when the relationships between

the items has to be modeled, in this case the mining algorithms will try to discover *subgraphs*. In the context of this thesis, we choose to represent each frame of a video with graph in order to take into account the topology of the images. Therefore a video can be seen as a database of graphs (one per frame) which can be mined to extract interesting subgraphs that should correspond to parts of the main objects. A more detailed study of Pattern Mining can be found in *Han and Kamber* (2006), the rest of this chapter will focus on the more specific problem of Graph Mining.

When mining graphs, two different cases can be distinguished depending on the type of database mined. Indeed it is possible to either mine a large single graph, or in a database composed of several graphs. Depending on the setup, the frequency of the patterns will have to be computed differently.

3.3 Definitions and Notations

3.3.1 Graphs

Graphs are powerful mathematical tools that are used to model relationships among a set of elements. Simple graphs are defined as a pair of elements $G =< V, E >$ where $V = \{v_1, ..., v_n\}$ is a set of nodes and $E \subseteq V \times V$ a set of edges connecting them. If there is an edge between each pair of nodes, the graph is said to be *complete*. Often, labels are given to nodes and edges to model more precisely the features of the elements and their relationships. Those graphs are called *labeled* or *attributed graphs*.

Definition 3.1 (Labeled Graph)**.** A labeled graph is a graph $G =< V, E, L >$ where $V = \{v_1, ..., v_n\}$ is a set of nodes, $E \subseteq V \times V$ a set of edges connecting pairs of nodes (v_i, v_j). The labeling function $L : V \cup E \to \mathbb{N}$ maps each edge and each node of the graph to a label.

Note that, in the case of directed graphs, each edge is an ordered pair of nodes (v_i, v_j), while in the case of undirected graphs the pairs of nodes are unordered. Graph mining algorithms look for *subgraphs* that appear frequently in the database.

Definition 3.2 (Subgraph)**.** Given two graphs $G =< V, E >$ and $G' =< V', E' >$, G' is a *subgraph* of G if and only if $V' \subseteq V$ and $E' \subseteq E$.

Definition 3.3 (Induced Subgraph)**.** Given two graphs $G =< V, E >$ and $G' =< V', E' >$, G' is an *induced subgraph* of G if and only if $V' \subseteq V$ and $E' = E \cap V' \times V'$.

If a subgraph G' of another graph G is *complete*, G' is called a *clique* of G.

A *path* in a graph $G =< V, E >$ is a sequence such that consecutive nodes are connected by edges. A *finite path* is a path with a finite number of nodes. Its first

node is called the *starting node* and its last one the *ending node*. If the *starting node* and the *ending node* are the same the path is called a *cycle*. A *path* is said to be *simple* if it never passes twice through the same node.

Most of the time graph mining algorithms deal with *connected graphs*, i.e., graphs in which there exist at least one *path* connecting each pair of nodes.

Trees are also particular graphs that are the focus of many graph mining algorithms. A tree is a *connected* graph with no *cycle*. A *rooted tree*, is a *tree* for which one node is singled out as the *root*. If no *root* is designated, the *tree* is called a *free tree*.

Finally, in this thesis, the graphs used to represent images are *plane graphs*, meaning that they are drawn in the plane without any of their edges crossing. Note that a planar graph is a graph that can be drawn in the plane without any of its edges crossing, while a plane graph is a planar embedding of a planar graph. Each *plane graph* is composed of a set of faces.

Definition 3.4 (Face)**.** Given a plane graph, a *face* is a connected region of the plane which is bounded by a cycle 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.

Definition 3.5 (Plane graph)**.** A plane graph is a tuple $G = (V, E, F, f_e, L)$ where V is a set of nodes, E is a set of edges, F is a set of *faces* and L is a labeling function on $V \cup E$. The unbounded region f_e in the embedding of the graph is called the *outer face* of the graph. The other faces are called *internal faces*.

For example, Figure 3.1 presents three plane graphs and the graph g_1 has two internal faces $\langle 1, 2, 3\rangle$ and $\langle 2, 4, 5, 3\rangle$, and its outer face is $\langle 1, 3, 5, 4, 2\rangle$.

Definition 3.6 (k-connectedness)**.** A plane graph is k-connected if k is the size of the smallest subset of vertices such that the graph becomes disconnected if you delete them.

The k-connectedness can also be defined using Menger's theorem *Menger* (1927).

Theorem 3.7 (Menger's theorem)**.** *A graph G is k-connected if and only if every pair of vertices is connected by k internally disjoint paths.*

Note that, using Menger's theorem, we can see that in a 2-connected graph each pair of distinct nodes is connected by at least 2 internally disjoint paths forming a simple cycle (no node or edge is used more than once). Therefore a plane graph is 2-connected if each face (and in particular the outer face) is a simple cycle.

3.3.2 Isomorphism and Subgraph Isomorphism

Graph isomorphisms are used to assess if two graphs are equivalent, i.e., if we can find a mapping between the nodes that preserves the edges and the labels.

Definition 3.8 (Graph Isomorphism)**.** Two graphs $G = (V, E, L)$ and $G' = (V', E', L')$ are said isomorphic if and only if there exists a *bijective* function $f : V \to V'$ such that

- $\forall v \in V, L(v) = L'(f(v))$,
- $\forall (v_1, v_2) \in V \times V, (v_1, v_2) \in E \Leftrightarrow (f(v_1), f(v_2)) \in E'$
- $\forall (v_1, v_2) \in E, L(v_1, v_2) = L'(f(v_1), f(v_2))$.

Definition 3.9 (Subgraph Isomorphism)**.** A graph G_1 is said to be subgraph isomorphic to a graph G_2, noted $G_1 \subseteq G_2$, if there exists a subgraph G_2' of G_2 such that G_1 is isomorphic to G_2'.

A subgraph isomorphism of a pattern P in a graph G is called an *occurrence* of P in G.

Those definitions can be extended to the specific case of plane graphs. As for general graphs, a plane graph is a plane subgraph of another plane graph if there exists a correspondence between their nodes which preserves the labels and the edges, except that the correspondence between nodes should also preserve the internal faces (if the outer face is also preserved, then the graphs are plane isomorphic).

Definition 3.10 (Plane subgraph isomorphism)**.** Let $G = (V, E, F, f_e, L)$ and $G' = (V', E', F', f_e', L)$ 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 $f : V \to V'$ such that:

- $\forall v \in V, L(v) = L'(f(v))$,
- $\forall (v_1, v_2) \in E, L(v_1, v_2) = L'(f(v_1), f(v_2))$,
- $\forall$ internal faces $F = \langle v_1, ..., v_k \rangle$ of G, $f(F) = \langle f(v_1), ..., f(v_k) \rangle \in F'$.

As we can see in Figure 3.1, the internal faces $\langle 2, 4, 5 \rangle$, $\langle 2, 5, 7 \rangle$ and $\langle 4, 5, 7 \rangle$ of G are not present in g_2, therefore it is not plane subgraph isomorphic to G.

From this an occurrence of a plane graph in a larger graph is defined as follows:

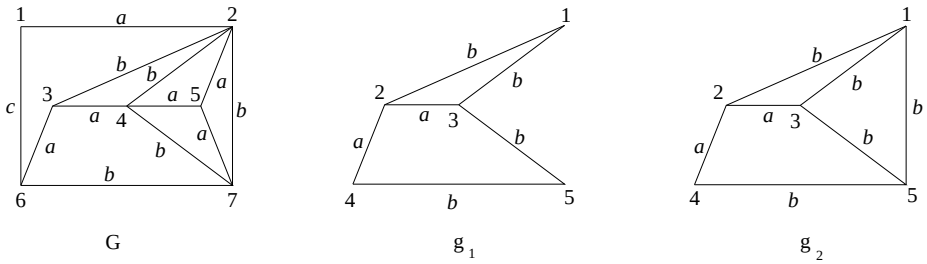

FIGURE 3.1: Plane graphs. The edge labels are in $\{a, b, c\}$ and we assume that all node labels are equal to a (not represented). Graph g_1 is a plane subgraph of G while g_2 is not.

Definition 3.11 (Occurrence of a plane graph in a larger graph)**.** Let two plane graphs G and G'. If G' is plane subgraph isomorphic to G, the corresponding injective function f is called an *o*ccurrence of G' in G.

Example 3.12. In Figure 3.1, 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.

3.3.3 Support and Frequency of a Subgraph Pattern

The *support* of a *subgraph pattern* P in a database $\mathcal{D}$ corresponds to the number of graphs $G_i \in \mathcal{D}$ to which it is *subgraph isomorphic*:

$$support_{\mathcal{D}}(P) = |\{G_i | P \subseteq G_i \text{ and } G_i \in \mathcal{D}\}|.$$

The *frequency* of a pattern P is the ratio of its *support* divided by the size of the database:

$$frequency_{\mathcal{D}}(P) = \frac{support_{\mathcal{D}}(P)}{|\mathcal{D}|}.$$

Note that, while counting the *support* of a *subgraph pattern* in this fashion is appropriate when mining a database of multiple graphs, it has to be defined differently in the case where a single graph is mined (*Bringmann and Nijssen* (2008)). Indeed, in this case if the support is counted in number of non identical occurrences, it is not anti-monotonic. For example, in Figure 3.2, we can see that there is only one occurrence of pattern p_1 in g while p_2 has 8 different occurrences. Therefore, in this scenario, the support is usually counted in terms of the number of non overlapping occurrences. This can be computed by first building the overlap graph of the occurrences in which nodes represent the occurrences of a pattern and an edge connects

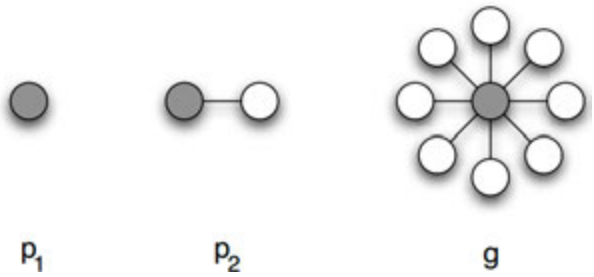

FIGURE 3.2: Patterns with non-monotonic support. Although p_1 is a sub-pattern of p_2, its support in g is inferior to the support of p_2 equating 8 (*Bringmann and Nijssen* (2008))

two nodes if the corresponding occurrences overlap. Then the support of a pattern P corresponds to the size of the Maximum Independent Set (MIS) of the overlap graph of the occurrences of P, which is anti-monotonic (*Gudes et al.* (2006)). Since the MIS problem is known to be NP-hard (*Michael and Johnson* (1979)), some algorithms such as SIGRAM (*Kuramochi and Karypis* (2006)) use approximations of the MIS. Nonetheless, since in the context of this thesis we mine databases of graphs, we will use the above definition.

Similarly to *itemsets*, not all *frequent subgraphs* are always interesting. Therefore some algorithms focus on mining particular types of graph patterns (*Yan and Han* (2006)). For example, some approaches mine *approximate subgraphs* (e.g. *Holder et al.* (1994)), this allows to represent several subgraphs with minor variations with a single *subgraph pattern*. Other methods look for *discriminative subgraphs* (e.g. *Borgelt and Berthold* (2002)), which are *subgraph patterns* that are frequent in some dataset but not in another one. They have been used in the context of graph indexing, in *Yan et al.* (2004) for example, to reduce the number of subgraphs pattern used as index for indexing graphs. *Coherent subgraph* (*Huan et al.* (2003b)) are graphs for which the support of their individual edges is correlated. Another type of interesting patterns are *dense subgraphs*, which are graphs with high connectivity. They are used for the analysis of massive networks, such as finding communities in social network graphs, for instance in *Newman* (2004). Another common way to reduce the output patterns is to only focus on *closed frequent subgraphs* (*Yan and Han* (2003)) or *maximal frequent subgraphs* (*Huan et al.* (2004a)).

3.4 The Different Components of Graph Mining Algorithms

Three main problems have to be addressed by graph mining algorithms. The first one is of counting the number of occurrences of each pattern, which requires matching patterns to subgraphs in the database. The second problem is the one of

efficiently generating candidate subgraph patterns. Finally, a mechanism is needed to avoid processing multiple times the same candidate subgraph pattern. The rest of this section will focus on those three problems. A particular attention will be given to how they are solved by the gSpan algorithm (*Yan and Han* (2002)) since we used it for comparison to our contributions (chapter 4).

3.4.1 Graph Matching

To count the number of occurrences of each pattern, it is necessary to match them in the database by *subgraph isomorphisms*, which is a well known NP-Complete problem (*Michael and Johnson* (1979)). Therefore, mining algorithms usually try to avoid computing too many *subgraph isomorphisms*. Graph matching algorithms can be classified into two types: *exact matching* and *inexact matching* algorithms.

Exact matching algorithms are simpler than *inexact matching* ones. The Ullman algorithm (*Ullmann* (1976)) is one of the first and still very popular *exact matching* algorithms. This algorithm builds a search tree of possible mappings between the graphs. It extends a partial matching iteratively by adding to it new node-to-node correspondences. The partial matching is expanded until no further correspondence can be added without violating the edge structure or the consistence of the labels on the nodes and edges. If the search space is fully explored without finding any valid match, the two graphs are not isomorphic to each other. VF and VF2 (*Cordella et al.* (2004)) are more recent algorithms using this search tree approach. More recently, *Solnon* (2010) proposed an algorithm called LAD[1]. This algorithm formulates the matching problem as a Constraint Satisfaction Problem (CSP). The author devised a new conditional constraint expressing the fact that *subgraph isomorphism* should preserve edges and that two different nodes cannot be matched to a same unique node. Another recent algorithm, called RI was presented in *Bonnici et al.* (2013). This algorithm matches in priority nodes with high degree which are the ones that can put the most constraint on the rest of the matching. According to a recent experimental study on biological databases (*Carletti et al.* (2013)), RI is currently the best *exact isomorphism* algorithm, although the study showed that LAD performed better on dense graphs.

There exist efficient polynomial algorithms for some subclasses of graphs such as *paths* (*Babel et al.* (1996)), *trees* (*Buss* (1997)), or *plane graphs* (*Damiand et al.* (2009); *De La Higuera et al.* (2013)). This low complexity led to the development of algorithms specialized in the mining of paths (*Kramer et al.* (2001)) free trees

[1]LAD algorithm source code available here: `http://liris.cnrs.fr/csolnon/LAD.html`

(*Rückert and Stefan* (2004)) and combinatorial maps (*Gosselin et al.* (2011)). For instance, in *Horváth et al.* (2010), the authors focus on finding frequent subgraphs in *outer-planar graphs*, which are graphs that can be represented in the plane in such a manner that all their nodes lie on the outer boundary, i.e., it is possible to add a new node to the graph and connecting it to all other nodes without crossing any edge.

Sometimes one is only interested in finding how similar two graphs are. It can be achieved through *exact matching* by finding the Maximum Common Subgraph (MCS). The MCS is a common subgraph between two graphs G_1 and G_2 such that there exists no bigger common subgraph between them. A distance measure can be derived from the MCS in the following way:

$$dist(G_1, G_2) = \frac{|mcs(G_1, G_2)|}{max(|G_1|, |G_2|)},$$

where $|G_1|$, $|G_2|$ and $|mcs(G_1, G_2)|$ are respectively the size of G_1, G_2 , $mcs(G_1, G_2)$ in terms of number of nodes. A comparison of MCS algorithms can be found in *Bunke et al.* (2002).

Inexact matching algorithms, try to deal with small differences to find graphs that are similar. An intuitive approach consists in computing a *graph edit distance* (*Bunke and Allermann* (1983)). Given an *edit cost matrix* that reflects the cost of deleting a node or an edge, substituting two nodes or inserting a node or an edge, the aim is to find the minimum cost sequence of edit operations that turns a graph into another. Note that in order to accurately represent the similarity between graphs, *graph edit distances* require a properly tuned *edit cost matrix*, which is usually achieved by mean of a learning procedure.

Another family of *inexact matching* algorithms uses relaxation labeling (*Fischler and Elschlager* (1973)). The aim of those approaches is to assign to each node of one of the graph labels representing the nodes of the other graphs. Each node of the first graph possesses a vector of probabilities of being assigned each label. An initial labeling of the nodes is computed based on their attributes, connectivity or other informations. The probability vectors of each node is iteratively refined until the values stop changing or a maximum amount of iterations is reached.

Inexact matching has also been achieved using *spectral methods* (*Umeyama* (1988)). Those approaches are based on the fact that eigenvalues and eigenvectors of the adjacency matrix of a graph are invariant with respect to node permutation. This means that the adjacency matrices of two isomorphic graphs will have the same Eigen decomposition. The converse is not true. The Eigen decomposition is used to efficiently find a permutation matrix P, which represents the mapping of nodes of a graph G_1

to the nodes of another graph G_2, minimizing the following cost of the matching:

$$cost(P) = ||PA_{G_1}P^T - A_{G_2}||^2,$$

where A_{G_1} and A_{G_2} are the adjacency matrices of graphs G_1 and G_2. The Eigen decomposition of the adjacency matrices is given by $A_{G_1} = U_{G_1}\Lambda_{G_1}U_{G_1}^T$ and $A_{G_2} = U_{G_2}\Lambda_{G_2}U_{G_2}^T$ where Λ_{G_1} and Λ_{G_2} contain the eigenvalues of A_{G_1} and A_{G_2}. P can be obtained by binarizing the matrix $\bar{U}_{G_1}\bar{U}_{G_2}^T$ where $\bar{U}_{G_1}$ is obtained by taking the absolute value of each element of U_{G_1}

Many other *inexact matching* methods have been developed such as *Artificial neural networks* using Hopefield networks (*Sperduti and Starita* (1997)) or *graph kernels* (*Neuhaus and Bunke* (2007)). An extensive survey of both *exact* and *inexact matching methods* is presented in *Conte et al.* (2004). Note that, *inexact matching* approaches have difficulties dealing with large graphs and therefore are usually too computationally expensive to be applied in a graph mining context.

3.4.2 Canonical Representations

Testing if a particular subgraph has already been discovered requires doing costly isomorphism tests. To avoid such cost, graph mining algorithms have been using *canonical codes* to represent graphs by ordered sequences. For instance, in *Inokuchi et al.* (2000), the AGM algorithm represents a graph by concatenating the elements in the upper triangle part of its adjacency matrix into a sequence. For example, suppose that a graph with k nodes has the following adjacency matrix:

$$X_k = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,k} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & x_{k,k} \end{pmatrix}$$

The corresponding code of such a graph is:

$$code(X_k) = x_{1,1}x_{1,2}x_{2,2}x_{1,3}x_{2,3}x_{3,3}\cdots x_{k-1,k}x_{k,k}.$$

Several different codes can be obtained for the same graph by permuting the columns of the adjacency matrix (and the rows accordingly). For this reason, before building a code, the indexes of the matrix are sorted based on the labels of the nodes. An order is defined on those codes and the minimum one is defined as the *canonical code* of the graph. Canonical codes have the property of being equal for

isomorphic graphs. In *Inokuchi et al.* (2000), this property is used to check if two candidate subgraphs are isomorphic in order to not count duplicates when computing the support of each pattern.

One of the most popular algorithms using a *canonical representation* is gSpan (*Yan and Han* (2002)). Here the code of each graph is built by doing a Depth-First Search (DFS) traversal of its nodes. Figure 3.3 shows an example of graph (a) and 3 DFS trees (b),(c) and (d) obtained by starting the DFS traversal at different nodes of (a). This exploration of the nodes of a graph defines an order on them based on their time of discovery, i.e., nodes of a graph are sub-scripted based on their order of discovery. The authors then define an order on the edges. Each edge is described by a 5-tuple $(i, j, l_i, l_{(i,j)}, l_j)$, with i, j the subscripts of the nodes, l_i, l_j their labels and $l_{i,j}$ the label of the edge (i, j). This order is used to build the *DFS code* of the DFS trees. Codes of the 3 DFS trees of Figure 3.3 are shown in table 3.1.

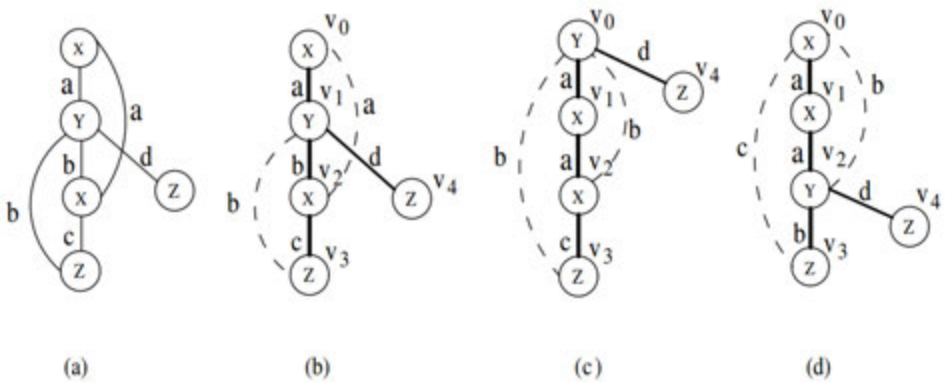

FIGURE 3.3: Depth-First Search Tree (*Yan and Han* (2002)). In (b)-(c) thick black edges are the DFS tree edges, called *forward edges*, dashed ones are *backward edges*

edge	(b)	(c)	(d)
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,0,Z,b,Y)	(3,1,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)

TABLE 3.1: DFS codes for the DFS trees of Figure 3.3, (*Yan and Han* (2002))

Based on the order on the edges, an order on the codes is built such that among all possible *DFS codes* of a graph, the minimum one is its *canonical code*. As for

AGM, this *canonical code* can be used to check whether a subgraph has already been discovered and therefore can be safely pruned from the search space without missing any important pattern.

3.4.3 Candidate Generation

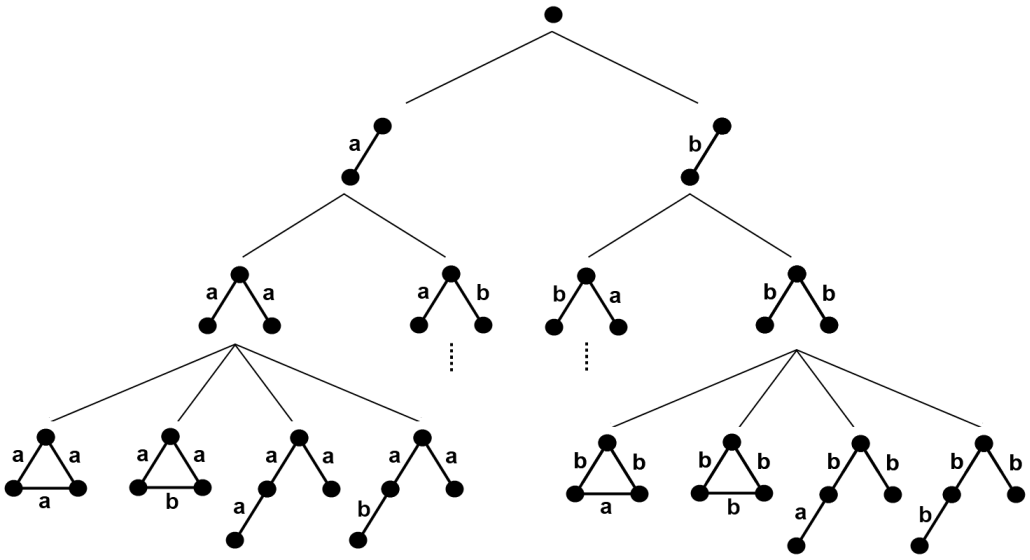

FIGURE 3.4: Partial search space of all graphs with two different labels, **a** and **b**, only on the edges.

The search space of the mining algorithms is a tree in which each pattern p of size k is connected to its *father* pattern of size $k-1$, from which it has been grown, and to its children of size $k+1$. For instance, Figure 3.4, represents a part of the search space of all possible graphs with two labels, **a** and **b**, only on the edges. At each depth level of the tree, patterns of size n are extended into patterns of size $n+1$ by adding-one edge. Such a search space grows exponentially in the number of edges of the patterns, thus, it is quickly impractical to explore. The main way to bound it is to exploit the *anti-monotonicity property* of the *minimum frequency constraint*, which ensures that the growth of infrequent patterns can be stopped without missing any frequent ones.

There exist two main ways to explore the search space, the *Apriori* based approach and *pattern growth* based approach. They explore the search space in a Breadth-First Search (BFS) and Depth-First Search (DFS) manner respectively (see Figure 3.5). The *Apriori* strategy builds all candidate patterns of size k before building the ones of size $k+1$. New candidate patterns are obtained by joining frequent patterns of

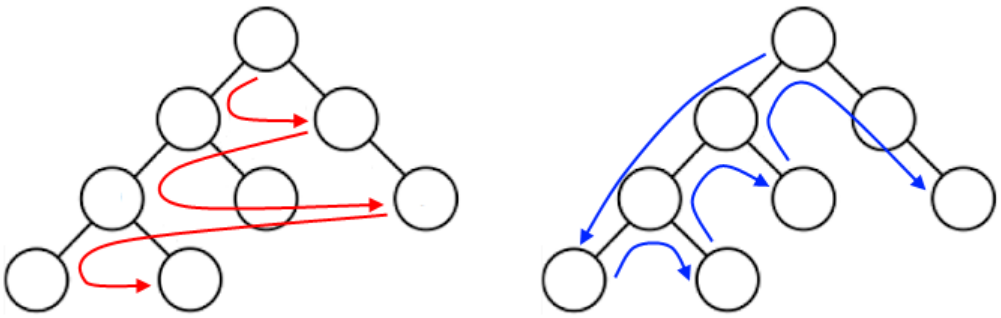

FIGURE 3.5: BFS(in red) and DFS(in blue) traversal of the same search space. Each node represents one candidate pattern.

size k that share a common subgraph of size $k-1$ such as in the AGM(*Inokuchi et al.* (2000)) or FSG(*Kuramochi and Karypis* (2001)) algorithms. The main drawback of the *Apriori* method is that it requires a lot of memory to store all subgraphs of size k needed to build the patterns of size $k+1$. Moreover, the joining operation between graphs can be complex to realize (*Kuramochi and Karypis* (2001)).

The *pattern growth* based approach generates patterns by adding edges or nodes to frequent subgraphs recursively until all their frequent super-graphs are discovered. To avoid generating subgraph patterns that do not appear in the database or have no chance of being frequent, this pattern growth-approach start from frequent edges/nodes in the database. A pattern can only be extended by adding one of those frequent edges/nodes if it is connected to occurrences of the pattern. This strategy is less costly in memory than the *Apriori* approach because it only needs to store occurrences for the current branch of the search space that is explored. gSpan(*Yan and Han* (2002)) is the first algorithm to use a *pattern growth* strategy to explore the search space. To extend the subgraphs, the authors use a *right-most extension* strategy. Let v_0 and v_n be respectively the first, and last nodes visited when performing the DFS traversal of the nodes of a graph to build its *DFS code*. v_n is called the *right-most vertex* and the direct path between v_0 and v_n in the DFS tree is called the *rightmost path*. The authors showed that to extend the DFS code representing a subgraph G in a valid way, *forward edges* (edges of G in the DFS tree) could be added only on a node of the *right-most path* and *backward edges* (edges of G not in the DFS tree) could be added only on the *right-most vertex*. With this property, invalid extension generation can be avoided, therefore reducing the search space. The authors of gSpan also showed that when exploring the search space of DFS codes in a DFS manner, similarly to the anti monotonicity property of the frequency, any non minimal code discovered can be pruned without missing any minimum DFS code. This property allows to further prune the search space while keeping the complete-

ness of the algorithm. Note that there is no clearly better strategy between DFS and BFS exploration. Depending on the data, one strategy can perform better than the other. Their efficiency with respect to the problem at hand usually has to be assessed through experimental studies.

3.5 Review of Graph Mining Algorithms

This section will shortly present inexact and exact frequent graph mining methods that have been presented in the literature. A recent and more precise survey of the frequent graph mining methods was presented in *Jiang et al.* (2013).

3.5.1 Exact Mining

Exact mining algorithm aim at finding all the frequent subgraphs in the dataset. In the general case of arbitrary graphs, such complete algorithms perform efficiently only on sparse graphs with a high number of labels on edges and nodes to diminish the cost of isomorphism tests.

One of the first exact mining algorithms, called AGM (*Inokuchi et al.* (2000)), is based on the frequent itemset mining algorithm Apriori. This algorithm assumes that all nodes in a graph have a distinct label and treats the graphs of the database as transactions and their nodes as items. It explores the search space in a BFS manner and uses a *canonical adjacency matrix* representation to compare subgraphs and avoid processing several times the same one. Candidate subgraph patterns of size n are generated by joining frequent subgraph patterns of size $n-1$ that differ in one edge.

The algorithm FSG, presented in *Kuramochi and Karypis* (2001), also using the *canonical adjacency matrix* representation and BFS exploration of the search space, is aimed at finding frequent connected subgraph patterns. While, unlike AGM it can deal with graphs having some of their nodes sharing the same label, experiments show that, when too many of those nodes are present in the database, FSG tends to not perform well.

Another algorithm, named DPMine *Vanetik et al.* (2002), reduces the number of candidate subgraph patterns by extending the patterns with edge-disjoint paths instead of simple nodes or edges. It starts by discovering all frequent paths in the database and then all frequent subgraphs with two paths. Using the Apriori strategy, to generate candidates with k paths it merges pairs of subgraph patterns with $k-1$ paths that have $k-2$ paths in common.

To generate new candidates the Apriori strategy requires keeping in memory all size $k-1$ frequent subgraph patterns to obtain the size k candidate subgraph patterns, algorithms using this approach require a lot of memory. To deal with this problem, a number of algorithms using a DFS traversal of the search space have been developed.

In the previous sections, we already discussed one of the most popular of those algorithms, namely gSpan. Another frequently cited one is MoFa (*Borgelt and Berthold* (2002)) which is directed at mining connected subgraph patterns. This algorithm stores all occurrences of the frequent subgraph patterns which allows it to generate only subgraph patterns that appear in the database. Besides, when a new subgraph pattern is generated, the subgraph isomorphism tests required to compute its support can use as basis the occurrences of the sub-pattern from which it has been grown and only match the newly added edge. MoFa also defines an order on the nodes of a subgraph pattern based on their time of addition to the pattern. When a node is added to a pattern, later extensions can only happen at this node or the ones added after him. Although this strategy can limit the generation of duplicate candidates subgraph pattern, the case still arises and has to be dealt with through isomorphism tests.

In *Huan et al.* (2003a), the authors presented an algorithm called FFSM using the *canonical adjacency matrix* representation. It keeps a list of the occurrences of the frequent subgraph patterns to avoid explicit subgraph isomorphism test when counting the support. The particularity of this algorithm is that it uses both join operations between pairs of k-edge subgraphs sharing a common (k-1)-edge subgraph and extension operations adding one edge to a k-edge graph to produce (k+1)-edge candidate subgraph patterns.

Finally, another algorithm using the pattern extension growth strategy, called GASTON, was presented in *Nijssen and Kok* (2004). This algorithm also stores all occurrences for fast isomorphism testing. The main feature of GASTON is that it speeds up the mining process by first mining simpler patterns before moving on to arbitrary graphs. Firstly, frequent paths are extracted and then extended into trees by adding one edge. Those frequent trees are further extended into arbitrary graphs by adding cycles to them. Since path and tree isomorphism can be done in polynomial time, this hierarchical approach avoids processing some of the non frequent subgraph patterns by discarding them at the path or tree mining stages.

An extensive comparison of gSpan, MoFa, FFSM and GASTON was presented in *Wörlein et al.* (2005). It concluded that storing occurrences of the frequent subgraph patterns did not considerably speed up the mining process. Indeed, although it does not store occurrences of the patterns, gSpan remained competitive with other

algorithms unless patterns became too large. The authors also noted that the candidate generation and the computing of occurrence lists (or the support counting for gSpan) cost much more computational time than the pruning of duplicate candidates. As expected, using canonical representations is better than computing explicit subgraph isomorphism tests, and GASTON's strategy to delay the generation of arbitrary graphs to later stages is even more efficient. Finally all algorithms tested scaled linearly in the size of the database.

For the single graph case, the authors of *Kuramochi and Karypis* (2006) presented two mining algorithms called HSIGRAM and VSIGRAM. They explore the search space in a BFS and DFS manner respectively. The support of the subgraph patterns corresponds to their number of non overlapping occurrences. This is computed by solving the maximum independent set problem on the overlap graph of the occurrences of each subgraph pattern. The authors implemented both exact and approximate methods to find the maximum independent set in each overlap graph. According to the experiments, both algorithm scale well on large graphs but VSIGRAM is faster than HSIGRAM. This is due to the fact that VSIGRAM reduces the cost of subgraph isomorphism tests by storing a list of the occurrences of patterns along the DFS path.

As was already mentioned, exact mining algorithms discover often too many patterns while the user might be interested in a specific kind of patterns. A first possibility is to mine closed subgraph patterns, i.e., patterns for which all super-patterns have lower support. A very popular algorithm in this category is CloseGraph (*Yan and Han* (2003)). The authors noted that, in the case of biochemical data, 2,000 closed patterns could achieve the same accuracy than 1,000,000 patterns. Their algorithm is based on gSpan. It explores the search space in a DFS manner, uses DFS codes and right-most extension. This algorithm prunes the search space by only extending patterns that are closed.

Another type of pattern that have been the focus of some algorithms are maximal patterns, i.e., patterns for which no super-pattern are frequent. One such algorithm frequently cited is SPIN (*Huan et al.* (2004a)). Similarly to GASTON, this algorithm first mines frequent trees which are then extended into graphs by adding edges to them. Among all possible spanning trees for a given subgraph pattern, the canonical spanning tree is defined as the maximal one according to a total order on trees (*Chi et al.* (2003); *Huan et al.* (2004b)). The subgraph patterns with isomorphic canonical spanning trees are put in the same equivalence class. SPIN tries to enumerate only subgraph patterns that are maximal in their equivalence class. The experiments conducted by the authors showed that SPIN could mine a database much more quickly than gSpan or FFSM.

Some graph mining algorithms choose to focus on discovering clique patterns. For example, CLAN (*Wang et al.* (2006)), mines frequent closed cliques in transactional databases graphs with labels only on the nodes. This algorithm takes advantage of the fact that cliques are fully connected sets of nodes, this means that any two cliques with the same nodes will have the exact same structure. Therefore each clique can be represented by a simple sequence of the labels of all its nodes. The minimum such sequence, according to a lexicographic ordering on the labels of the nodes, is the *canonical code* of the clique. The search space is explored in a DFS manner and patterns are extended by adding a node along all the edges that connect it to the other nodes of the pattern. The authors show that any prefix of the *canonical code* of a clique is also *canonical*. This means that the generation of *non canonical codes* can be avoided by only adding a node to a clique C if its label is lexicographically no smaller than the last label in the code of C. The authors also note that any node of a size k clique must have a degree of at least $k-1$. Therefore when scanning the database to find the possible extensions of a size k clique, only nodes with a degree of at least $k-1$ are considered. The authors of *Zeng et al.* (2006) extended this work and presented the algorithm Cocain to mine closed frequent quasi-cliques. A graph of size k is a γ-quasi-cliques if the degree of all its nodes, have a degree above $\gamma(k-1)$, where $0.5 \leq \gamma \leq 1$ is a user parameter. Note that a γ-quasi-clique is fully connected when $\gamma = 1$ and singleton graphs are considered as γ-quasi-cliques. The authors defined that two γ-quasi-cliques are γ-isomorphic if they have the same size and there is a bijection between their nodes preserving their labels. As for CLAN, a *canonical representation* of quasi-cliques is used to uniquely represent patterns. The search space, explored in a DFS manner, is pruned using structural properties of the quasi-cliques.

Algorithm	Type of Pattern	Representation	Candidate Generation	Exploration	Database
AGM	graphs	CAM	level-wise join	BFS	transactional
FSG	graphs	CAM	level-wise join	BFS	transactional
DPMine	graphs	n/a	path enumeration + level-wise join	BFS	transactional & single graph
gSpan	graphs	DFS Code	rightmost path extension	DFS	transactional
MoFa	graphs	n/a	extension	DFS	transactional
FFSM	graphs	CAM	join + extension	DFS	transactional
GASTON	graphs	n/a	path, tree and graph enumeration	DFS	transactional
HSIGRAM	graphs	CAM	level-wise join	BFS	single graph
VSIGRAM	graphs	CAM	extension	DFS	single graph
CloseGraph	closed graphs	DFS code	right most extension	DFS	transactional
SPIN	maximal graphs	canonical spanning tree	tree join + extension	BFS	transactional
CLAN	closed cliques	vertex label sequence	node extension	DFS	transactional
Cocain	closed quasi-cliques	vertex label sequence	node extension	DFS	transactional
FREQT	trees	depth-label sequence	rightmost path extension	DFS	transactional
TreeMiner	trees	label sequence	equivalence class extension	BFS	transactional

FIGURE 3.6: Differences between the mining algorithms discussed in section 3.5.1. In this table, CAM stand for *Canonical Adjacency Matrix*, DFS for *Depth First Search* and BFS for *Breadth First Search*

A large number of mining algorithms have been developed to mine trees. Among them, FREQT (*Kenji et al.* (2004)), uses a right-most extension strategy, adding new nodes only on the right-most branch of trees to avoid duplicate candidate generation. It also stores, for each pattern, a list of the right-most leaves of each occurrences to quickly find the occurrences of super-patterns when counting the support. Another popular tree mining algorithm, named TreeMiner (*Zaki* (2002)), represents trees as sequences of labels in the order they are discovered when traversing it in a DFS manner. Trees of size k that are equivalent up to the $(k-1)$th node are joined to produce candidate tree patterns of size $k+1$. FREQT and TreeMiner are frequently used by other approaches as a basis for comparison. Many other tree mining algorithms exist but since this family of graphs is not the focus of this thesis their details are omitted in this document. The interested reader can refer to the recent survey on frequent graph mining presented in *Jiang et al.* (2013) to learn more about frequent tree mining algorithms. Table 3.6 summarizes the differences between the mining algorithms we discussed in this section.

3.5.2 Inexact Mining

Inexact frequent graph mining algorithms allow subgraphs with minor variations to be counted as occurrences of the same pattern. While this increases the number of frequent subgraph patterns, those approaches are usually not interested in finding all of them. Instead they focus on extracting more interesting patterns that can capture relationships among the data that would have been discarded by exact frequent graph mining algorithms because of small variations in the graphs of the database.

One of the first and most popular such algorithms is SUBDUE (*Holder et al.* (1994)). A graph edit distance is used to match patterns with the graphs in the database. It performs a Beam Search of the search space, which consists in Breadth-First Search exploration but only keeping the best subgraph patterns at each iteration.

GREW (*Kuramochi and Karypis* (2004)) is another inexact frequent graph mining algorithm. This algorithm is designed to mine single large graphs to discover patterns with a number of node disjoint occurrences (occurrences with no nodes in common) higher than a frequency threshold. Moreover nodes of the dataset can contribute to the support of multiple patterns only if they have a subgraph/super-graph relationship. Like FGM, this algorithm explores the search space in a Breadth-First manner. New candidate subgraphs are generated by merging frequent subgraphs of previous iterations connected by one or several edges. Note that this merging process is not limited to increasing the size of successive subgraphs by only one node or edge at a time. Since GREW mines node disjoint patterns (unless they have a subgraph/super-graph relationship), it can rewrite the input graph by collapsing nodes of occurrences of frequent patterns into single nodes reducing the relative size of the graph to be mined at each iteration. This has for consequence the underestimation of the frequency of the patterns, therefore some of them are discarded while they are actually frequent. The authors compared their algorithm to SUBDUE and showed that it could discover patterns 4 times bigger in a lot less time.

Another approximate frequent graph mining algorithm, called Monkey, was presented in *Zhang et al.* (2007). This algorithm mines transactional databases of graphs. Instead of using conventional subgraph isomorphism to compare graphs, the authors define that a graph G_1 is β edge subgraph isomorphic to a graph G_2 if there is a subgraph isomorphism between G_1 and G_2 for which at most β edges are not preserved, noted $G_1 \subseteq_\beta G_2$. Let $sup(D, G, \beta) = \{G_i | G \subseteq_\beta G_i, G_i \in D\}$ be the set of graphs in the database D to which G is β edge subgraph isomorphic. Given three parameters, α, β and γ, in order for a subgraph G to be approximately frequent, it is required that $|sup(D, G, \beta)| \geq \gamma$ and that each edge of G appears in at least $\gamma - \alpha$ graphs of $sup(D, G, \beta)$. This algorithm explores the search space in a DFS

manner to discover frequent approximate trees. Then, those approximately frequent trees are recursively extended by adding one edge connecting two of their nodes until the candidate subgraph is not approximately frequent or has been searched before. The authors showed that their algorithm could find interesting patterns that exact mining algorithms would miss (*Zhang et al.* (2007)). According to the authors this algorithm was 20% faster than a basic DFS based algorithm, but increasing β would slow it down heavily due to the explosion of approximately frequent patterns.

The same β edge subgraph isomorphism is used by the RAM algorithm of *Zhang and Yang* (2008). This algorithm uses feature vectors instead of canonical codes to check if a candidate subgraph pattern has already been processed. For each subgraph pattern, a feature vector is built based on simple features such as the connectivity, the degree of nodes, the size of the graph, the weight of the Minimum Spanning Tree or other features that can be computed in linear time when growing the pattern. Similarly to the usage of canonical codes, if a pattern B is discovered after another pattern A and they have equivalent feature vectors, B is discarded and the search space pruned. Note that two isomorphic graphs have the same feature vector, but different graphs might as well. This means that some patterns might be discarded even though they have not been processed. This is overcome by growing patterns by adding edges to them in random order. This means that for some execution of the algorithm a pattern A might be reached before another pattern B, while for another run, B might be found before A. The authors state that if a pattern has probability x of being discovered in one run, it has probability $1-(1-x)^p$ of being discovered in p runs. For example if 20% of patterns might be missed in one run, only 1% would be missed after 3 executions. Experiments showed that RAM could discover important patterns that where missed by exact mining methods.

In *Chen et al.* (2007) the authors presented the algorithm gApprox which mines approximate patterns from a single large graph with edges weighted by a positive real number. In this setup, an approximate pattern is a fragment of the input graph. The approximation comes from the matching technique used by the authors to compare a pattern with a candidate subgraph. First, each node is associated a list of matchable nodes built using a thresholded similarity measure. The tightness of the relationship between (v_1, v_2) is defined as the sum of the costs on the edges of the shortest path between v_1 and v_2, noted $dist(v_1, v_2)$. In this way, the cost of mapping (v_1, v_2) to its image $(m(v_1), m(v_2))$ under the mapping m is equivalent to the absolute difference between the tightness of (v_1, v_2) and $(m(v_1), m(v_2))$. The degree of approximation associated to a particular mapping m of the nodes of a pattern P to a graph G is equal to :

$$approx(P \xrightarrow{m} G) = \sum_{v \in V_P} d(v, m(v)) + \sum_{v_i, v_j \in V_P} |dist(v_i, v_j) - dist(m(v_i), m(v_j))|,$$

where V_P is the set of nodes of P and $d(v, m(v))$ is a dissimilarity measure between nodes. Given an error tolerance Δ, a graph G is an approximate occurrence of a pattern P if $approx(P \xrightarrow{m} G) \leq \Delta$. The support of a pattern corresponds to its maximum number of node disjoint approximate occurrences. Finding such a maximal number of disjoint occurrences requires solving the NP-hard Maximum Independent Set problem on the overlap graph of the occurrences of P. To avoid such a costly computation the authors use an upper bound of the maximum number of node disjoint occurrences. Finally, the search space is explored in a DFS manner, finding all possible connected sets of nodes in the input graph. Experiments on protein-protein interaction networks showed that gApprox could efficiently discover a large number of biologically interesting patterns.

More recently, *Jia et al.* (2011) presented APGM, an approximate algorithm aimed at mining transactional graph databases. The authors define a graph $G = (V, N, L_e, L_n)$ to be approximate subgraph isomorphic to a graph $G' = (V', N', L'_e, L'_n)$ in the database, if there exist an injection $f : V \to V'$ such that

- $S = \prod_{v \in V} \frac{M_{L_n(v), L'_n(f(v))}}{M_{L_n(v), L_n(v)}} \geq \tau$,
- $\forall v_1, v_2 \in V, (v_1, v_2) \in E \Rightarrow (f(v_1), f(v_2)) \in E'$, and
- $\forall (v_1, v_2) \in E, L_e(v_1, v_2) = L'_e(f(v_1), f(v_2))$.

Where S is the approximate subgraph isomorphism score, M is a matrix indexed by the labels of the two graphs such that $M_{i,j}$ represents the cost of changing label i to label j, and τ is a given similarity threshold. Since there could exist multiple mappings f satisfying the approximate subgraph isomorphism test, the matching score between a pattern and a graph in the database is defined as the maximum approximate subgraph isomorphism score obtained by this pattern in this graph. The support of a pattern corresponds to the sum of the matching scores with all graphs in the database, divided by the size of the database. This approach allows to match together graphs that are structurally equivalent but that have nodes labeled differently. An upper bound of the support is used to efficiently prune the search space explored in a DFS manner. The experiments conducted by the authors showed that APGM could discover more and bigger patterns in less time than an exact mining approach.

Some recent approaches have been focusing on mining *probabilistic graphs*, i.e., graphs with a probability of existence assigned to each edge. In this scenario an *uncertain graphs* is a graph $\mathcal{G} =< V_\mathcal{G}, E_\mathcal{G}, P >$ where $P(e)$ is the probability of existence of edge $e \in E_\mathcal{G}$. Each *uncertain graph* implicates a set of exact graphs $I(\mathcal{G}) = \{I =< V_\mathcal{G}, E_I > | E_I \subseteq E_\mathcal{G}\}$. The probability that an uncertain graph G implicates an exact graph I is:

$$P(\mathcal{G} \Rightarrow I) = \prod_{e \in E_I} P(e) \prod_{e' \in E_\mathcal{G} \setminus E_I} (1 - P(e')).$$

In *Zou et al.* (2009), the authors present the algorithm MUSE, aimed at mining frequent subgraph patterns in transactional databases of *uncertain graphs*. Given an *uncertain graph database* $D = \mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n$, the probability that a pattern p occurs in one of the *uncertain graphs* $\mathcal{G}_i$ is:

$$P(p \subseteq \mathcal{G}_i) = \sum_{I \in I(\mathcal{G}_i)} P(\mathcal{G}_i \Rightarrow I) \cdot \Psi(I, p), \tag{3.1}$$

where $\Psi(I, p) = 1$ if p is *subgraph isomorphic* to I, 0 otherwise. The expected support of p in database D is computed as follows:

$$esup_D(p) = \frac{1}{|D|} \sum_{i=1}^{|D|} P(p \subseteq \mathcal{G}_i).$$

The algorithm performs a depth-first search of the search space of candidate frequent subgraph patterns and approximates their expected support. Experiments showed that the approximation of the expected support resulted in few false frequent patterns returned. The authors also showed that the time complexity of their algorithm increased linearly with the number of *uncertain graphs* in the database.

In *Jin et al.* (2011), the authors describe an algorithm to discover *highly reliable subgraphs* from an uncertain graph $\mathcal{G} =< V_\mathcal{G}, E_\mathcal{G} >$. A *highly reliable subgraph* is a subset of nodes of the input graph that has a probability of remaining connected under uncertainty, called *reliability*, higher than a given threshold. The *reliability* of a particular subset of nodes is computed in the same way than the probability of a pattern occurring in *uncertain graphs* for MUSE, but this time the indicator function Ψ in equation 3.1 indicates connectivity instead of *subgraph isomorphism*. The authors estimate the *reliability* of a subset of nodes $V_s \subseteq V_\mathcal{G}$ through Monte-Carlo sampling. N graphs are sampled according to their probability of being implicated by G. The number of them in which the subset V_s is connected is its *estimated reliability* in $\mathcal{G}$. The authors reformulate this as the problem of mining *frequent cohesive sets*.

Given a database of graphs D that have the same nodes (in this case the set of sampled implicated graphs of $\mathcal{G}$), *a frequent cohesive set* is a subset of nodes V_s that is connected in more graphs of the database than a given threshold. The frequency of a *cohesive set* corresponds to the *estimated reliability* of V_s in $\mathcal{G}$ and is not anti-monotonic, which means that a super-pattern can be more frequent than some of its sub-patterns. In order to still be able to bound the search space, the authors first find *the maximal frequent cohesive sets* (MFCS) by relaxing the problem into the one of finding *the maximal frequent linked sets* (MFLS). A *linked set* is a subset of nodes that are connected, possibly through nodes outside the set. The MFLS problem can be turned into a *frequent itemset* problem by computing the connected components of each graph of D and treating the node sets of each connected component as a transaction. Since all the MFCS are contained in the set of MLFS, the authors can recover the MFCS by a peeling procedure. For each MFLS m, the database is peeled by removing all nodes that are not in m. This might disconnect m since it is likely to be connected through nodes outside of it. Then the peeled database is mined to discover new MLFS that correspond to MFCS of the original database D. This process is repeated recursively until all MFCS have been found.

3.6 Graph Representations of Videos

There exist various methods to represent images in ways that capture more semantic information than a simple matrix of pixels. For example, color histograms (*Swain and Ballard* (1991)) can capture the distribution of the pixels' color. Two images can be compared by measuring the distance between their color histograms. While this representation is simple, a single histogram cannot capture the geometry and topology of an image and it is possible to have two very different images with similar histograms. Some approaches use a vocabulary of visual words. The images are described by histograms of the frequency of appearance of each word(*Nowak et al.* (2006)). While visual words can retain some spatial information at a local level, like texture patterns, this image representation suffers from the same lack of spatial information at a global level than color histograms. Another possibility consists in representing images at multiple scales using pyramidal representations (*Glantz et al.* (2004)) or quad-trees (*Samet* (1984)). Pyramidal and quad-tree representations can be used to compare images at multiple scales, exploiting both the local and global information, but they are difficult to update when images change over time and they do not model their topology explicitly. It is also possible to use strings to describe images using the Freeman code (*Freeman* (1961)). Eight symbols are used to encode

each of the eight possible directions around a pixel (i.e., left,right,top,bottom, and both diagonals). A shape is encoded by the directions taken while following the pixels of its contour. This technique is commonly used to match handwritten symbols by measuring the similarity of the codes (e.g., using the edit distance for example).

In this thesis we focused on graph representations of images that model their topology. Some types of representations are particularly suited for certain problems. For example, in *Kotropoulos et al.* (2000), the authors compare human faces by using grids, measuring visual features at each of its node. The amount of deformation needed to fit the grid representing one face to the grid representing another face serves as a measure of similarity. This approach is well suited to face recognition applications since common features such as two eyes, a nose and a mouth are present on all human faces. But it cannot deal with all the variety in the objects that can appear in a video and it is not robust to variations in view point. More generally, graph representations of images are mostly based on two types of visual features: interest points or segmented regions.

3.6.1 Graph Representations of Images Based on Interest Points

To represent the topology of images, one possibility is to extract interest points and then build a Neighborhood Graph, connecting points according to some neighborhood relationship. For example, we can build the Euclidean Minimum Spanning Tree (*Agarwal et al.* (1991)), which corresponds to the spanning tree connecting the nodes with the shortest edges in term of euclidean length. Interest points can also be connected to form a Relative Neighborhood Graph (*Toussaint* (1980)), where two nodes v_1 and v_2 are connected if the intersection between the two circles centered on v_1 and v_2 and with radius v_1v_2 does not contain any other node. Similarly, the Gabriel Graph (*Gabriel and Sokal* (1969)), which connects two nodes v_1 and v_2 if the circle of diameter v_1v_2 does not contain any other node, can be used to represent images with interest points. Finally, interest points can also be used to build a Delaunay triangulation (*Lee and Schachter* (1980)), which consists in dividing the space into triangular faces so that the circumscribed circle of every faces does not contain any node. Unsuccessful attempts to triangulate interest points for image processing applications were described in *Samuel* (2011). Figure 3.7 shows an example of Euclidean Minimum Spanning Tree, Relative Neighborhood Graph, Gabriel Graph and Delaunay Triangulation built for a particular set of points.

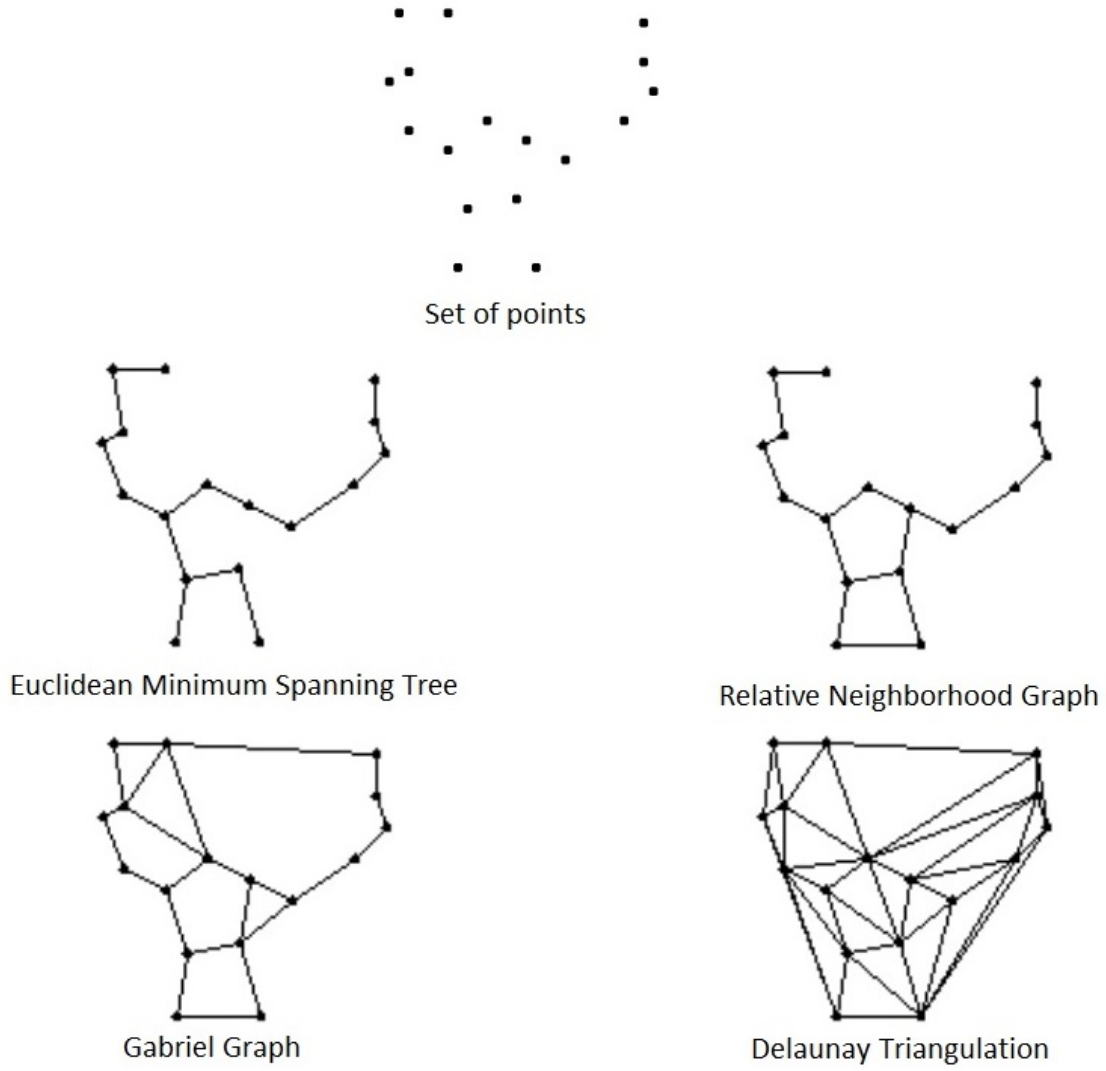

FIGURE 3.7: Neighborhood graphs (*Samuel* (2011))

3.6.2 Graph Representations of Images Based on Segmented Regions

Segmented images are often the basis of graph representations of images. In *Cai et al.* (2013) the authors describe an approach that consists in connecting with an edge the nodes of the graph, representing the segmented regions, if the distance between the two corresponding regions is lower than their average radius. The four models based on interest points described in the previous section can also be used with segmented regions by considering their barycenter's coordinates. There also exist models specific to segmented images. One of the first such models is the Region Adjacency Graphs (RAG) (*Trémeau and Colantoni* (2000)). In a RAG, each node

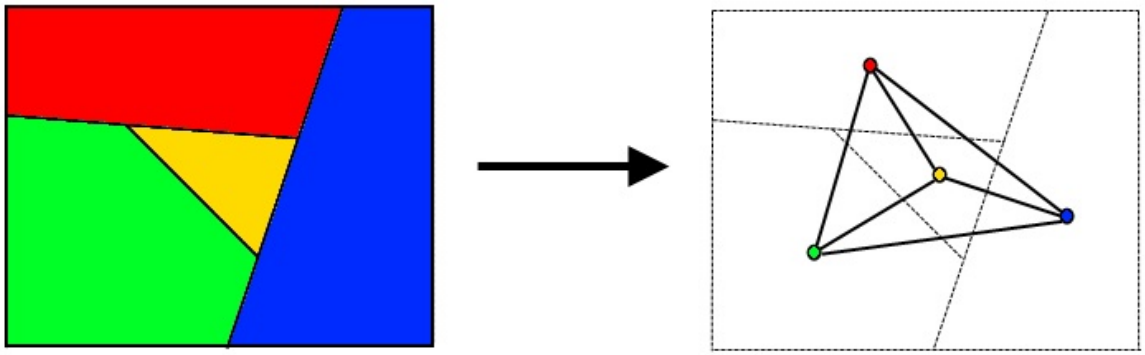

FIGURE 3.8: Example of segmented images with 4 regions *(left)* and its corresponding RAG *(right)*

represents a region and an edge connects two nodes if the corresponding two regions are adjacent in the image (see Figure 3.8). This representation cannot model multiple adjacency between two regions or distinguish between simple adjacency and inclusion. To deal with this problem, Dual Graphs (*Kropatsch* (1995)) have been proposed. This model is composed of two graphs. The first one is a RAG G that can have multiple edges between two same regions to model the multiple adjacency. The edges of this graph divide the plane into faces. Based on G a dual graph G' is built in which nodes represent the faces of G and for each edge in G separating two faces f_1 and f_2, an edge is set in G' between the nodes representing f_1 and f_2. The drawback with this model is that it requires to maintain two graphs for each image. Further more, the inclusion of a region R_1 into another region R_2 is modeled by a loop on R_2 surrounding R_1. This means that the geometry of the edges has to be analyzed to detect inclusion relationships. Another model used to model the topology of segmented images are the Combinatorial Maps (*Lienhardt* (1991)). They encode explicitly the orientation of edges around nodes. They can be seen as a decomposition of the image into faces (i.e., segmented regions in our case) which are further decomposed into edges. Figure 3.9 shows an example of such a decomposition. Each edge results in two half edges called darts. Note that although we only consider image representations in two dimensions, combinatorial maps can be used in higher dimensions (*Damiand* (2001)). In the 3 dimensional case, an object can be decomposed into volumes, then faces and then darts.

Formally, a combinatorial map is a triplet $M = (D, \beta_1, \beta_2)$ where D is a set of darts, β_1 a permutation on D defining the faces by relating each dart to the next one around the face, and β_2 an involution on D representing the adjacency between the

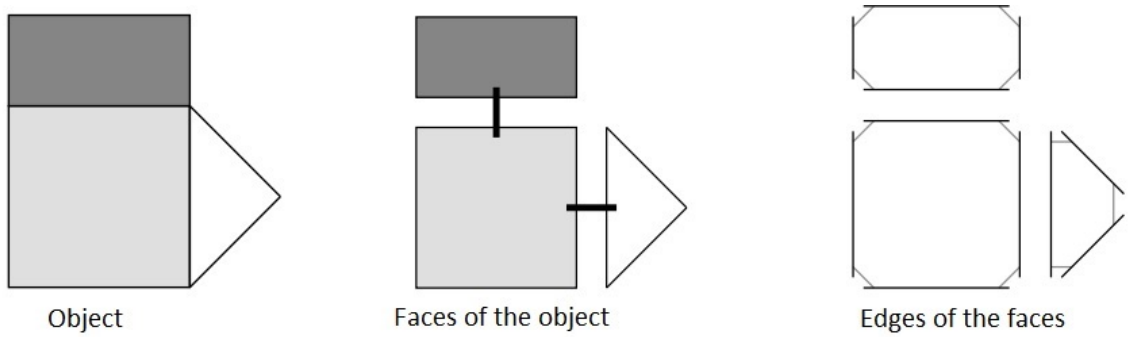

FIGURE 3.9: Example of decomposition of an object into darts (*Damiand* (2001))

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
β_1	2	3	4	5	6	7	1	9
β_2	15	14	18	17	10	9	8	7

TABLE 3.2: Permutation β_1 and involution β_2 of the combinatorial map in Figure 3.10 (*Samuel* (2011)

faces by relating darts on the same edge belonging to two adjacent faces. Figure 3.10 gives an example of a combinatorial map representing the object in Figure 3.9, and table 3.2 shows the values for β_1 and β_2. As shown in the work of *Damiand* (2001) combinatorial maps can be used to represent the inter pixel boundary between regions of a segmented image and therefore model the geometry of the regions. Unfortunately a lot of darts might be required to represent complex shapes.

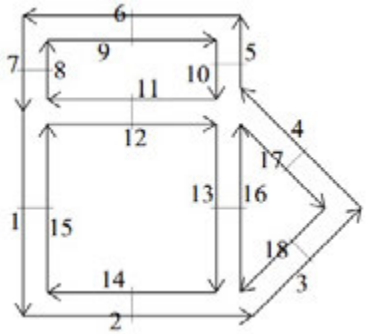

FIGURE 3.10: Example of combinatorial map representing the object in Figure 3.9 (*Samuel* (2011))

3.6.3 Video Representation

Videos can be represented by a set of graphs. For instance in *Chang et al.* (2004) the authors represent each frame of a video with a RAG. Any of the graph representations described in the previous section could be used. We chose to use segmented

regions as the basis of our graph representation and focused on models using plane graphs since those are more efficient to compare since the plane isomorphism test can be solved in polynomial time.

The method of *Cai et al.* (2013) is simple but does not necessarily results in plane graphs. Dual graphs require to manage two graphs per image which makes them harder to handle. The combinatorial maps model the topology of images accurately and result in plane graphs that are similar to or graphs. However their structure is complex and they are better suited to model high dimensional objects.

We chose to use a simple RAG representation with multiple edges between nodes when two regions have multiple adjacency resulting in multigraphs. In our video application, each frame is segmented individually using the image segmentation algorithm of *Felzenszwalb and Huttenlocher* (2004) and represented by a plane graph G_i. This segmentation algorithm is discussed in section 2.1.2. We also used the video segmentation algorithm of *Grundmann et al.* (2010) as a basis for our graph construction in some of our experiments. Each node in those graphs represents a segmented frame region, and is associated to the coordinates (x, y) of the barycenter of this region. One special node is added to represent an unbounded region encompassing all the image. Informations on the regions (e.g., size, average color etc...) and on their borders (e.g, length for example) can be added to nodes and edges as labels. In our experiments we mostly used two types of node labels based either on a discretization of the size of the regions or a discretization of their average color. The inclusion relationship can also be represented by a label on the edges. For the sake of comparison, we also tried another plane graph representation based on the segmented regions by triangulating the nodes using a Delaunay triangulation.

With those graphs representations, we expect to be able to detect and track the main objects of a video by mining frequent plane subgraph patterns from the database of plane graphs representing it.

3.7 Conclusion

We have presented various existing approaches to tackle the graph mining problem. In the context of this thesis, we want to mine frequent subgraph patterns from a transactional database of graphs representing images of a video. Since videos evolve through time, which means that exact patterns are unlikely to be present in all frames, inexact mining methods seem good candidates to tackle our problem. Moreover, the image segmentation process can introduce a lot of noise. For instance some regions can be regrouped in a single one, or conversely a single region can be fragmented into

multiple smaller ones from one frame to another. This means that we would need a mining technique that can deal with uncertainties during the matching phase on both the nodes and the edges. Unfortunately, efficient inexact mining methods usually only allow uncertainties on the edges (e.g, gApprox) or the nodes (e.g, APGM). Often they require the identification between the different graphs of the database to be known (e.g., when mining social networks), or mine single graphs (e.g., GREW). As shown by *Zhang et al.* (2007), while small uncertainties can be tackled, it is difficult to deal with large variations without the search space of candidate patterns becoming too large. The approach of the RAM algorithm can deal efficiently with uncertainties but the features used to represent each graph need to be carefully selected and tuned in order to accurately account for the structural properties of data.

Exact mining approaches consider similar subgraphs to be occurrences of different patterns. However interesting relationships can still be captured by the output exact frequent patterns. While exact graph mining is intractable in large datasets of dense graphs, focusing on a particular subclass of patterns is a good strategy to heavily reduce the number of candidate patterns that need to be generated and processed.

The next part of this document will discuss how plane graphs representing a video can be efficiently mined and how the number of extracted patterns can be reduced by taking into account spatial and temporal constraints. We will also show the meaningfulness of those patterns for object tracking in videos.

Part II

Contributions

CHAPTER 4

Mining Spatio-Temporal Patterns in Dynamic Graphs

4.1 Introduction

In our work we represent videos by series of plane graphs (see definition 3.5 of a plane graph), with one graph per frame. Those graphs are obtained by first segmenting each frame using the color segmentation algorithm of *Felzenszwalb and Huttenlocher* (2004) and representing each segmented regions by a node. Those nodes are connected by mean of adjacency relationship (*Trémeau and Colantoni* (2000)) or Delaunay triangulation (*Lee and Schachter* (1980)). With this method each video can be represented by a series of plane graphs that can be seen as a dynamic graph in which both nodes and edges evolve.

This chapter discusses our contributions in the field of data mining. More precisely, we investigate how plane graphs can be efficiently mined in order to extract meaningful patterns, i.e., in our case patterns that represent objects or parts of them. To do this, we take advantage of a polynomial plane subgraph isomorphism algorithm (*Damiand et al.* (2009)) to efficiently find the occurrence of each pattern. The main focus will be on how to exploit the spatio-temporal information of videos to reduce the number of patterns discovered. Indeed, in a video, objects move smoothly along a trajectory, therefore the occurrences of a pattern representing the same object should not be too far apart in term of number of frames and spatial distance. To deal with this, we developed two distances, a temporal one and a spatial one, and use them to output only patterns that meet some spatial and temporal constraints.

A first section will provide definitions for the concepts we will use. Then section 4.3 will detail our PLAne GRAph Mining algorithm called PLAGRAM and its variants, DYPLAGRAM, which only uses the temporal constraint and DYPLAGRAM_ST, which exploits both the spatial and temporal constraints during the mining phase. Those

three variants all explore the search space of candidate plane subgraph patterns in a depth-first manner and use canonical codes to avoid processing the same subgraph several times. Another key difference with other graph mining algorithms is that in our approach we extend subgraph patterns by adding to them complete faces (see definition 3.4) instead of just one node or edge. This extension strategy speeds up the exploration of the search space but means that our algorithms are restricted to finding plane subgraph patterns that are 2-connected, i.e., each face composing it is a simple cycle. In the section 4.4 we discuss the results of the experiments conducted to show the efficiency of our dedicated mining algorithm in comparison with the general purpose gSpan algorithm (*Yan and Han* (2002)). We also study the impact of the spatio-temporal constraints on the efficiency of our algorithms before concluding this chapter in section 4.5.

4.2 Definitions

4.2.1 Dynamic Plane Graph and Frequency of Plane Subgraph Patterns

The set of plane graphs representing the frames of a video can be seen as a dynamic plane graph in which both nodes and edges evolve through time. The frames in a video are ordered, and this order is taken into account when computing spatio-temporal patterns. We thus define a dynamic plane graph as an ordered set of graphs.

Definition 4.1 (Dynamic plane graph)**.** A dynamic plane graph $\mathcal{D}$ is an ordered set of plane graphs $\{G_1, G_2, .., G_n\}$. Each node of these graphs is associated to spatial coordinates (x, y).

Building on the definition of an occurrence of a plane graph in a larger graph (see definition 3.11) we define an occurrence of a plane graph in a dynamic plane graph and its frequency.

Definition 4.2 (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 $\text{Occ}(P) = \{(i, f) \mid f \text{ is an occurrence of } P \text{ in } G_i\}$.

Definition 4.3 (Barycenter of an occurrence)**.** The barycenter of an occurrence is the average of the coordinates of its nodes.

Definition 4.4 (Frequency of a plane graph in a dynamic graph)**.** The frequency $\text{freq}(P)$ 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., $|\{i \mid \exists f,\ (i, f) \in \text{Occ}(P)\}|$.

4.2.2 Occurrence Graph and Spatio-Temporal Patterns

In typical subgraph mining problems, where the input collection of graphs does not represent a dynamic graph, the frequency freq(P) of a pattern graph P, which is used by PLAGRAM, is computed regardless of the fact that its occurrences may be far apart with respect to time and/or space. Indeed, in the context of a video tracking application, the tracked object changes of appearance smoothly, due to changes in view point and orientation. Therefore we expect a pattern representing a part of it in a frame t to also be present in nearby frames, i.e., frames with a time-stamp close to t with respect to some temporal constraint. Similarly, the tracked object is expected to move smoothly. Thus, occurrences of a pattern representing the same object should not be too far, in term of spatial distance, in consecutive frames.

To define a frequency that takes into account the spatio-temporal distance between the occurrences, we define in this section the notion of an occurrence graph in which occurrences of the same pattern that are close to one another are linked. Then, we define spatio-temporal patterns in this occurrence graph and the associated frequency.

Note that we use two different definitions of an occurrence graph, one for PLAGRAM and DYPLAGRAM, and one for DYPLAGRAM_ST.

4.2.2.1 Definition of Occurrence graph Used by PLAGRAM and DYPLAGRAM

Definition 4.5 (Occurrence graph and Spatio-temporal 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 barycenters is lower than a spatial threshold ϵ and their temporal 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'$\}$. In the rest of this document, each connected component of the occurrence graph of P is called a *spatio-temporal pattern* S based on P. The frequency of a spatio-temporal pattern corresponds to the number of frames in which it has at least one occurrence.

Definition 4.6 (Frequency of a spatio-temporal pattern)**.** The frequency of a spatio-temporal pattern S in a dynamic graph $\mathcal{D}$, denoted $\text{freq}_{st}(S)$, is $|\{i \mid \exists f,\ (i, f) \in S\}|$.

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.

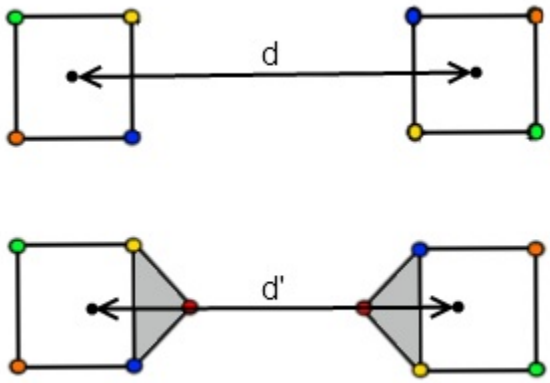

FIGURE 4.1: The euclidean distance between the barycenters of the two occurrences of the sub-pattern (upper part of the image) is higher than the one of the super-pattern in the bottom. This means the occurrences of the super-pattern can be close while the corresponding ones of the sub-pattern are not, resulting in a super-pattern that can be frequent, with respect to freq_{st}, while its sub-pattern is not

However, when defining the occurrence graph as in definition 4.5, 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 spatio-temporal 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 spatio-temporal patterns S' and S''. In other words, two spatio-temporal patterns S' and S'' based on P' may be infrequent, while the spatio-temporal pattern S based on P is frequent. This is illustrated in Figure 4.1.

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

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

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

Definition 4.7 (Subsequence frequency)**.** The subsequence frequency of a plane graph P in $\mathcal{D}$, denoted $\text{freq}_{seq}(P)$, 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 $\text{freq}_{st}(S)$, since the sequence of the G_is that contains an occurrence of S satisfies (a) and (b) in Definition 4.7. 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.

4.2.2.2 Definition of Occurrence graph Used by DyPlagram_st

While freq_{seq} has the anti-monotonicity property, it only accounts for the temporal constraint. To "give" the anti-monotonicity property to freq_{st} we need to redefine the occurrence graph so that it is not possible anymore for two distinct spatio-temporal patterns, based on the same pattern, to merge in a single spatio-temporal pattern composed of more occurrences than the two initial ones. To do so we used a different spatial distance than the euclidean distance between the barycenters of the occurrences. Instead we measure the distance between each node of one occurrence and its corresponding node in the other occurrence and keep the maximum one.

Definition 4.8 (Distance between occurrences)**.** The distance between two occurrences $o = (i, f)$ and $o' = (i', f')$ of a plane graph $P = (V, E, F, f_e, L)$ in a dynamic graph $\mathcal{D}$ is defined as: $\text{dist}(o, o') = \max_{s \in V} d(f(s), f'(s))$, where d denote the Euclidean distance between the nodes.

This distance has an anti-monotonic property:

Proposition 4.9. *For any pairs of patterns $P = (V, E, F, f_e, L)$ and $P' = (V', E', F', f'_e, L')$ such that P is a plane subgraph of P' and two occurrences $o_1 = (f_1, i)$, $o_2 = (f_2, i)$ of P and two occurrences $o'_1 = (f'_1, i)$, $o'_2 = (f'_2, i)$ of P' such that f_1 is a restriction of f'_1 (i.e., $f_1 = f'_1$ on V) and f'_2 is a restriction of f_2, then we have $dist(o_1, o_2) \leq dist(o'_1, o'_2)$.*

Proof. Let $P = (V, E, F, f_e, L)$ be a pattern and two of its occurrences $o_1 = (i, f_1, i)$, $o_2 = (j, f_2)$. Extending P to $P' = (V', E', F', f'_e, L')$, with $o'_1 = (i, f'_1)$ and $o'_2 = (j, f'_2)$ the occurrences of P' respectively extended from o_1 and o_2, we have $f_1(V) \subseteq f'_1(V')$ and $f_2(V) \subseteq f'_2(V')$. Therefore $\max_{s \in V} d(f_1(s), f_2(s)) \leq \max_{s' \in V'} d(f'_1(s'), f'_2(s'))$. □

Figure 4.2 gives a graphical example of this spatial distance.

To redefine the occurrence graph so that freq_{st} has the anti-monotonicity property, we used the parent relationship on patterns, defined by the depth-first traversal of the search space performed by our mining algorithm.

Definition 4.10 (Parent of a pattern and of an occurrence)**.** Given a pattern P with $n \geq 2$ internal faces, the pattern $p(P)$ with $n - 1$ faces that can be extended into P by the addition of one face is called the parent of P. And given an occurrence

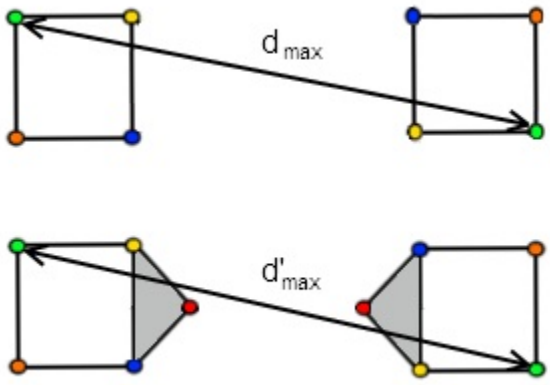

FIGURE 4.2: Extending a pattern cannot result in a lower spatial distance, therefore this distance has an anti-monotonic property.

$o = (i, f)$ of P, we call the *parent of* o the occurrence $p(o) = (i, f')$ such that f' is the restriction of f to the nodes of $p(P)$.

The definition of the parent of an occurrence is then used to define the occurrence graph. The nodes of the occurrence graph are the occurrences of a pattern and the edges connect "close" occurrences. This graph is constructed for each pattern in the mining algorithm.

Definition 4.11 (Occurrence graph and Spatio-temporal pattern)**.** Given a spatial distance dist, a spatial threshold ϵ, a temporal threshold τ, a plane graph $P = (V, E, F, f_e, L)$ and a dynamic graph $\mathcal{D}$, we define the *occurrence graph of* P as an oriented graph whose set of nodes is $\mathrm{Occ}(P)$.

- If P has only one face, then there is an edge between the occurrences $o_1 = (i, f_1)$ and $o_2 = (j, f_2)$ such that $j > i$ if $0 < j - i \leq \tau$ and $\mathrm{dist}(o_1, o_2) \leq \epsilon.(j - i)$ and there is no occurrence $o_3 = (k, f_3)$ with $i < k < j$ and $\mathrm{dist}(o_1, o_3) \leq \epsilon.(k - i)$.
- If P has more than one face, then there is an edge from $o_1 = (i, f_1)$ to $o_2 = (j, f_2)$ if there is an edge $(p(o_1), p(o_2))$ in the occurrences graph of $p(P)$ and $\mathrm{dist}(o_1, o_2) \leq \epsilon.(j - i)$.

A *spatio-temporal pattern* S *based on* P is a connected component of the occurrence graph of P.

This definition is such that the occurrence graph of a pattern P is always a subgraph of the occurrence graph of its parent pattern $p(P)$ (if we identify the node o of the occurrence graph of P with the node $p(o)$ of the occurrence graph of $p(P)$). In

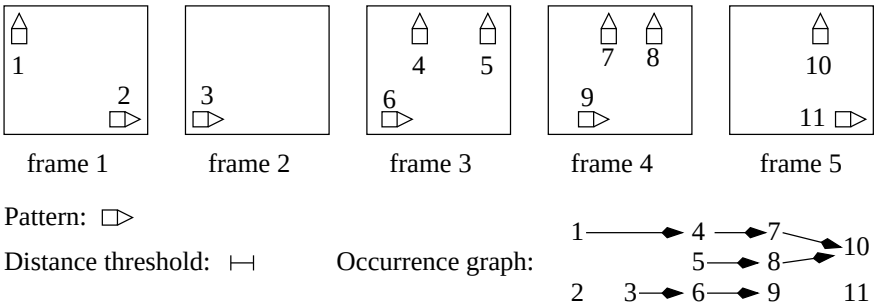

FIGURE 4.3: Occurrences of a pattern and occurrence graph with $\tau = 3$.

practice, P is obtained by extending occurrences of $p(P)$ and removing the ones that do not respect the spatio-temporal constraints. This ensures that the spatio-temporal patterns based on P get "smaller" as the pattern P grows, and this ensures that the frequency of a spatio-temporal pattern freq_{st} has the anti-monotonicity property. Beside, contrary to defintion 4.5, with this definition, the spatial constraint now takes into account the number of frames separating two occurrences. The idea is that if we expect an object to move 10 pixels between frames t and $t+1$, we should expect it to move 10×2 pixels between t and $t+2$, therefore the spatial threshold is multiplied by the number of frames separating the two occurrences. Another improvement is the fact that we now only connect occurrences with the closest occurrences in term of time-stamp. In other words, if two occurrences $o_1(i, f)$ and $o_2(g, i+1)$ are connected, no occurrence $o_3(h, j)$ with $j > i+1$ can be connected to o_1, even if the spatio-temporal constraints are met. This reduces the number of edges in the occurrence graph by removing redundant transitivity edges without breaking any connected component.

Fig. 4.3 shows 11 occurrences of a pattern P in a video with five frames for $\tau = 3$ and with $\text{freq}(P) = 5$ (since it occurres in all 5 frames). Since occurrences 1 and 4 are close to each other, i.e., their spatial distance is lower than $2 \times \epsilon$ and their temporal distance is $2 \leq \tau$, there is an edge $(1, 4)$ in the occurrence graph of P. Conversely, the edges $(3, 5)$ or $(2, 11)$ do not exist in the occurrence graph, as the spatial distance between 3 and 5 or the temporal distance between 2 and 11 are too large. There are 4 spatio-temporal patterns based on P: $S_1 = \{1, 4, 5, 7, 8, 10\}$, $S_2 = \{3, 6, 9\}$, $S_3 = \{2\}$ and $S_4 = \{11\}$. The frequencies of these patterns are: $\text{freq}_{st}(S_1) = 4$, $\text{freq}_{st}(S_2) = 3$, and $\text{freq}_{st}(S_3) = \text{freq}_{st}(S_4) = 1$.

Proposition 4.12. *Given a pattern P with more than one face, and given a spatio-temporal pattern S based on P, there is a spatio-temporal pattern S' based on the parent $p(P)$ of P with a larger freq_{st}, i.e., $\text{freq}_{st}(S) \leq \text{freq}_{st}(S')$.*

Proof. Let two occurrences o and o' of a pattern P. If this pattern has more than one face, it has a parent pattern $p(P)$ from which it has been extended. According

Algorithm variant	Occurrence graph definition used	Frequency measure used	Constraints enforced during mining
PLAGRAM	def 4.5	freq	none
DYPLAGRAM		freq_{seq}	temporal
DYPLAGRAM_ST	def 4.11	freq_{st}	spatio-temporal

TABLE 4.1: Major differences between PLAGRAM, DYPLAGRAM and DYPLAGRAM_ST

to definition 4.11, there must be an edge between $p(o)$ and $p(o')$ in the occurrence graph of $p(P)$ for o and o' to be connected, even if o and o' meet the spatio-temporal constraints. Therefore extending a pattern can only remove edges from its occurrence graph. This means that any connected component in the occurrence graph of P is a subgraph of a connected component in $p(P)$. Therefore for any spatio-temporal pattern S based on P there is a pattern S' based on $p(P)$ that covers at least the same number of frames and has a larger or equal freq_{st} □

This proposition shows that, given a minimum threshold σ_{st} on freq_{st}, if a pattern does not have a frequent spatio-temporal pattern then any super-pattern does not either. This allows us to prune the search space of candidate patterns.

4.2.3 Problem Definition

Given dynamic graph $\mathcal{D}$, a frequency threshold σ_{st}, a spatial threshold ϵ and a time threshold τ, the problem is to compute all spatio-temporal patterns of $\mathcal{D}$ with freq_{st} greater than σ_{st}.

4.3 Mining Spatio-Temporal Patterns

DYPLAGRAM_ST takes advantage of the new definition of an occurrence graph (definition 4.11) and can use freq_{st} to mine spatio-temporal patterns directly. PLAGRAM and DYPLAGRAM do not use freq_{st}, instead they respectively use freq and freq_{seq}. Therefore, to solve the problem defined in Section 4.2.3 with those two algorithms, the idea is to first mine for all frequent graph patterns (using either freq or freq_{seq}) and then, in a post-processing step, construct the occurrence graph of each frequent pattern to compute the spatio-temporal patterns, as described in Definition 4.5. The key differences between the three variants of our algorithm are summarized in table 4.1.

The rest of this section will give the details concerning the extension strategy used by our algorithms, the canonical codes used to avoid processing several times

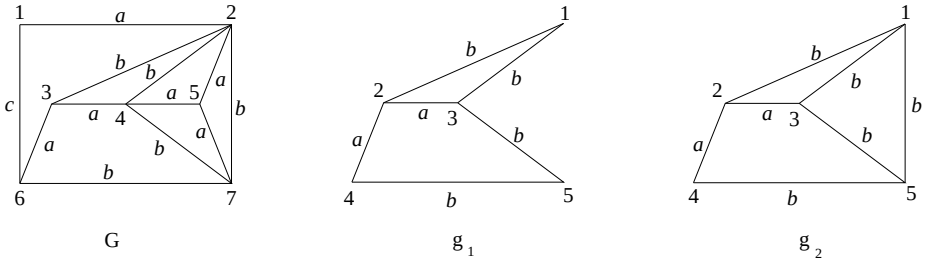

FIGURE 4.4: Plane graphs. The edge labels are in $\{a, b, c\}$ and we assume that all node labels are equal to a (not represented). Graph g_1 is a plane subgraph of G while g_2 is not.

the same subgraph and the strategy to explore the search space. Then we give the pseudo-codes for the three variants of our approach.

4.3.1 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 4.13 (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. 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 $u \neq v$). This restriction is related to that of GSPAN, where a graph is extended by the addition of a single edge, and only to nodes of the rightmost path of the depth-first search tree. 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).

In Figure 4.4, 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 3.3.2).

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, DYPLAGRAM and DYPLAGRAM_ST generate fewer extensions than GSPAN, as we show in Section 4.4. GSPAN extends graph patterns edge by edge, and several extensions may be generated from one node.

4.3.2 Graph Codes

To avoid multiple generations of the same pattern, the graphs are represented by canonical codes. Therefore, to find the frequent patterns, our algorithms explore a code search space. Here, we define these new codes. Next, we present important properties of the 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 $(i, j, L(i), L(i, j), L(j))$, 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 4.14 (Valid code for a plane graph)**.**

- If $g = (V, N, F, f_e, L)$ 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 $(1, 2, L(1), L(1, 2), L(2)).(2, 3, \ldots)$, $(3, 4, \ldots) \ldots, (n{-}1, n, \ldots).(n, 1, \ldots)$. 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 4.14 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.

Table 4.2 shows four valid codes of graph g_1 in Figure 4.4 (among seven valid codes). Figure 4.5 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

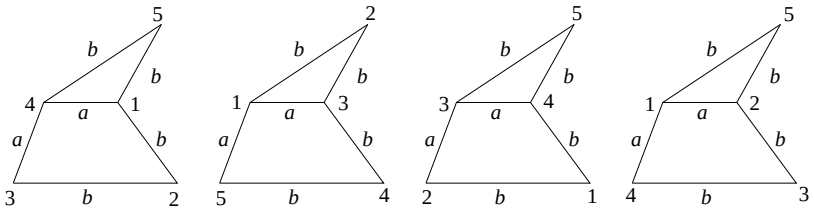

FIGURE 4.5: Four copies of g_1 of Figure 4.4 with node indices corresponding, respectively, to the codes α, β, γ, and δ in Table 4.2.

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 4.2: Four valid codes for graph g_1.

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.

4.3.3 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.

An example tree rooted at code α (of Table 4.2) is represented in Figure 4.6. 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 4.6 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 $(i, j, L(i), L(i, j), L(j)) < (x, y, L(x), L(x, y), L(y))$ if $i < x$ or ($i = x$ and $j < y$) or

($i = x$ and $j = y$ and $L(i) < L(x)$), 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 4.15 (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.

In Figure 4.5, 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.

Theorem 4.16. *In the code search tree, if a code is not canonical, then neither are its descendants.*

Proof. 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. □

In Figure 4.6, $\alpha.A.D$ 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.

4.3.4 Algorithms

This section first discusses the pseudo-codes of PLAGRAM and DYPLAGRAM since those two variants are very similar, then gives the pseudo-code of DYPLAGRAM_ST.

4.3.4.1 Pseudo-codes of PLAGRAM and DYPLAGRAM

The pseudo-codes of PLAGRAM and DYPLAGRAM are shown, respectively, in Figures 4.7 and 4.8.

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. As for GSPAN, our algorithms perform 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

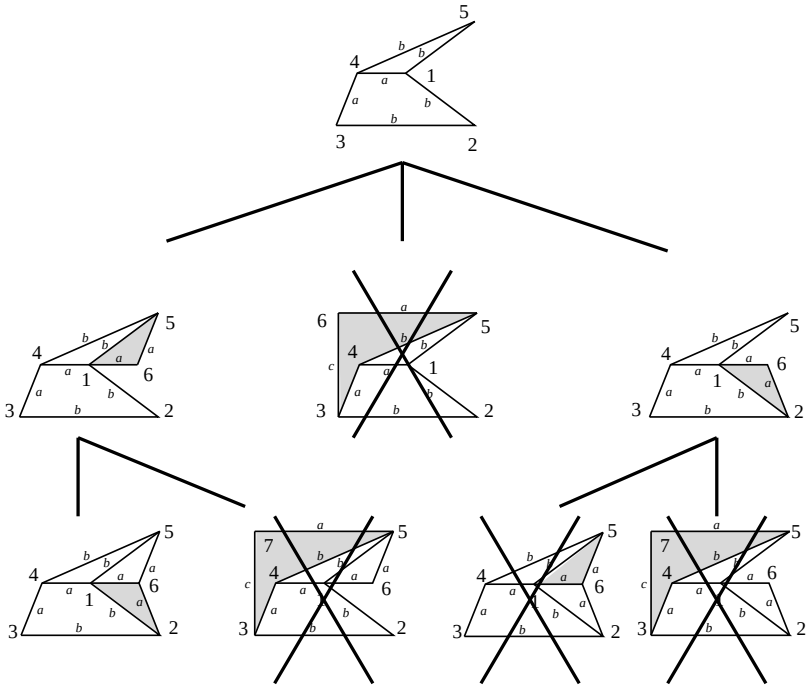

FIGURE 4.6: Part of the code tree starting from code α of Table 4.2. For each pattern, the gray face corresponds to the last added extension. The extension codes are A, ..., G (the complete code of the last line leftmost pattern is thus $\alpha.A.D$). The crossed codes are pruned since they are not canonical.

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,$\mathcal{D}$, σ)
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.E is canonical then
8               output(P.E)
9               mine(P.E,D,σ)
```

FIGURE 4.7: 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 $\text{freq}_{seq}(P) > \sigma$.

```
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.E is canonical then
8               output(P.E)
9               mine(P.E,D,σ,τ)
```

Figure 4.8: Algorithm DyPlagram.

with frequent edges. In both algorithms, the function *mine* 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 makes 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 4.7). 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(*Damiand et al.* (2009)), 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 occurrence is, in the worst case, $O(m.m_i)$.

The 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 4.17). We show in the experimental section that this complexity improvement over gSpan is visible in the measured matching times.

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 step.

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, *Damiand et al.* (2009) for a quadratic algorithm). The simplest algorithm is to enumerate every possible code for a graph to test if one particular code is canonical (with an exponential complexity). 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 quadratic, experimental evaluations show that the canonical tests are not the bottleneck of our algorithms.

Theorem 4.17 (Complexity)**.** *The total complexity of the 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 4.18 (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. 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 4.16 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. □

Actually, the algorithms output codes and not graphs. However, since a code is a list of edges, it is easy to reconstruct a graph from its code.

Once patterns have been extracted using either Plagram or DyPlagram, the occurrence graph can be constructed in a post-processing step to generate the spatio-temporal patterns. This is done by connecting occurrences of the same pattern with

Input: List of occurrences of P, frequency (freq_{st}) threshold σ, time threshold τ, and spatial threshold ϵ.
Output: frequent spatio-temporal 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 spatio-temporal pattern`
8 Output the frequent connected components.

FIGURE 4.9: Generation of spatio-temporal patterns.

an edge if they respect the spatio-temporal constraints, and then computing the connected components of the occurrence graph.

4.3.4.2 Post-Processing Generation of Spatio-Temporal Patterns

PLAGRAM and DYPLAGRAM respectively use freq and freq_{seq} instead of freq_{st} and do not build the occurrence graph during the mining phase. Nonetheless, to generate the spatio-temporal patterns from the frequent patterns returned by both algorithms, the occurrence graph needs to be built in a post processing phase. When those two variants 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, and k is the index of $G_k \in \mathcal{D}$ where this occurrence appear. From this list, the algorithm of Figure 4.9 computes the spatio-temporal patterns based on P as follows: first, it builds the occurrence graph of pattern P with respect to ϵ and τ, as defined in Definition 4.11 (lines 1-6). Given an occurrence (x, y, k), the algorithm computes its distance with 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 spatio-temporal patterns based on a pattern P is $O(\tau.\max_i(m_i).\sum_i m_i)$.

4.3.4.3 Pseudo-code of DyPlagram_st

Given a frequency threshold σ (also called minimum support), a minimum threshold σ_{st} for freq_{st} a spatial threshold ϵ and a temporal threshold τ, the proposed algorithm DyPlagram_st computes all spatio-temporal patterns with $\text{freq}_{st} \geq \sigma_{st}$ based on patterns with $\text{freq} \geq \sigma$ (the thresholds ϵ and τ are used in the construction of the occurrence graph, see Def. 4.11).

With the new distance used for freq_{st}, the frequency constraint is now anti-monotonic (see definition 4.8), therefore we can use it in the DyPlagram_st algorithm directly during the mining phase. However, this frequency is not defined on patterns but on spatio-temporal patterns. We must therefore also build the occurrence graph and the spatio-temporal patterns in the algorithm.

As its predecessors, DyPlagram_st uses canonical codes to represents patterns and extensions. This allows us to efficiently enumerate only the so called valid extensions of a pattern. Informally, a valid extension of a pattern is an extension that lead to a pattern not already considered by the algorithm. This is a very efficient way to avoid considering several times the same pattern.

As can be seen in the pseudo code of the DyPlagram_st algorithm in Figure 4.10, first all frequent one face patterns are built and then the recursive function `mine` is called for all of them.

Lines 1, 6, 7, 8, 9, 10, and 11 of the algorithm in Figure 4.10 were not in DyPlagram. Thanks to Prop. 4.12, this algorithm is correct and output exactly the spatio-temporal patterns whose freq_{st} is above the user defined threshold σ.

4.4 Experiments

We now present the computational results obtained by our proposed algorithms Plagram, DyPlagram and DyPlagram_st. 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, in section 4.4.2, a comparison between Plagram and gSpan, for which we gave details in chapter 3. Section 4.4.3 presents a series of experiments aimed at evaluating the impact of enforcing the temporal and spatio-temporal constraints during the mining phase on the efficiency of the algorithms. In summary, the conducted experiments aimed to answer three main questions:

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

Algorithm: DyPlagram_st$(\mathcal{D}, \sigma, \sigma_{st}, \tau, \epsilon)$
Input: graph database $\mathcal{D}$, frequency threshold σ, spatio-temporal frequency threshold σ_{st}, time threshold τ and spatial threshold ϵ.
Output: spatio-temporal patterns S in $\mathcal{D}$ such that $\text{freq}_{st}(S) > \sigma_s t$ and $\text{freq}_{seq}(P) > \sigma$ with P the pattern on which S is based.

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

mine$(P, \mathcal{D}, \sigma, \sigma_{st}, \tau, \epsilon)$

```
1   occurrences_graph(P) = empty_graph
2   LE = ∅    //list of extensions of P
3   for all graph G_i ∈ D do
4       for all occurrences f of P in G_i do
5           LE = LE ∪ build_extensions(P, G_i, f)
6           Add this occurrence to occurrences_graph(P)
7   Computes the edges of occurrences_graph(P) (using ε and τ)
8   Computes all spatio-temporal patterns based on P
9   for each spatio-temporal pattern S based on P do
10      if freq_st(S) ≥ σ_st then output(S)
11  if there is no frequent spatio-temporal pattern then return
12  else
13  for all extensions E in LE do
14      if freq_seq(E) > σ then
15          if P.E is canonical then
16              mine(P.E, σ, σ_st, τ, ε, D)
17      return
```

FIGURE 4.10: DyPlagram_st algorithm

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

3. What impact the spatio-temporal constraints have on the efficiency of the mining phase ?

For GSPAN, we asked the authors of *Bringmann and Nijssen* (2008) 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.

Before further discussing the experiments, we first describe the video datasets we used.

4.4.1 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 (X-Wings) 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 spatio-temporal patterns can be used to represent the trajectory of the X-Wings individually, as reported in Section 5.4).

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 bins containing the same number of regions, 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. Each continuous frontier between two regions is represented by one edge. On average, each frame led to a graph 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.

Contrarily 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 4.11.

4.4.2 A comparison of PLAGRAM and GSPAN

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

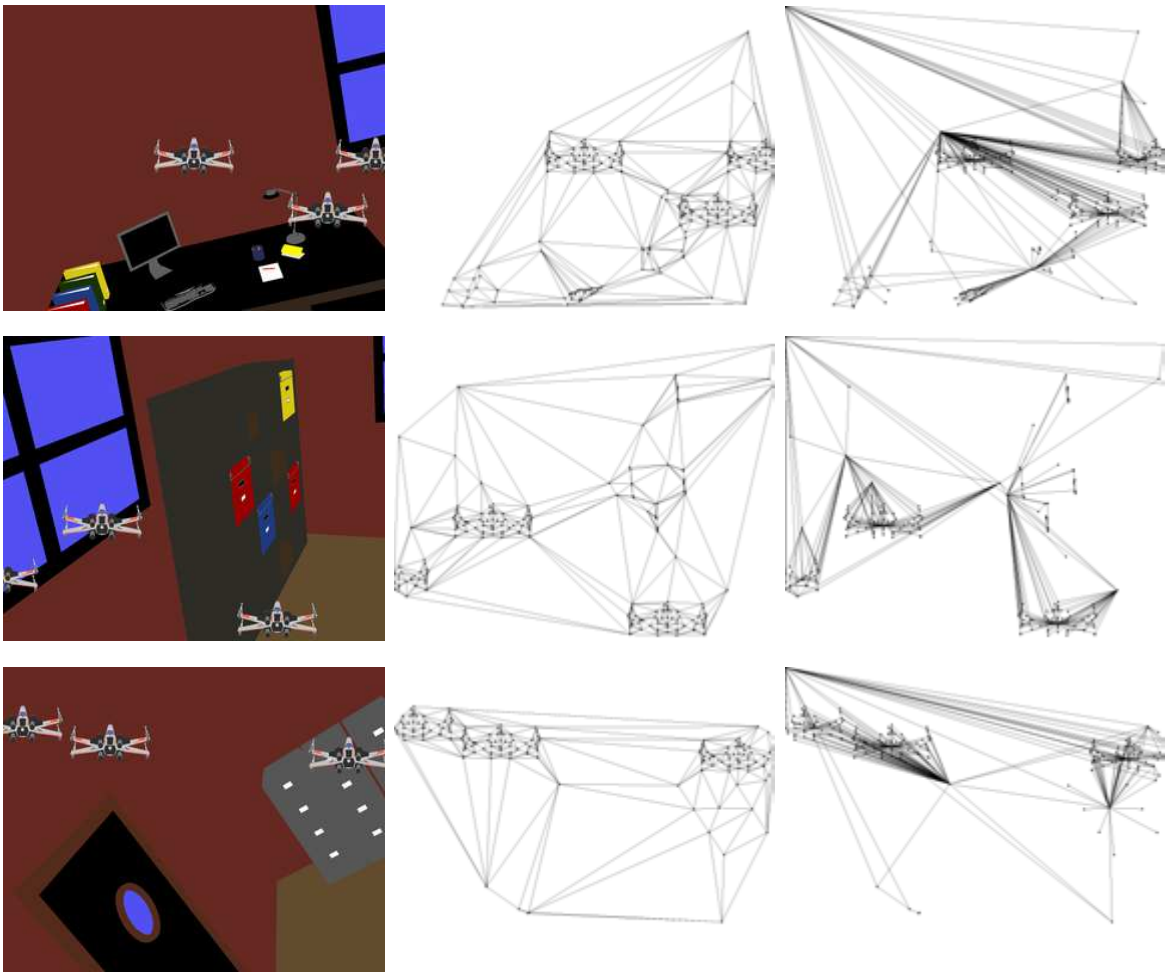

FIGURE 4.11: Example video frames (left) along with their corresponding triangulated (middle) and RAG (right) representations. In the latter, the upper-left node represents the outer region.

- The ratio of the total pattern matching step time to the total size of the matched 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 4.12 and 4.13 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 approximately 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 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 edges) than those generated by Plagram. If the complexity of the subgraph isomorphism test of Plagram was the same as that of gSpan (i.e., exponential in the size of the graph), 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.

RAG dataset As shown in Figure 4.13, 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.

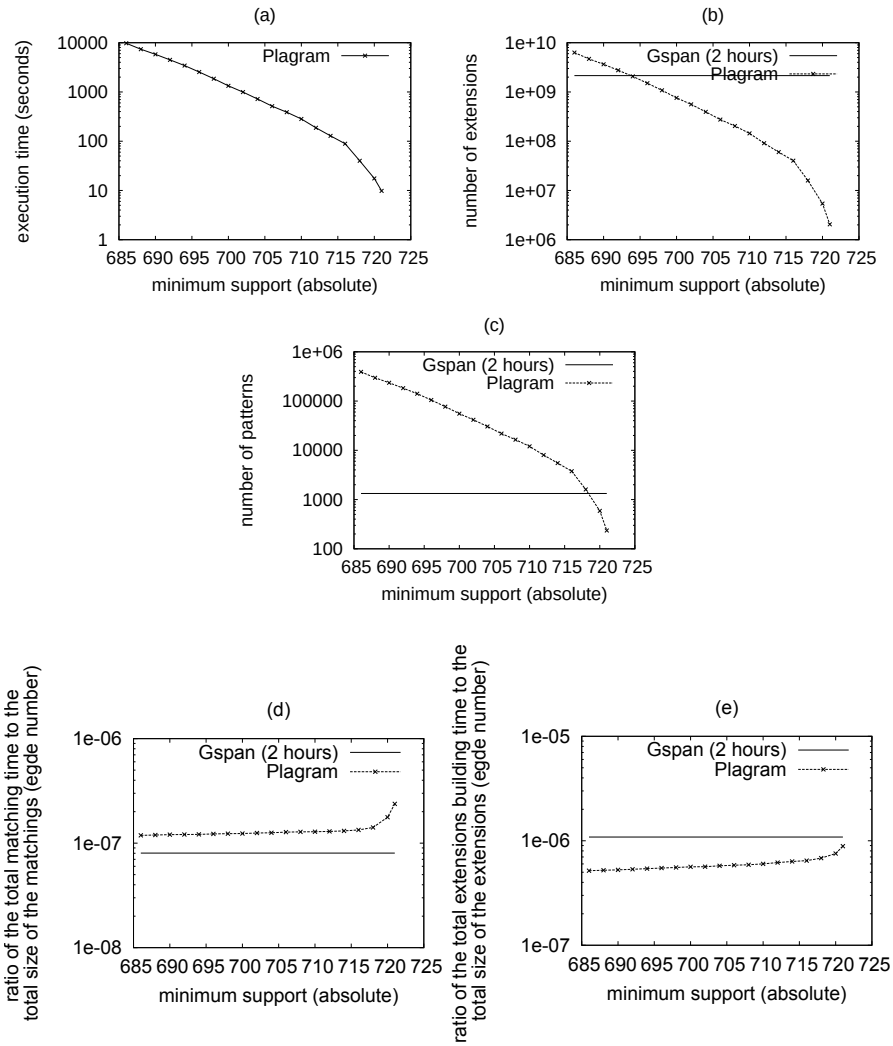

FIGURE 4.12: Efficiency of PLAGRAM and the 2-hour execution of GSPAN on the *Triangulated* dataset.

Step Times We also measured the relative times of the main steps of the algorithms PLAGRAM and GSPAN2. 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 varied from 40% to 60% of the total execution time. The pattern matching step was around 20%, while the canonical-test step was almost always less than 5%. Figure 4.14 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 it produces

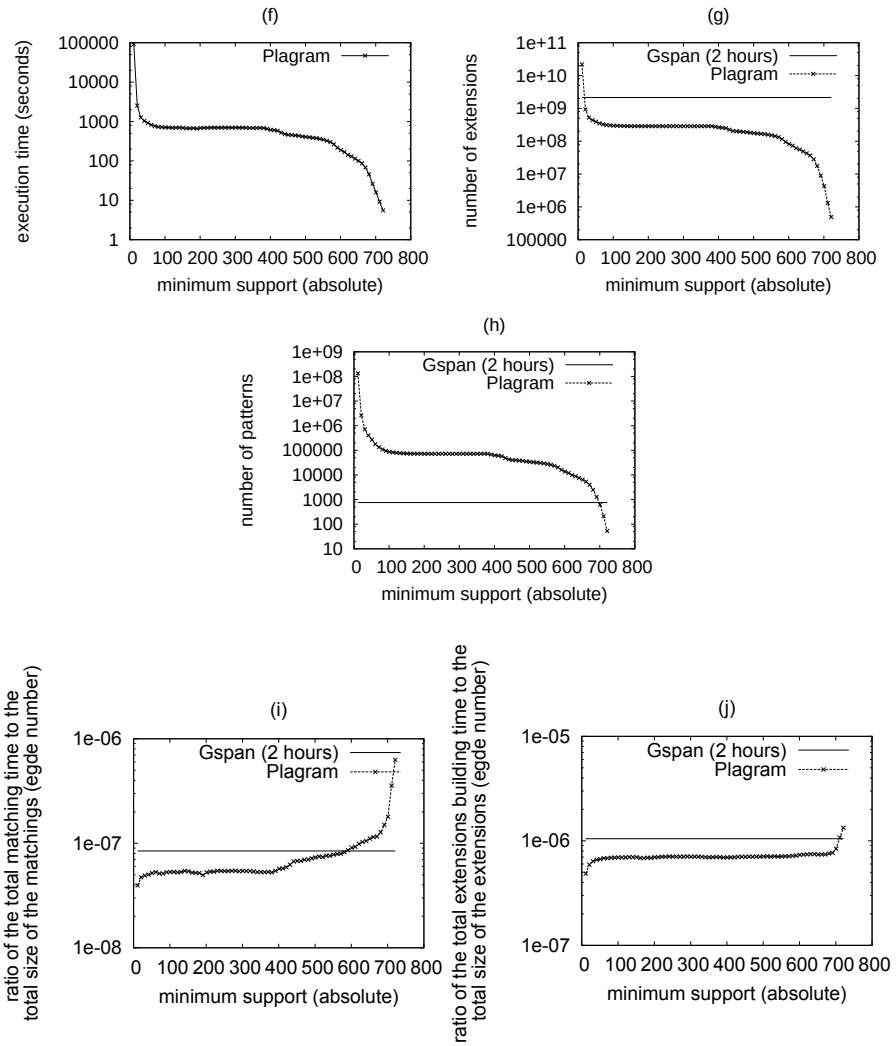

FIGURE 4.13: Efficiency of PLAGRAM and the 2-hour execution of GSPAN on the *RAG* dataset.

rather than only the lower complexity of pattern matching as one could expect.

Scalability Considering the scalability of the proposed algorithm, we evaluated the performance of PLAGRAM with respect to the number of node labels. Several datasets were constructed by varying the number of node labels, from 4 to 16, on the *Triangulated* and *RAG* datasets. As predictable, the total execution times decreased in inverse proportion to the number of labels on both datasets.

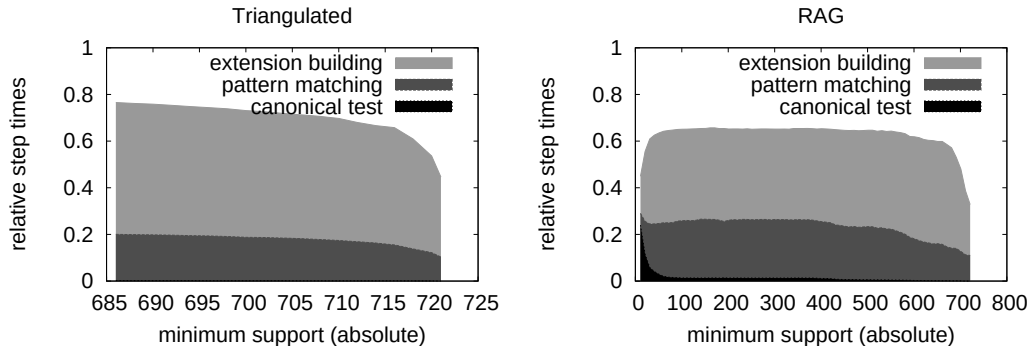

FIGURE 4.14: Relative step times of PLAGRAM. Black region: canonical-test time, dark gray region: matching time, and light gray region: extension building time (the unfilled space at the top-right of the graphs corresponds to other steps of the algorithm, e.g., I/O operations).

4.4.3 Impact of the Spatio-Temporal Constraints on the Efficiency

We first evaluate the consequences of the temporal constraint alone by comparing the efficiency of PLAGRAM and DYPLAGRAM. We then compare DYPLAGRAM with DYPLAGRAM_ST to assess the impact of additionally enforcing the spatial constraint during the mining phase.

4.4.3.1 Temporal Constraint Only: PLAGRAM vs DYPLAGRAM

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 4.15 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.

The next series of experiments investigate the evolution of the execution time and number of occurrences for τ varying from 1 to 50. In those experiments we set the frequency threshold to $\sigma = 100$ and the spatial threshold to $\epsilon = 20$ (only for DYPLAGRAM_ST). Figure 4.16 presents the results we obtained for the PLAGRAM, DYPLAGRAM and DYPLAGRAM_ST algorithms. Note that, since PLAGRAM does not use the temporal constraint, the results for this algorithm remain constant for any value of τ. As we can see, the temporal constraint only has influence for lower values (below 10), above that the execution time and number of occurrences for PLAGRAM

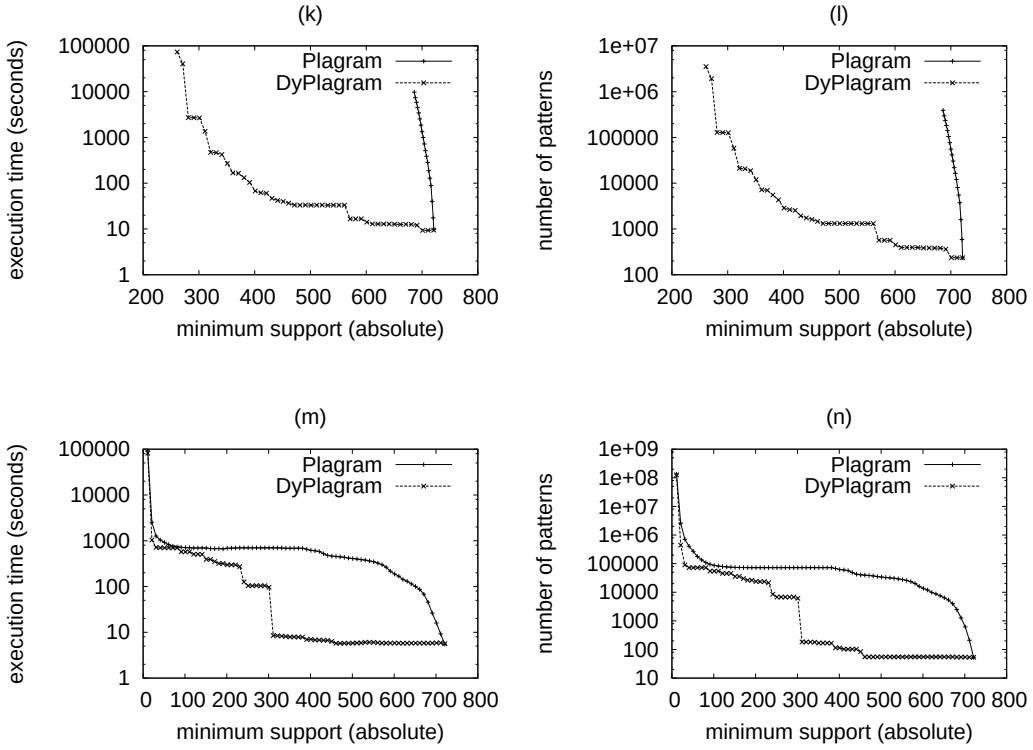

FIGURE 4.15: Efficiency of DYPLAGRAM and PLAGRAM on the *Triangulated* ((k) and (l)) and RAG ((m) and (n)) datasets.

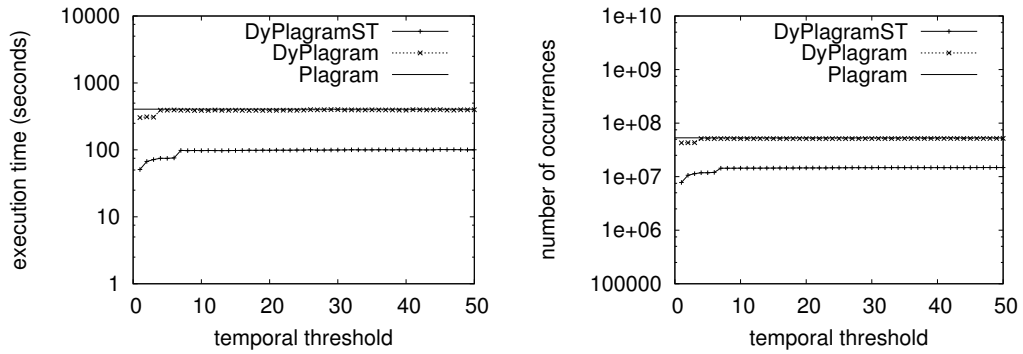

FIGURE 4.16: Efficiency of PLAGRAM, DYPLAGRAM and DYPLAGRAM_ST for different temporal threshold values on the RAG dataset with the computational time on the left and the number of occurrences of frequent (spatio-temporal) patterns discovered on the right. $\sigma = 100$

and DYPLAGRAM is almost equivalent. This is due to the fact that RAGs produced with the synthetic video are very stable, hence there is no large temporal gap between the occurrences of the same pattern unless the objects are occluded or out of the field of view.

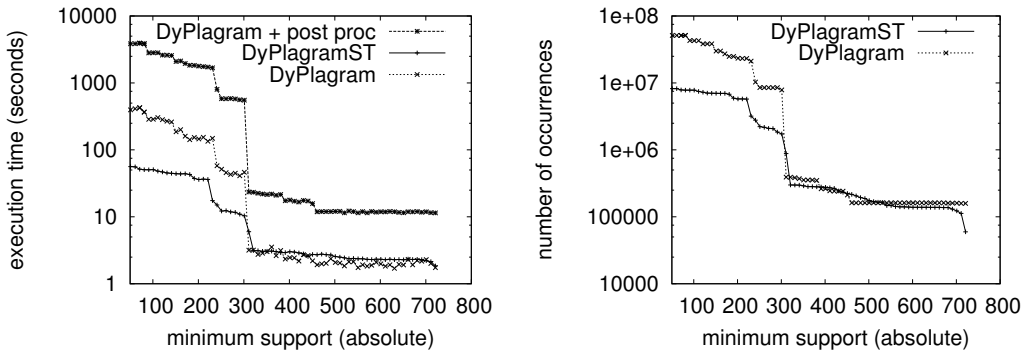

FIGURE 4.17: Efficiency of DYPLAGRAM and DYPLAGRAM_ST on the RAG dataset with the computational time on the left and the number of occurrences of frequent spatio-temporal patterns discovered on the right.

4.4.3.2 Spatial and Temporal Constraints: DYPLAGRAM vs DYPLAGRAM_ST

Here we evaluate how the spatial and temporal constraints combined influence the efficiency of the mining phase. Figure 4.17 shows efficiency results comparing DYPLAGRAM and DYPLAGRAM_ST on the RAG dataset. Those experiments have been conducted with $\text{minfreq}_{st} = 50$, a temporal threshold $\tau = 1$ and a spatial threshold $\epsilon = 20$. As expected, pushing the spatial constraints during the mining step allows us to generate less occurrences. This speeds up the mining process, allowing DYPLAGRAM_ST to extract patterns with low minimum support more efficiently than DYPLAGRAM especially for support < 350.

Figure 4.18 shows the evolution of the execution time and number of occurrences discovered for PLAGRAM, DYPLAGRAM and DYPLAGRAM_ST for a minimum support $\sigma = 100$ and for ϵ varying from 5 to 100. The temporal threshold is set to $\tau = 25$ for both DYPLAGRAM and DYPLAGRAM_ST. Since PLAGRAM and DYPLAGRAM do not use the spatial constraint their execution time and number of occurrences discovered remains constant for all values of ϵ. Note that DYPLAGRAM discovers a little less occurrences and takes a little less time than PLAGRAM (406 seconds for PLAGRAM and 396 seconds for DYPLAGRAM) but for clarity they are both represented by the same line in Figure 4.18. As we can see, even for high spatial threshold values, the spatial constraint improves the efficiency of the algorithm.

4.5 Conclusions

We presented a frequent plane graph mining algorithm called PLAGRAM and its two extensions DYPLAGRAM and DYPLAGRAM_ST to mine spatio-temporal pat-

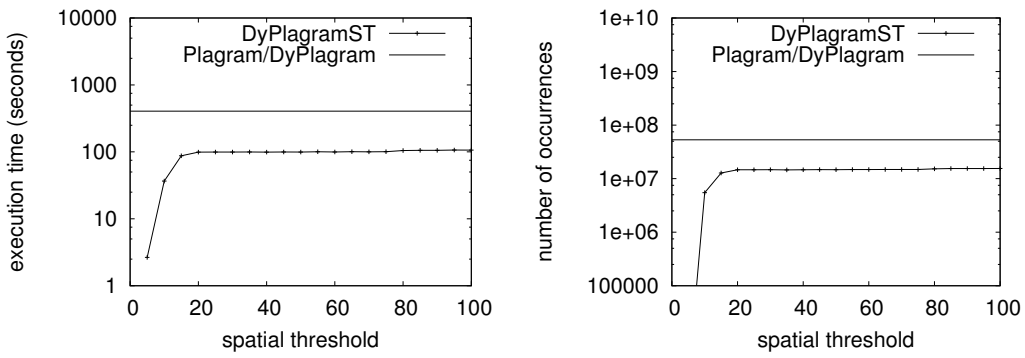

FIGURE 4.18: Efficiency of PLAGRAM, DYPLAGRAM and DYPLAGRAM_ST for different spatial threshold values on the RAG dataset with the computational time on the left and the number of occurrences of frequent (spatio-temporal) patterns discovered on the right. $\sigma = 100$ and $\tau = 25$ (for DYPLAGRAM and DYPLAGRAM_ST

terns. Our conducted experiments showed that PLAGRAM (and, consequently, DYPLAGRAM and DYPLAGRAM_ST) 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 proved that our algorithm benifits a lot from enforcing spatial and temporal constraints during the mining process. Indeed, by permitting to directly mine spatio-temporal constraints the algorithm generates less occurrences resulting in a lower processing time. Our experiments also show that the RAG datasets could be mined more efficiently than the triangulated one.

Overall PLAGRAM and its variants have proved to be efficient at mining 2-connected plane graph databases. The next chapter will present our contributions in the field of object tracking. We will discuss how meaningful the (spatio-temporal) patterns are in the context of video tracking and how they could be used to follow objects in videos.

CHAPTER 5

Tracking Objects in Videos Using Spatio-Temporal Patterns

5.1 Introduction

In the previous chapter we showed how the PLAGRAM algorithm could be used to efficiently extract frequent plane graph patterns from a database of plane graphs. We also demonstrated that using spatio-temporal constraints could further increase the efficiency of the mining process. This chapter studies how meaningful the extracted patterns are in the context of object tracking.

The appearance of moving objects changes over time and so does their graph representation. Besides, the instability of the segmentation process also results in different graph representations of the same object from one frame to another. Consequently, it is very unlikely to be able to follow an object using one single spatio-temporal pattern. Instead we have to find a way of combining several patterns to obtain complete tracks of the interesting objects. To do so, we use two different strategies described in Section 5.2. Both of them use the spatio-temporal patterns discovered by the DYPLAGRAM_ST algorithm described in Chapter 4, which constructs the occurrence graph of each pattern during the mining phase.

In a first strategy, we build a graph which concatenates the occurrence graphs of all frequent patterns. We add edges to this graph connecting the occurrences of different patterns that are similar. Then, we look for paths called spatio-temporal paths in this global occurrence graph. With the first method, the user needs to select a region of interest in the first frame of the video by drawing a bounding box. We then select the shortest spatio-temporal path (with respect to some weights on the edges) that starts from one of the occurrences contained in the selected area.

The second strategy exploits a similarity measure between the trajectories of the spatio-temporal patterns to group them into clusters representing the objects of the

scene. The clusters are obtained with a hierarchical clustering algorithm. With this approach, we cut the hierarchy to obtain the best clusters (i.e., clusters that best match the main objects) automatically.

In Section 5.3, we describe the datasets used in our experiments. Section 5.4 reports the experiments we conducted to first assess the meaningfulness (and thus the potential) of the spatio-temporal patterns discovered by PLAGRAM, DYPLAGRAM, and DYPLAGRAM_ST (which are not yet spatio-temporal paths nor clusters). An experimental study of both tracking strategies is presented in Section 5.5. Finally we conclude in Section 5.6 and give two industrial applications of our approach in Section 5.7.

5.2 Tracking with Patterns

We first describe a method that uses spatio-temporal paths in a global occurrence graph to track object. Then we detail an alternative hierarchical clustering approach.

5.2.1 Spatio Temporal Path

When tracking an object in a real video, we cannot expect the object to be represented by the same graph pattern during the whole video (e.g., due to changes in view point or instability of the segmentation). Thus, if we want to track it, we need to use several spatio-temporal patterns. To do so, we propose to merge the occurrence graph of each pattern into a global occurrence graph, and add similarity edges to it so that similar occurrences of different patterns that appear in the same frame are connected. In this way, it is possible for a path in the occurrence graph, called a spatio-temporal path, to "jump" from a spatio-temporal pattern to another one that has similar occurrences. The similarity between two occurrences is derived from their overlap in term of nodes. It can be efficiently computed by counting how many regions they have in common. Indeed, since the similarity edges connect only occurrences that appear in the same frame, their set of common nodes can be obtained by computing the intersection between their node list.

Definition 5.1 (Similarity of two occurrences)**.** Let $o = (i, f)$ and $o' = (i, f')$ be two occurrences of two different patterns $P = (V, E, F, f_e, L)$ and $P' = (V', E', F', f'_e, L')$ with f, f' the mappings of the nodes of the patterns to the graph of the ith frame. The similarity between these occurrences is defined as $sim(o, o') = \frac{|f(V) \cap f'(V')|}{|f(V)|}$.

This similarity is used to weight the similarity edges of the global occurrence graph in combination with the spatial distance between occurrences. Note that this

similarity is not symmetric in order to favor the transition from smaller patterns to bigger ones. The other edges of the global occurrence graph, i.e., the ones connecting occurrences of the same pattern if they are close in space and time, are weighted according to the temporal distance between the occurrences.

Definition 5.2 (Global occurrence graph)**.** Given a set of patterns $\mathcal{P}$, temporal and spatial thresholds τ and ε, a similarity threshold μ, the global occurrence graph is a weighted oriented graph: its node set is $V = \cup_{P \in \mathcal{P}} \text{Occ}(P)$ and its edge set is $E = E_{\mathcal{P}} \cup E_{sim}$ where :

- $E_{\mathcal{P}}$ is the union of the edge sets of all patterns' occurrence graphs. The weight of an edge $((i, f), (i', f'))$ is $w = \frac{(i'-i-1)}{\tau}$.
- $E_{sim} = \{(o, o', w) |\ o = (i, f), o' = (i, f'), sim(o, o') > \mu\}$ is the set of similarity edges with

$$w = \begin{cases} 0 & \text{if } |V| < |V'| \\ \frac{1}{2}(\frac{1-sim(o,o')}{1-\mu} + \frac{d}{\epsilon}) & \text{otherwise.} \end{cases}$$

  where V and V' are the node sets of the patterns corresponding respectively to occurrences o and o', and d is the distance between the barycenters of o and o'.

A *spatio-temporal path* is a path in the global occurrence graph.

In definition 5.2, the edges in $E_{\mathcal{P}}$ are edges between 2 occurrences of the same pattern that are not in the same frame. If these two occurrences are in consecutive frames, the weight is 0 (when $i' = i + 1$) otherwise the weight increases with the number of frames between them (normalized by the temporal threshold τ).

The edges in E_{sim} are *similarity edges* between 2 occurrences of different patterns that are in the same frame and whose similarity is above μ. To favor paths that use large patterns we set the weight of edges going from smaller occurrences to bigger ones to 0. The weight of an edge from an occurrence of a large pattern to a smaller one increases as the similarity decreases and the spatial distance increases. Otherwise the weight on the edges increases as the spatial distance increases and the similarity between the occurrences decreases.

5.2.2 Clusters of Spatio-Temporal Patterns

In this second approach, we want to cluster the spatio-temporal patterns. To do so, we first need to define a distance function between them and then a clustering algorithm that can regroup them in clusters corresponding to interesting objects. One

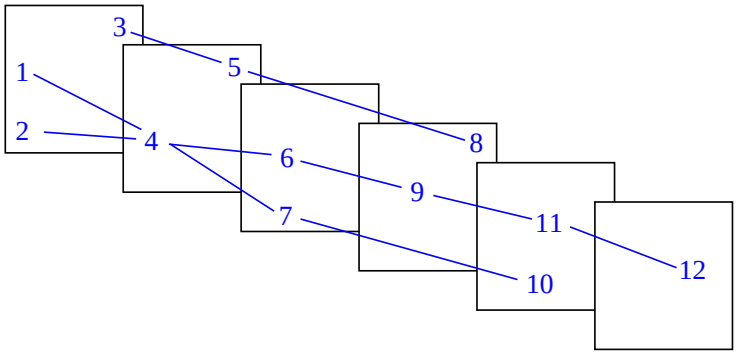

FIGURE 5.1: Example of an occurrence graph for a given pattern P which occurs 12 different times in 6 frames of a given video.

of the main difficulties here is to estimate how many clusters to produce, and how to choose the ones that are more likely to represent an interesting object.

5.2.2.1 Dissimilarity between spatio-temporal patterns

Each spatio-temporal pattern p can be represented as a trajectory $p_{tr} = \{(x_i^p, y_i^p) | f_s^p \leq i \leq f_e^p\}$ with f_s^p and f_e^p respectively the starting and ending frames of p. For each spatio-temporal pattern, the coordinates (x_i^p, y_i^p) of the points of its trajectory are obtained by computing the barycenters of its occurrences in each frame i. For example, in Figure 5.1, we would compute the barycenter of occurrences 1 and 2 for the second spatio-temporal pattern. Since the temporal threshold τ allows spatio-temporal patterns to have gaps in the sequence of their occurrences, the coordinates of the points of the trajectory in those frames are interpolated between the previous and the next known points.

Let A and B be two patterns. Let $f_s = max(f_s^A, f_s^B)$ and $f_e = min(f_e^A, f_e^B)$. The distance between two spatio-temporal patterns is defined as:

$$d(A,B) = \begin{cases} \text{if } f_e - f_s > 0 \\ \text{then } d_{traj}(A,B) * (2 - d_{ov}(A,B)) \\ \text{else } \infty \end{cases}$$

$$\text{where } d_{traj}(A,B) = \sum_{f_s}^{f_e} \frac{\sqrt{(x_i^A - x_i^B)^2 + (y_i^A - y_i^B)^2}}{f_s - f_e + 1}$$

$$\text{and } d_{ov}(A,B) = \frac{f_s - f_e + 1}{min(f_s^A, f_s^B) - max(f_e^A, f_e^B) + 1}$$

In words, if two patterns never belong to the same frames, their distance is infinite. Otherwise, their distance is the normalized (over the number of common frames) sum

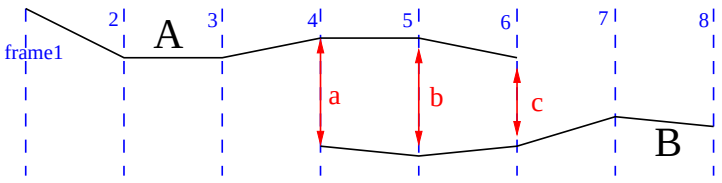

FIGURE 5.2: Example of two overlapping (on 3 frames) spatio-temporal patterns A and B

of the Euclidean distances between the barycenters of the patterns that appear in common frames. We added a penalty between 1 and 2 to take into account the proportion of common frames compared to the span of the union of the two spatio-temporal patterns. For example, in Figure 5.2, the distance between the two patterns is $\frac{(a+b+c)}{6-4+1} * (2 - \frac{3}{8-1+1})$.

5.2.2.2 Clustering algorithm

To cluster our spatio-temporal patterns without knowing in advance the number of interesting clusters, we decided to use a simple hierarchical clustering algorithm (*Anderberg* (1973)) with the distance function previously defined. The main problem of this algorithm is the choice of the criterion to cut the hierarchy of the dendogram without any information a priori about the quality of the resulting clustering. We decided to cut the hierarchy at the level of the creation of the cluster with the highest *lifetime* (*Ana and Jain* (2003)). The *lifetime* of a cluster corresponds to the difference between the similarity at which it has been formed and the similarity at which it is merged with another cluster. However, the lifetime criterion tends to behave badly in the presence of outliers which are fused at the top of the hierarchy and often have the maximum lifetime (the hierarchy is thus cut at a high level with very few clusters). To overcome this drawback, we decided to ignore the 10% first levels of the hierarchy (note that there are i clusters at level i) before computing the lifetime. A visual example of the lifetime is depicted in Figure 5.3. We also tried an other criteria, called the *gain* (*Jung et al.* (2003)), but this criteria tended to cut the hierarchy at the top, even when ignoring the 10% first levels, which resulted in clusters with low precision.

5.2.2.3 Selection of the best clusters in the clustering

Since with our approach, a lot of the spatio-temporal patterns of the background are not part of any precise cluster, the optimal number of clusters is usually much higher than the true number of main objects.

Therefore, after having cut the hierarchy, we still have to decide which clusters

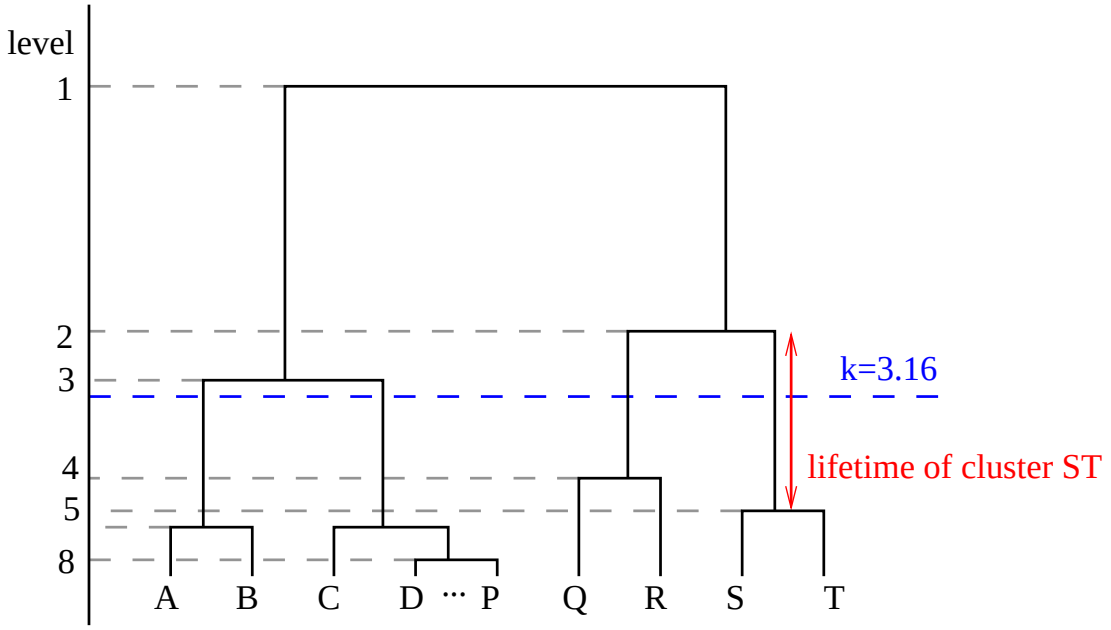

FIGURE 5.3: Example of hierarchy of clusters with the lifetime of cluster ST depicted with a red arrow. The hierarchy is cut at level 5 because cluster ST has the longest lifetime among those below level $k = \sqrt{20/2} = 3.16$.

are the most interesting. The idea is to rank the clusters and only keep the best ranked ones. In the rest of this document, the *size* of a cluster refers to the number of spatio-temporal patterns composing it and the *length* of a cluster refers to the number of frames it covers. More precisely, the length is computed as the difference between the frame number of the first and last frames the cluster has an occurrence in. We tried different strategies to rank the clusters. We first ranked them according to their length only or size only but the strategy of ranking the clusters according to their length first and then according to their size gave better results overall. This third strategy still has problems. In particular, in the case where interesting objects do not appear in all the frames of the video, top ranked clusters do not always represent those interesting objects. Instead they often are small clusters composed of few patterns with low discriminative power that cover all the video but do not represent anything interesting. To deal with this problem we changed our ranking strategy to favor the biggest clusters among the ones that covered the majority of the video. More precisely, we first keep all the clusters with length l such that $l_{max} \geq l \geq l_{max} - 0.1 \times |\mathcal{D}|$, where l_{max} is the length of the longest cluster and $|\mathcal{D}|$ is the number of graphs in the database (i.e., the number of frames in the video). Within the clusters of this length, we select the one (or randomly among the ones) with the highest number of spatio-temporal patterns and then the highest number of occurrences. This cluster is called the *longest*

in the rest of this document.

To decide how many interesting objects should be tracked in the video in a completely unsupervised manner (without selecting them in the first frame), we could either assume that there is only one object, or find among the longest clusters the ones that are sufficiently far from each other. However, in our experiments, we select for each video the n longest clusters, with n being the number of main objects in the video. We then measure their precision and recall with respect to the ground truth.

5.3 Datasets

The benchmark datasets presented in section 2.4 are not entirely satisfactory because most of them focus on video surveillance setups or are composed of videos with too few frames for the mining step to extract meaningful patterns. Note that in these cases, our algorithm could also be used but may perform worse than the optimized dedicated ones. To assess the qualities of our algorithm, we thus introduce our own dataset. We used 4 videos for these experiments. The two first ones are synthetic videos, based on the video presented in section 4.4.1. They allow us to avoid the possible segmentation problems by keeping the true colored regions. The two last ones are real videos. The real videos are first segmented and for all of them we create a region adjacency graph (RAG) (*Chang et al.* (2004)) for all the frames of the video. As our RAGs greatly depend on the segmentation, we tried two types of segmentation. The first segmentation (static) is done independently on each frame using the algorithm[1] presented in *Felzenszwalb and Huttenlocher* (2004). This algorithm has three parameters for which we use the default values. As discussed in section 2.1.2, it favors the merging of small regions which may result in an unstable segmentation when objects are getting close to or moving away from the camera. Indeed, the decision of merging two regions depends on the value of $\tau(R_i) = k/|R_i|$, where k is a parameter, R_i a region and $|R_i|$ its size. The higher the size of the region $|R_i|$ the lower is $\tau(R_i)$ and therefore, the higher are the chances of merging R_i with another region. In order to prevent this behavior, we modified the code of this algorithm to make its second parameter independent from the size of the regions by removing the size of the regions $|R_i|$ form the computation of $\tau(R_i)$. In this way, the decision of merging two regions depends only on their color difference and the magnitude of the parameter k. Figure 5.4 shows examples of RAGs representing a frame of our videos. The second segmentation is the (dynamic) video segmentation

[1] Efficient graph base segmentation source code available here: http://cs.brown.edu/~pff/segment/

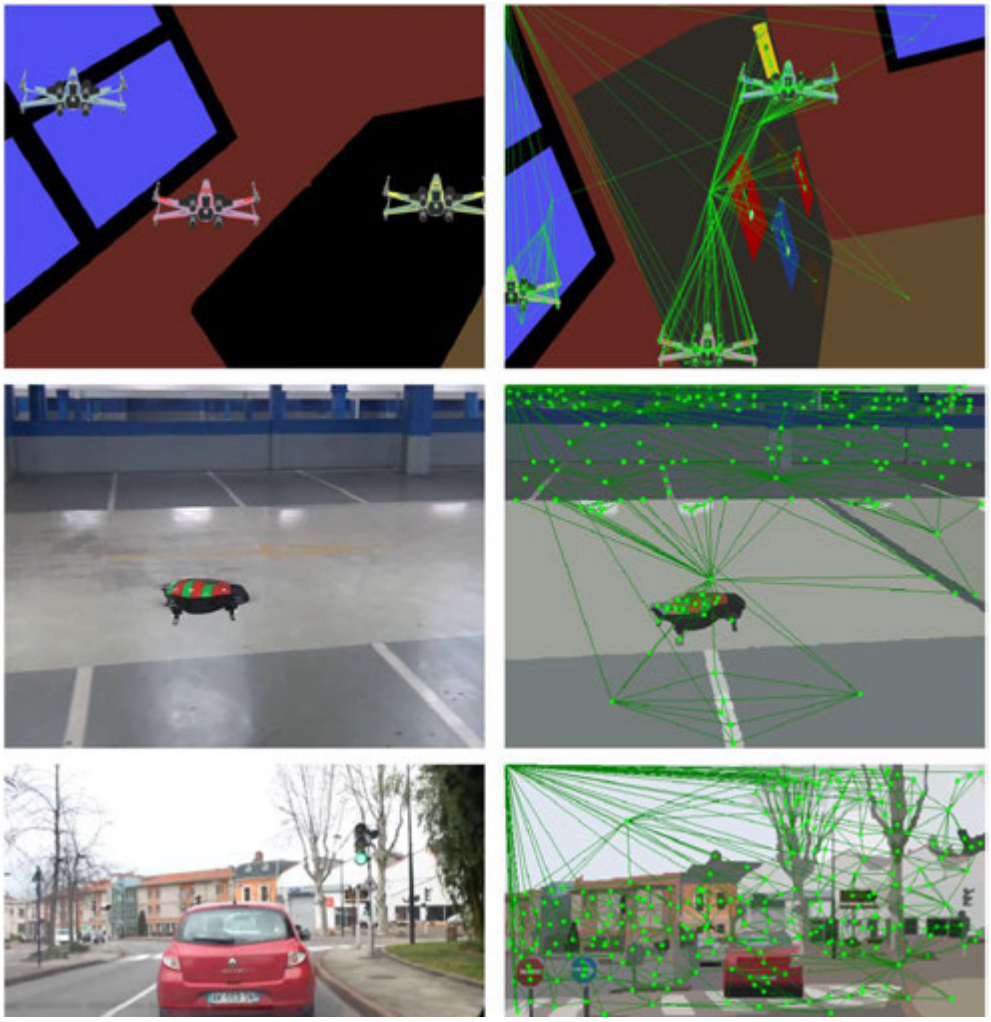

FIGURE 5.4: Example of frames and RAGs obtained from the synthetic videos (top), from the segmented real drone video (middle), and from the segmented car video (bottom).

algorithm[2] presented in *Grundmann et al.* (2010). This algorithm outputs regions that are identified through time, i.e, it provides a correspondence between regions in different frames.

Synthetic videos We used the synthetic video we have already used for the experiments discussed in the chapter 4 on the efficiency of our mining algorithms (see section 4.4.1). In the rest of this document this video is called Anim1. Based on this synthetic video, we produced a second one, called Anim2, that is identical except for the color of the X-Wings that is different for each one of them (cf. top of Figure 5.4). These videos were used to assess whether our approach can deal with scenes involving several objects occluding each others and moving out of the field of view. We also

[2]Video segmentation web service at this address: `http://videosegmentation.com/`

used a simpler video, in which there is only one X-Wing, to assess the meaningfulness of the patterns returned by PLAGRAM. This video has the exact same background as the two other ones. It has 721 frames and on average, each frame led to a graph with 245.2 nodes, with an average degree of 2.23.

Real Videos The first real video (cf. middle of Figure 5.4) is composed of 950 frames, each RAG has on average 194.5 nodes with an average degree of 5.35. This video shows a drone flying across a covered parking lot. This video is simple but the segmentation still suffers from the illumination changes. The second real video (cf. bottom of Figure 5.4) is made of 5619 frames, each RAG has on average 207.5 nodes with an average degree of 5.5. This video is shot from a car while following another car (the main object). In this video the main object goes out of the field of view, its scale changes, the global illumination changes all the time and it is also longer than the other ones which allows us to test the efficiency of our approach. This video has been divided into 3 parts (car1000, car2000, car3000) which correspond to the 1000, 2000 and 3000 first frames of the car video. This has been done since the tracking difficulty gradually increases along the video.

For both videos, we use the same modified segmentation algorithm with standard parameters to segment the images. With these videos, we want to assess whether our approach can deal with changing appearances and with the segmentation inaccuracies.

The labels of the nodes of the graphs of both the synthetic and real videos were obtained by discretization of the size and the average color of the regions. Figure 5.4 presents some frames of the videos along with the corresponding RAGs.

5.4 Meaningfulness of the (Spatio-Temporal) Patterns

To evaluate how meaningful our (spatio-temporal) patterns are, before constructing any spatio-temporal path or clusters, we study whether they can be used to track a given object in a video.

We start by introducing two measures which assess how precisely a (spatio-temporal) pattern p corresponds to a given target object o in the video frames. These measures, also used later on to evaluate our more elaborate tracking strategies, are adaptations of the popular measures *precision* and *recall* as described below:

- **precision**: fraction of the occurrences of p (in the target graphs) for which every node maps to the object o in the corresponding video frames. The intuition behind this measure is to evaluate the *purity* of p, that is, p has the maximum precision if it maps only to the object 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 of p where every node maps to o. Here, the intuition is to evaluate the *completeness* of p. More precisely, the idea is to check whether the occurrences of p map to all occurrences of o in the set of video frames.

Since our algorithms are exhaustive, that is, they mine for all frequent (spatio-temporal) patterns in the graph database without supervision, the mining results may consist of different (spatio-temporal) patterns corresponding to different objects, or even to no specific one (w.r.t. the proposed measures). Therefore, to follow a specific object in the video, the user should be able to select from the entire set of output (spatio-temporal) patterns those that correspond to this object. A basic strategy for this task is the following:

1. First, the user selects a frame area where there exists an object he or she is interested in tracking, that is, the target object. This is done in a user selected frame, referred to here as f, where this object occurs.

2. Afterwards, the user starts the graph mining process by executing either PLAGRAM with a given minimum support or DYPLAGRAM with a given minimum support and time constraint as input, followed by the post-processing step described in section 4.3.4.2 with a spatial constraint. Alternatively, spatio-temporal patterns can be directly extracted using DYPLAGRAM_ST.

3. Next, the (spatio-temporal) patterns that have no occurrences in the user-selected area, in frame f, are discarded. The remaining patterns are considered the target patterns, i.e., those that characterize the target object.

4. Finally, all the occurrences of the target (spatio-temporal) patterns are mapped to the video frames, allowing the user to detect the position of the target object through the video.

5.4.1 Output of PLAGRAM (plane graph patterns)

We first evaluated our strategy on the patterns returned by PLAGRAM (the plane patterns). In those experiments we used the simple video with only one X-Wing. We checked whether it would be possible to follow the X-Wing in this basic video by considering the patterns that matched it in the first frame, i.e., patterns that where inside the user selected area. As might be expected, those patterns had different precision and recall with respect to the X-Wing. Some examples are given in Figure

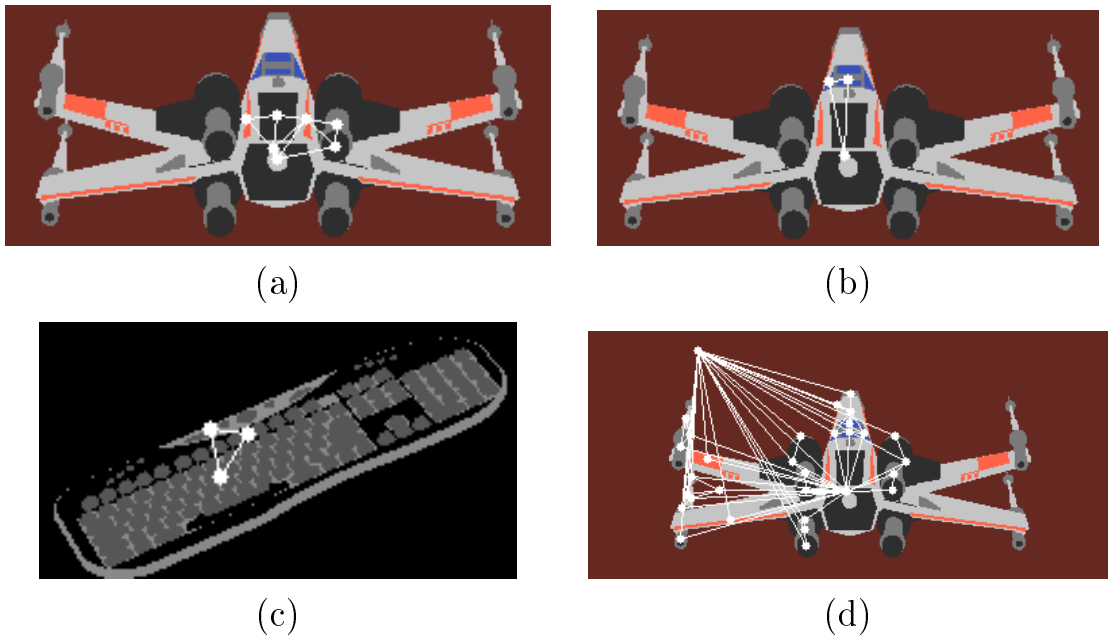

FIGURE 5.5: (a): pattern with 100% precision and recall in the *Triangulated* dataset. (b) & (c): 2 occurrences of the same pattern (the X-Wing and a keyboard, respectively) with 52% precision and 100% recall in the *RAG* dataset. (d): example pattern with 0% precision and recall in the *RAG* dataset.

5.5. In (b) and (c), we show 2 different occurrences of a pattern with 100% support, 52% precision, and 100% recall in the *RAG* dataset. Now, consider the graph in (d). It illustrates an occurrence of a pattern with support of 378 frames in the *RAG* dataset. Note that this occurrence had a node outside of the X-Wing area; this decreased the precision of the corresponding pattern. Indeed, it had 0% precision and recall.

After executing step 3, we got the patterns whose average precision and recall (in percentage) are shown in Table 5.1.

Observe that the selected patterns had, on average, very good quality, making step 4 successful. Considering the *Triangulated* dataset, the average precision increased in inverse proportion to the minimum support, while the average recall decreased with the minimum support. Here, lower minimum support led to bigger patterns with higher precision and lower recall. In the *RAG* dataset, the behavior was different: big patterns had nodes that did not map to the X-Wing. In addition, small patterns with low support did not have good precision nor recall. As a consequence, the average precision and recall decreased with the minimum support.

However, in a less simplistic context (e.g., when multiple identical object are present), the precision and recall of the patterns returned by PLAGRAM drops dramatically. This is due to the fact that, in the video used for these experiments (see

	Triangulated		*RAG*	
Support	precision (%)	recall (%)	precision (%)	recall (%)
721	96.2	99.8	97.1	99.9
711	97.6	98.9	97.2	99.8
701	99.3	97.7	96.5	99.0
691	99.7	96.3	93.9	95.6
681	99.8	95.0	92.8	93.9
671	99.8	93.7	92.5	93.5
661	99.9	92.5	92.5	93.5
651	99.9	91.0	91.8	92.6

TABLE 5.1: Average precision and recall (in percentage) computed for the patterns selected at step 3 of the proposed object tracking strategy.

Figure 4.11), the 3 X-Wings may overlap and two of them can partially go out of the video frames. Besides, as the target X-Wings are identical, some ambiguities may happen w.r.t the target patterns. For example, a target pattern may be very frequent just because it maps to multiple X-Wings and thus appear in almost every frame, but imprecise (i.e., with a low precision) with respect to a given X-Wing o if it maps not only to o, but to different X-Wings through the video. Therefore, to track a given object in our more complex video, the use of spatio-temporal constraints becomes necessary.

5.4.2 Output of DyPlagram and DyPlagram_st

To check whether the defined spatio-temporal patterns can well represent the individual trajectory followed by several similar objects, or the trajectory of objects in real videos, we used the previously described strategy on the video Anim1 that shows 3 identical X-Wings moving in a room and using the size discretization to label the nodes of the RAGs. We also performed experiments on the real video with the drone to assess the meaningfulness of the spatio-temporal patterns when the appearance of the target changes and segmentation errors are introduced.

Experiments on Anim1 For DyPlagram, we first extracted all frequent patterns with a frequency threshold of $\sigma = 721$, corresponding to the number of frames in the video, and a temporal constraint $\tau = 1$ to focus on patterns that appear in every frame (note that some occurrences of the same pattern may correspond to different spatio-temporal patterns). Then, we post-processed them to generate spatio-temporal

	DyPlagram with post-processing			DyPlagram_st		
	Precision(%)	Recall(%)	# ST patterns	Precision(%)	Recall(%)	# ST patterns
	$\epsilon = 10$, $\sigma_{st} = 10$					
X-Wing 1	78	7	151	78	7	**114**
X-Wing 2	72	3	129	**95**	**3**	71
X-Wing 3	87	2	**131**	88	2	**84**
$\epsilon = 20$, $\sigma_{st} = 50$						
X-Wing 1	77	15	73	82	17	65
X-Wing 2	93	26	43	**100**	29	**39**
X-Wing 3	100	10	60	100	10	60
$\epsilon = 170$, $\sigma_{st} = 50$						
X-Wing 1	45	38	27	**51**	42	24
X-Wing 2	51	10	15	49	8	17
X-Wing 3	60	12	21	**69**	13	19

Table 5.2: Evaluation of the spatio-temporal patterns issued from all patterns with $\sigma = 721$ and $\tau = 1$ for DyPlagram and for DyPlagram_st. The labels are created from the size of the region. For both algorithms, the third column indicates how many spatio-temporal patterns have been discovered. Those experiments were conducted on the synthetic video Anim1 with 3 identical X-Wings.

patterns using the strategy described in 4.3.4.2. The same temporal threshold $\tau = 1$ and frequency threshold $\sigma = 721$ where used to directly extract comparable spatio-temporal patterns with DyPlagram_st.

Next, the precision and recall of every spatio-temporal pattern whose first occurrence mapped to an object o in a video frame i were computed with respect to the object o. The precision allows to assess if spatio-temporal patterns are robust, i.e, if they tend to follow the same object on all frames they have an occurrence in. The recall tells us how much of a target, in term of number of frames, spatio-temporal patterns cover.

Table 5.2 summarizes the results obtained with the spatio-temporal patterns generated by post processing the frequent patterns of DyPlagram, and the spatio-temporal patterns of DyPlagram_st. The spatio-temporal patterns were generated with a minimum freq_{st} threshold $\sigma_{st} = 10$ and 50, and 3 different spatial thresholds ϵ of 10, 20, and 170 pixels (from 20 to 160 pixels, the results were quite similar and thus are not reported here). For each experimented pair (σ_{st}, ϵ) and for each target X-Wing in the video, the first two columns give the average precision and recall computed for its associated spatio-temporal patterns (as defined in our strategy). In addition, the third column shows the total number of such patterns.

Table 5.2 shows that the spatio-temporal patterns obtained with DyPlagram_st are in general less numerous, more precise and have a better recall than the ones obtained with DyPlagram. The distance threshold ϵ has an important impact on the obtained results. Indeed, if it is set too low (to 10 pixels, in our example), we ob-

TABLE 5.3: Precision and recall computed for the spatio-temporal patterns produced by DYPLAGRAM_ST on the real video with $\sigma = \sigma_{st}$, and $\mu = 0.65$

		$\epsilon = 10$			$\epsilon = 20$		
τ	σ_{st}	Precision(%)	Recall(%)	# ST patterns	Precision(%)	Recall(%)	# ST patterns
10	100	**100**	26.18	10	**92.48**	22.97	13
	50	93.55	17.40	20	91.35	15.44	25
	10	89.78	2.87	294	89.70	2.72	334
25	100	91.28	35.34	11	89.02	30.03	14
	50	90.28	25.12	18	83.79	20.14	24
	10	88.90	3.18	307	89.47	2.94	358
100	100	89.52	**38.21**	14	89.02	**31.03**	19
	50	92.27	24.38	27	90.30	22.45	30
	10	89.01	4.03	258	89.88	3.63	302

tain spatio-temporal patterns with high average precision for each X-wing as different occurrences of patterns which map to different X-wing are very well distinguished. However, this leads to a low average recall: since only very close occurrences of the same pattern are linked, the spatio-temporal patterns tend to be short (i.e., have low freq_{st}). When using a distance threshold $\epsilon = 10$, no spatio-temporal patterns with $\text{freq}_{st} \geq 50$ were found for X-wing2 for DYPLAGRAM, which explains why we used $\sigma_{st} = 10$ in this case. Conversely, for a higher ϵ of 170 pixels, the average precision drops as the different X-wings are not well distinguished anymore. For example, it was possible to obtain spatio-temporal patterns with higher recall for the X-Wing 1 (when comparing to the other experiments), but, they had low average precision. Since the X-Wing 1 gets partially out of the video frames around 6 times, a higher number of spatio-temporal patterns were derived for this X-wing for $\sigma_{st} = 50$ and ϵ of at least 20, which represent the different time intervals where this X-wing is visible through the video. As another example, the X-Wing 2 is hidden only twice by the X-Wing 3 (during around 15 frames) and never goes out of the video frames. This explains the lower number of spatio-temporal patterns found for this object, also for $\sigma_{st} = 50$ and $\epsilon \geq 20$. Note that, in the case of our example video, increasing the time constraint τ could increase the length of the spatio-temporal patterns and thus their recall but this would lead to a lower precision.

Experiments on the drone video The aim of those experiments is to demonstrate that spatio-temporal patterns can serve has a basis to track objects in real videos. We experimented on the influence of different values for the spatio-temporal thresholds, using $\tau = 10, 25, 100$ and $\epsilon = 10, 20$ (above 20 the precision started to drop significantly which is expected for large ϵ values), and different values for the frequency threshold with $\sigma_{st} = 10, 50$ and 100. The precision and recall results for

the spatio-temporal patterns returned by DyPlagram_st under those parameters are presented in Table 5.3.

As expected, the precision is a little higher with $\epsilon = 10$ (100% for $\epsilon = 10$ when $\tau = 10$ and $\sigma_{st} = 100$ against 92.48% for $\epsilon = 20$). The fact that the average recall also decreases with a higher distance is more surprising at first glance. This is explained by the fact that most of the time, $\epsilon = 10$ is enough to follow the drone, but sometimes the drone or the camera movement accelerates. In those cases a higher distance might give longer and better spatio-temporal patterns but also might introduce some noisy ones which would decrease the average recall and precision.

The average recall also decreases when we lower σ_{st}. This is due to the fact that when using a low σ_{st} DyPlagram_st outputs short spatio-temporal patterns that necessarily have a low recall. Lowering σ_{st} slightly reduces the precision of the spatio-temporal patterns but increases their number.

As also expected, higher gaps lead to better recall (38.21% for $\tau = 100$ when $\epsilon = 10$ and $\sigma_{st} = 100$ against 26.18% for $\tau = 10$) as well as improve the coverage of the spatio-temporal patterns in the whole video. The precision does not seem to be influenced by τ when we allow small spatio-temporal patterns (i.e., a low σ_{st}).

Overall this series of experiments have shown that spatio-temporal patterns are robust and can follow a target with high precision. The fact that a single spatio-temporal pattern is not enough to track an object across all the frames of a video is also confirmed by the low recall results. In the next sections we will present the experiments we conducted to show how this problem can be solved with the spatio-temporal paths or clusters of spatio-temporal patterns.

5.4.3 Spatio-Temporal Paths for Object Tracking

To assess the effectiveness of the spatio-temporal paths for object tracking, we apply the following strategy. We first build the occurrence graph and then, for each target object, we select the occurrences matching it in the first frame. Then we compute the path of lowest cost starting from those occurrences and reaching the last frame using Dijkstra's shortest path algorithm. This means that this strategy is better suited to cases where the target appears in the last frame. Although it could still follow a target that is not present in the end of the video, in the last frames, this strategy would drift to some pattern that does not represent the object. In all experiments reported here we use a similarity of 2/3 ($\mu = 0.65$). We also tried with a similarity of 3/4 ($\mu = 0.75$) but in this caused the occurrence graph to have to few edges to find a complete track of any object. With a similarity of 1/2 ($\mu = 0.5$) the occurrences graphs took a lot of space in memory because of the number of edges

TABLE 5.4: Evaluation of the spatio-temporal path with $\sigma = 250$, $\sigma_{st} = 150$, $\mu = 0.65$, $\epsilon = 20$. The numbers between parenthesis correspond to the best precision and recall of the best path in term of recall, and the emphasized results are the best results for each X-Wing

		Size Discretization			Color Discretization		
	τ	Precision(%)	Recall(%)	Paths	Precision(%)	Recall(%)	Paths
X-Wing 1	10	**98.32** (99.72)	**97.50** (99.30)	34	93.92 (99.74)	93.60 (99.86)	21
X-Wing 2		**99.63** (99.73)	**97.26** (98.19)	24	98.65 (100)	**96.82** (99.02)	17
X-Wing 3		**9.49** (16.64)	**8.70** (15.39)	4	- (-)	- (-)	0
X-Wing 1	25	95.79 (100)	94.59 (99.02)	38	**99.17** (99.73)	**98.40** (100)	21
X-Wing 2		65.66 (99.61)	64.61 (98.05)	32	98.54 (100)	96.34 (99.02)	20
X-Wing 3		2.93 (9.09)	2.50 (8.59)	29	31.95 (31.95)	29.54 (29.54)	2
X-Wing 1	100	79.05 (100)	74.37 (94.31)	42	97.76 (100)	95.36 (99.30)	29
X-Wing 2		72.57 (97.53)	67.05 (93.62)	35	**98.87** (100)	96.30 (99.02)	39
X-Wing 3		5.42 (18.46)	4.82 (16.36)	31	**86.27** (90.52)	**75.92** (82.80)	23

between the occurrences and the track obtained drifted easily to elements of the background.

In practice the minimum support threshold σ can be set, for example, to 1/5 of the total number of frames (to make sure that the patterns occur enough and help the mining process). By default, it will be equal to the σ_{st} threshold. σ_{st} should be set as low as possible (depending on available memory). The τ should, in general, be set as high as possible (as will be shown in the experiments). The ϵ constraint depends on the motion speed of the target object and on the resolution of the video. Most of the time we use 20 pixels.

5.4.3.1 Evaluation of the Spatio-Temporal Path for Object Tracking

For the synthetic videos Anim2 with the 3 different airplaines, and for the drone video, we report the precision and recall results for the spatio-temporal paths. Each time we selected the spatio-temporal pattern with lowest weight starting from an occurrence in the area selected by the user and ending in the last frame. The precision and recall results are computed on the occurrences taken by the spatio-temporal path with lowest weight.

5.4.3.2 Experiments on the Synthetic Video Anim2

The experiments reported in Table 5.4 show the precision and recall results for the paths obtained on the synthetic video when varying the gap between 10 and 100.

Because of the nature of the video, we use a global minimum support σ of 250 in order to prune the number of frequent patterns. Indeed, since the synthetic video has been especially made to produce stable graphs, DYPLAGRAM_ST returns a lot of frequent patterns on this dataset which leads to a huge global occurrence graph

TABLE 5.5: Precision and recall computed for the spatio-temporal paths for the real video with $\sigma = \sigma_{st}$ and $\mu = 0.65$, using the color discretization to label the nodes of the graphs.

		$\epsilon = 10$			$\epsilon = 20$		
τ	σ_{st}	Precision(%)	Recall(%)	Paths	Precision(%)	Recall(%)	Paths
10	100	96.30 (96.30)	67.89 (67.89)	1	98.23 (100)	80.94 (82)	2
	50	98.25 (100)	70.00 (71.26)	2	26.16 (38.96)	24.03 (36.21)	3
	10	91.93 (93.34)	69.60 (70.63)	8	18.75 (36.09)	17.88 (34.73)	8
25	100	98.43 (100)	68.89 (70)	6	98.51 (100)	78.68 (79.68)	6
	50	98.66 (100)	69.05 (70)	7	98.72 (100)	78.82 (79.68)	7
	10	99.06 (100)	69.36 (70.21)	10	**99.03 (100)**	**80.63 (81.36)**	10
100	100	100 (100)	67.42 (67.78)	8	100 (100)	77.52 (79.68)	9
	50	100 (100)	67.36 (67.68)	9	100 (100)	77.54 (79.68)	9
	10	100 (100)	67.21 (67.78)	10	99.26 (100)	79.17 (79.78)	10

that possibly does not fit into memory for processing. To be able to perform various experiments, especially with the size discretization which does not permit to distinguish the three X-Wings at the mining step, we set the σ_{st} to 150 (although as already discussed, it is better to set it as low as possible).

Overall, we obtain very good results for the first two X-Wings (precision and recall close to 100%). We can clearly see the lack of discriminative power of the size discretization when the gap increases. Indeed the paths start to follow different X-Wings, reducing their precision and their recall. For those two X-Wings the color discretization always shows good results, with average precisions and recalls close to the ones of the best paths (values in brackets). Since the 3rd X-Wing moves back and forth horizontally across the field of view (getting almost completely out every 120 frames), only few paths starting on this X-Wing manage to reach the end of the video when we use a low gap. The paths which uniquely follow this X-Wing are thus more expensive than other paths on which the algorithm can "jump" using the similarity edges, decreasing the precision and recall. As we can see, increasing the gap allows to overcome this problem with the color discretization while keeping good results for the other two X-Wings.

5.4.3.3 Real Video with the drone

The experiments reported in Table 5.5 were made without using a global minimum support threshold (which is equivalent to set $\sigma = \sigma_{st}$). Because of the segmentation, this dataset is a lot less stable than the synthetic one resulting in less frequent patterns. For this one, only the color discretization gave good precision/recall results (we also tried the size and some other color discretizations using the HSV color space but our simple discretization of the RGB space worked better).

Table 5.5 shows the results for the spatio-temporal patterns of DyPlagram_st on the real dataset for the color discretization.

A distance ϵ equal to 20 gives the best results in most cases with high precision and good recall ($99.0380.63\sigma_{st} = 10$ for example). However, the values for $\tau = 10$ show the limits of the use of the shortest path algorithm to tackle our problem. Similarly to what was happening with the third X-Wing in the synthetic video, the shortest path might not always be following the object we want to track if elements in the background or other objects offer better stability than the original target and are close enough to "jump" on them.

The results with our preferred setting (low $\sigma_{st} = 10$, high $\tau = 100$ and a distance $\epsilon = 20$) show that the spatio-temporal paths can indeed be used to follow an object in the video. The similarity edges introduced are very useful to increase the recall of the patterns and experiments with a higher similarity constraint (for example with $\mu = 0.8$) provide worst results. This shows the importance of this "inexact" matching phase in the process. On the downside, the choice of the labels on the node (here it is a color information) seems to play a very important role to get interesting spatio-temporal patterns although it is difficult to evaluate in an unsupervised setting what could be the best ones. One solution could be to attach more diverse informations on the labels of the nodes to overcome this problem.

5.5 Clusters of Spatio-Temporal Patterns for Tracking

In this section we present the experiments we conducted to show that the top clusters, according to our ranking strategy, correspond to interesting objects and can be used to follow those objects.

5.5.1 Experimental design

To asses the quality of the tracks returned by our approach, we compare our algorithm to two other state of the art algorithms called TLD (*Kalal et al.* (2010)) and CT (*Zhang et al.* (2012)) that we already discussed in section 2.3.3. We also apply our algorithm on the video segmentation of *Grundmann et al.* (2010). To summarize we compare the 4 following approaches:

- TLD (Track Learn Detect) is a tracking algorithm *Kalal et al.* (2010) that requires manual selection of the target.
- CT (Compressive Tracking) is a tracking algorithm *Zhang et al.* (2012) that also requires manual selection of the target.

- TRAP is our tracking algorithm which mines frequent spatio-temporal patterns and clusters them. It uses the simple segmentation algorithm presented in *Felzenszwalb and Huttenlocher* (2004) for the real video (and the original regions for the synthetic ones). The value for the three parameters of the algorithm (τ, σ_{st} and ϵ) are discussed bellow.

- TRAP + VS (Video Segmentation) uses the second type of segmentation presented in *Grundmann et al.* (2010).

For the clusters obtained with our approach, the *precision* corresponds to the proportion of occurrences of the cluster that have all their nodes in the bounding box of the ground truth (at the corresponding frame). The *recall* of a cluster is the number of frames in which at least one occurrence of this cluster has all its nodes in the bounding box of the ground truth.

TLD and *CT* are given the ground truth of the first frame of each video as input. Both algorithms return a sequence of bounding boxes representing the track of the followed objects. The *precision* is the area the bounding boxes of the track and of the ground truth have in common, divided by the area of the bounding boxes of the track. The *recall* of the algorithm is the number of frames in which the center of the bounding box of the track is inside the bounding box of the ground truth.

As explained in Section 5.2.2.3, the choice of the clusters that are used to track the objects of interest is an important problem. In the experiments, we will show the results for the *Longest* cluster as defined in Section 5.2.2.2 but also the results for the *Best* cluster in the hierarchy (we chose the best cluster for all possible cut of the clustering hierarchy). This best cluster is the one for which the $Precision{*}Recall{*}100$ is the highest. Of course, these "best" results are just given to assess the possible improvements for our algorithm since they cannot be used in an unsupervised setting. In some experiments, the two criteria we use (cut with the lifetime and keep the longest cluster) are not always the best but we can show that a very good cluster exists and could be found using a different criteria.

5.5.1.1 Parameters of DyPlagram_st

The spatial threshold ϵ should be high enough depending on the motion of the objects and the motion of the camera. This can be estimated on the first frames of the video using optical flow techniques such as the ones described in section 2.2.3. However, setting this to 20 (pixels) for all experiments gave sufficiently good results. In general, giving a high value for this parameters will increase the mining time but will not harm the results. Similarly the time threshold τ is set for all videos to 25

		Anim 1: Identical Objects						Animation 2: $\neq$ Objects					
		Obj 1		Obj 2		Obj 3		Obj1		Obj2		Obj3	
		P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
TLD		22	14	**90**	17	0	0	14	13	36	5	0	0
CT		39	52	0	0	0	0	68	96	0	0	0	0
TRAP	Longest	**97**	**90**	41	**99**	**21**	**63**	**100**	**99**	**91**	**99**	8	12
	Best	99	90	92	88	87	49	100	99	91	99	72	92
VS+C	Longest	14	14	47	55	35	38	91	95	67	84	18	43
	Best	100	52	97	63	100	21	91	95	97	63	53	45

FIGURE 5.6: Precision and Recall of the CT, TLD and TRAP algorithms using the standard color segmentation, the TRAP algorithm using the video segmentation (TRAP+VS) and the video segmentation alone with a clustering phase (VC+C) on the two synthetic videos Anim1 and Anim2.

frames (1 second of the video). Again, this may not be the best set of parameters especially for the car video which is the most complex to deal with. The frequencies thresholds (σ and σ_{st}) should be set after having found a working τ and ϵ to obtain a significant number of spatio-temporal patterns ($600 < \#patterns < 2000$). A too large number would also slow down the algorithm. By default $\sigma = \sigma_{st}$. Note that σ controls the frequency of the patterns from which the spatio-temporal patterns can be generated. However, a very high σ_{st} threshold (for example, more than 20% of the length of the video) means that the structure of the object (and thus of the patterns representing it) should not change at all during 20% of the frames which is not very reasonable for most of the real videos that are recorded by amateurs. Thus, we impose that σ_{st} is always bellow 20% of the length (in frames) of the video. If the number of patterns is still too big with this bound, we can increase σ to get inside the $\#patterns$ bounds.

5.5.2 Results

We now present the results obtained for our clusters of spatio-temporal patterns in term of tracking quality and efficiency.

5.5.2.1 Tracking quality

Synthetic videos In Figure 5.6 we can see that CT and TLD do not give good results on the synthetic video especially when the 3 planes are identical. Indeed, the initial bounding box given in the ground truth includes a lot of background between the wings of the planes which corrupts the appearance model learned. Besides, there are occlusions between the objects and their rapid changes in direction make them hard to track. Our approach gives good results (97/90 P/R for Anim1 and 100/99 for Anim2) on the first plane which is the most stable. However there is no really

		Drone		Car 1000		Car 2000		Car 3000	
		P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
TLD		63	88	65	68	**55**	**46**	**55**	**31**
CT		**84**	**99**	9	14	8	8	5	5
TRAP	Longest	81	99	92	83	10	98	4	52
	Best	97	99	90	98	90	51	90	34
VS+TRAP	Longest	24	95	**92**	**90**	5	82	5	65
	Best	95	94	93	98	85	54	85	36
VS+C	Longest	**90**	**100**	0	0	0	0	0	0
	Best	100	100	95	100	84	100	98	79

FIGURE 5.7: Percentage of Precision (P) and Recall (R) of the CT, TLD and TRAP algorithms using the standard color segmentation, the TRAP algorithm using the video segmentation (TRAP+VS) and the video segmentation alone with a clustering (VC+C) on the two real videos.

FIGURE 5.8: Occurrences of frequent patterns (in green) in the longest cluster for the first 1000 frames of the car video

good cluster (where the recall and the precision would be both above 90%) in all the hierarchy representing the second and the third object. For the second object, the best cluster has 92% precision and 88% recall but this cluster exists only when cutting the hierarchy at 11 clusters whereas our lifetime criteria cuts the hierarchy at 252 clusters and thus does not allow us to find the best one. For the third object, the best cluster was only 361 frames long so it was not selected as the longest one. Because the third object goes almost completely out of the field of view for 3 to 4 seconds several times in the video, the clusters representing this object were easily split. When directly clustering the patterns extracted from the video segmentation (VS+C) and for Anim1, the best clusters have a high precision but their recall is low, reaching 63 (less for the longest cluster). This comes from the fact that the video segmentation is of course less accurate than the original segmentation and tends to over-segment regions due to the frame-to-frame region matching which decreases the relevance of the patterns.

The results for Anim2 show that the color difference between the three objects usually helps all the trackers (except for TLD). For our approach, the longest clusters

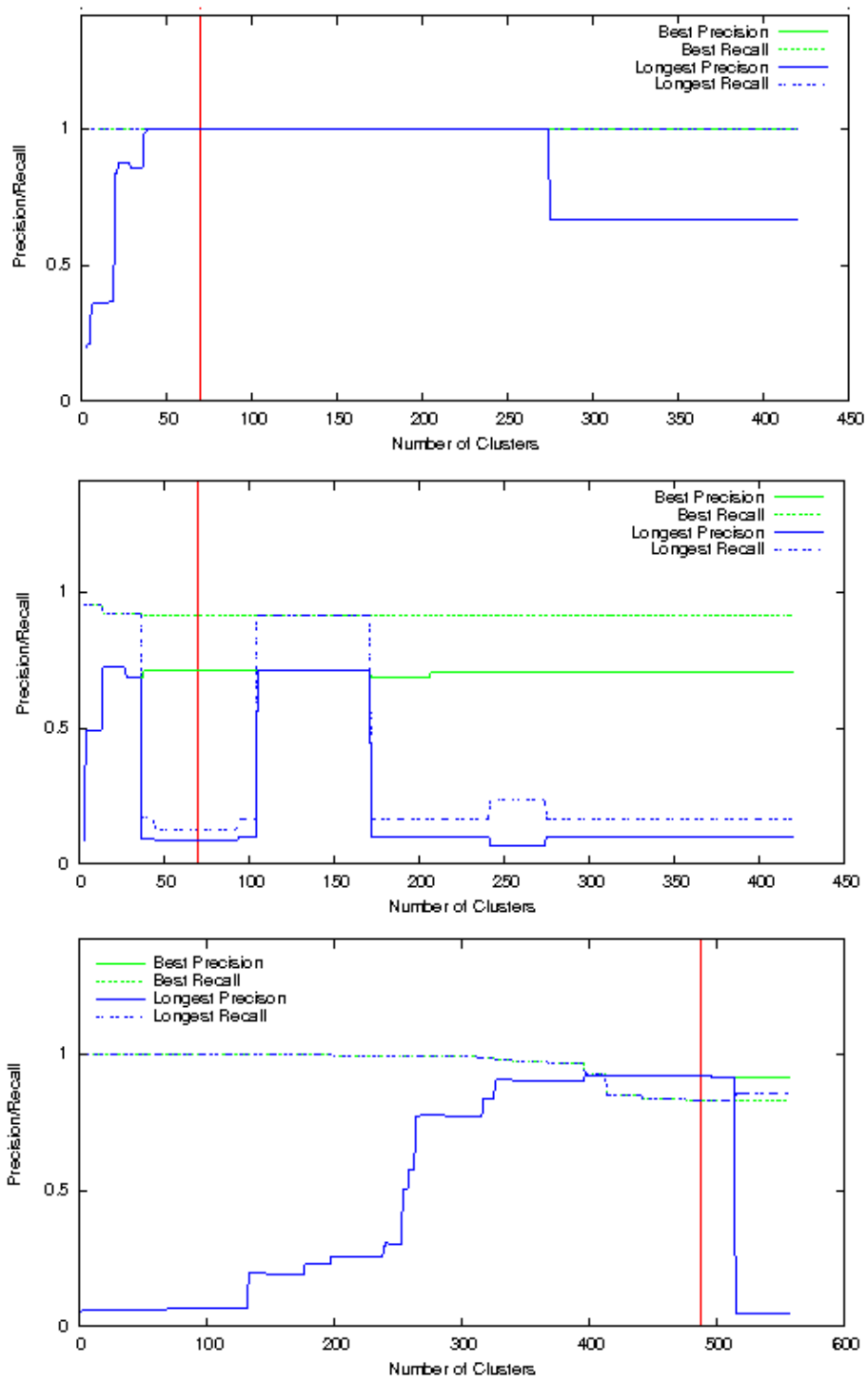

FIGURE 5.9: Precision an recall results of the best and longest clusters output by TRAP for the object 1 (top) and object 3 (middle) of animation 2 and for the car (bottom) for car1000. The vertical red line is the lifetime cut.

at the highest lifetime were the best ones in the hierarchy as can been seen in Figure 5.9 (top). The difference between the objects was discriminative enough to be able

to follow the third object with a best cluster with 72% precision and 92% recall. Unfortunately this good cluster was at the 14th level of the hierarchy while it was cut at the 70th level as can be seen in Figure 5.9 (middle). In this later case the size of the best cluster decreases around the 50th level of the hierarchy which causes it to be ranked lower than bigger clusters that do not match the third object.

The video segmentation did less mistakes on this animation and the results are thus better for the VS+C method. As we can see, the best cluster for the first X-Wing was the longest one with a precision of 91% and a recall of 95%. For the second X-Wing, the recall drops to 84% and the precision to 67%. For the last object, there is not enough patterns to build good clusters (the best one only has 53% precision and 42% recall).

Real videos TLD and CT both track the drone for almost all the video, the former with 63/88% (P/R) and the later with 84/99% (P/R) (see Figure 5.7). TLD loses it for some frames which results in a lower recall. Due to the large size of the output bounding box in some frames, the precision is lower than for our approaches for both algorithms. TRAP also follows the drone with 99% recall, but the longest cluster is less precise than the best cluster (81% versus 97%). Note that just reducing σ_{st} to 10 in this case would allow us to find the best cluster. Clustering the spatio-temporal patterns extracted from the video segmentation (VS+TRAP) also produces some good clusters but not at the level the lifetime cuts the hierarchy. Thanks to a high number of spatio-temporal patterns, VS+C obtains good results (the longest cluster has 100% recall and 90% precision).

From Figure 5.7, we can confirm that the car video is a much more difficult tracking problem. TLD follows the car until the frame 1305, losing it occasionally, but never with a good precision. CT never succeeds in following the car. For both types of segmentation, the longest cluster returned by TRAP follows the car until the frame 1200 and then loses it. At this point of the video the car is small and both segmentation segmented it in only one region. Since occurrences of frequent spatio-temporal patterns have at least 3 nodes (1 face, this is imposed by the DyPlagram_st algorithm), there is none matching the car in this part of the video. The best patterns for the first 2000 and 3000 frames all end at this frame, and, since there is no other long pattern matching the car, the longest clusters has a bad quality. As shown in tab 5.10, augmenting the gap allows us to skip the frames of the video where the car is too small which produces better results. However, the algorithm faces the same situation for a longer time at the frame 2300. This shows that if the gap threshold τ can allow us to deal with some situations where the object is hard to detect, it would

		Car1000		Car2000		Car3000	
		P	R	P	R	P	R
TRAP	Long	90	99	96	73	7	83
	Best	90	99	93	87	92	61
VS	Long	90	99	93	87	7	84
+TRAP	Best	92	98	88	91	22	39

FIGURE 5.10: Percentage of Precision (P) and Recall (R) obtained for the car video when increasing the gap τ to 75 for the TRAP algorithm.

be better to introduce a mechanism specifically designed to deal with long term occlusions. Figure 5.9 shows that, on the first 1000 frames, the longest cluster returned by TRAP is always the best one until the lowest levels of the hierarchy. This shows that the length criterion can be very good to find the best cluster when sufficient patterns representing the objects can be extracted and when no long disappearance of the targets splits the clusters. The VS+C method builds good clusters but only at the higher levels of the hierarchy, they are thus not found using our lifetime criterion.

In conclusion, our unsupervised methods give comparable (and most of the time better) results than the state-of-the-art trackers TDL and CT. However, we do not need to select the objects of interests in the first frame of the video which makes this method usable in practice to treat batches of off-line recorded videos such as Youtube ones.

5.5.2.2 Efficiency

For the synthetic videos, when keeping the default parameters for τ and ϵ we fell into the number of patterns problem mentioned in Section 5.5.1.1. For both animations, σ_{st} was set to 150 but σ was set to 250 for the first animation and to 220 for the second one. As can be seen in the Figure 5.11 it takes more than 15 minutes to process 1700 patterns in both cases. For the video segmentation (VS), we can not control the number of output patterns and for the simple animation video, we get only around 100 of them which made the clustering process very fast. Because of many changes in appearance for the real videos, there were less frequent patterns so we could keep the default setting for all parameters (except σ_{st} as discussed in Section 5.5.1.1). For TRAP, we used $\sigma_{st} = 15$ for the drone and $\sigma_{st} = 25$ for the car, and we set it to 35 for both videos when mining the more stable video segmentation. We mined the 5600 frames of the car video at once and then restricted the occurrence graph to the first 1000, 2000, and 3000 frames, this explains why the time results for the mining step of this video are constant. This is also why the number of patterns can

		Exec Time (s)			# pat
		Mine	Clust	Total	
Anim1	TRAP	11	1042	1053	1708
	VS+C	0	7	7	116
Anim2	TRAP	9	1180	1189	1667
	VS+C	0	6	6	113
Drone	TRAP	28	952	980	1421
	VS+TRAP	9	722	731	1349
	VS+C	0	521	521	1095
Car1000	TRAP	109	231	340	575
	VS+TRAP	212	204	416	520
	VS+C	0	4560	4560	2923
Car2000	TRAP	153	1005	1158	1046
	VS+TRAP	153	954	1107	985
	VS+C	0	28524	28524	5196
Car3000	TRAP	153	1758	1911	1232
	VS+TRAP	153	1981	2134	1237
	VS+C	0	69866	69866	6543

FIGURE 5.11: Execution time and number of patterns output by the TRAP algorithm and the method which uses the video segmentation followed by a clustering step (VS+C)

be as low as 500 when processing only the first 1000 frames. As we can see, the VS+C approach produces a lot more patterns which greatly increased the computation time.

In conclusion, the mining phase can give better results and is more efficient than directly using the output of the dynamic segmentation for real videos. However, both methods are far from usable in real time although the clustering step could easily be improved by designing an optimized algorithm.

5.6 Conclusions

The experiments we conducted on the spatio-temporal patterns showed they are meaningful in a video tracking context. When the first occurrence of a spatio-temporal pattern matches an object, the rest of its occurrences tend to also match the same object with high precision.

We also described two techniques that use spatio-temporal patterns to track the main objects of videos.

The spatio-temporal paths suffer from multiple limitations. They still require the user to select the target himself. They also tend to drift from the target, especially if the video is long. Indeed, it is sometimes less costly to move from the original selected occurrences to occurrences of a pattern that can be more easily followed.

Therefore sometimes, the shortest path computed starts by taking multiple similarity edges until it reaches the occurrence of a more stable pattern than the ones matching the target.

Our clustering approach solved the problem of the target selection. However, it is still unclear where exactly to cut the hierarchy to obtain the clusters and how many of them are matching an interesting object. Nonetheless, the highest ranked cluster often corresponds to the main object of the video.

5.7 Possible Applications

The works performed in the context of this thesis have lead to the patenting of two applications suggested by our industrial partner. A first possible application of our algorithm *TRAP* could be to summarize interactions between the main objects in videos. Automatic video summary is a broad and difficult problem that requires to be capable of analyzing a video and understanding what is happening in it. A major difficulty is that, to properly describe the content of a video, humans usually make use of their capabilities to separate the interesting objects from the none interesting ones and the background. So far this requires to know which object to look for in advance. Our technique can help solving this problem since it can identify and track interesting objects. Here we focus on an application designed to provide a textual summary of the location of main objects and their interactions between each other, i.e, when the image segments representing them overlap. Given the tracks of objects extracted using the *TRAP* algorithm, this application builds a XML file recording, for each object, the beginning and ending frames of each video segments in which the object appears without interruption. For each frame of each one of those video segments, the XML file includes the coordinates of the bounding boxes around the object as well as the segmented regions corresponding to it. The XML file also stores, for each object, a list of interactions, i.e, the beginning and ending frames of a video segment in which the bounding boxes of two objects overlap. This application could be used online by uploading a video to process, or by supplying an url to a video on Youtube for instance. Figure 5.12 gives an example of such an XML file describing the interactions of the main objects.

Such an application would also be useful as a basis for further description of a video content. Indeed, once the regions of the frames representing an object are known, it is possible to identify it by comparing its appearance to a database of objects and obtain more informations about it, such as its category for example (e.g., human, animal, car, plane, boat etc...). It could also be used to quickly retrieve video

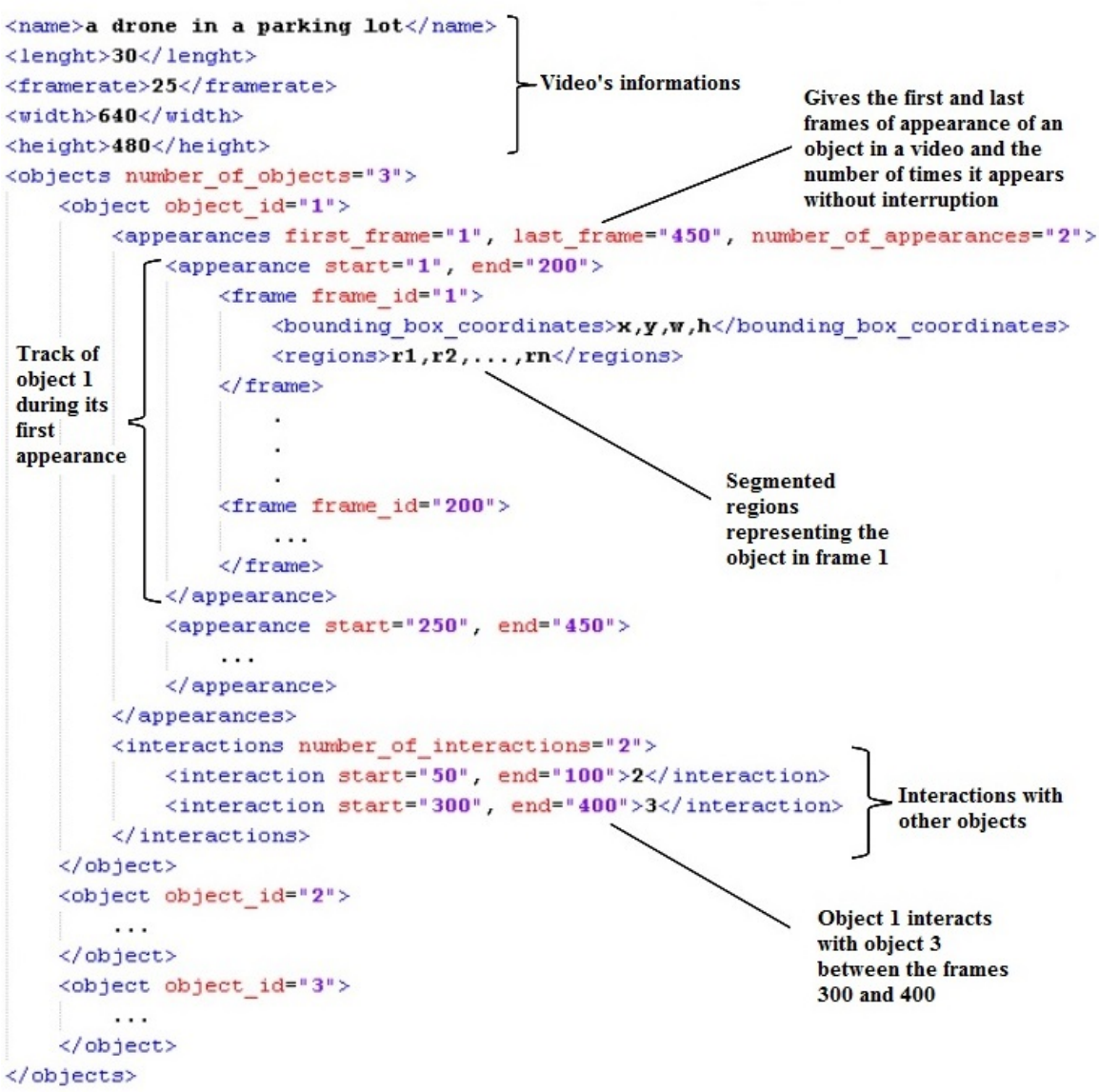

FIGURE 5.12: Example of XML file summarizing a video

segments where two objects interact with each other. Splitting a video in several segments or detecting key-frames depending on the objects present in the field of view and their interactions is another possible use for this application. It could also be useful to retrieve videos in which a particular object appears a lot.

The second application we patented consists in using the clusters returned by our algorithm *TRAP* as a basis to produce a cloud of images representing the main objects of a video, similarly to the way clouds of word representing textual documents. From each cluster we can derive an image segment in each frame t, corresponding to the object it represents, by recovering the part of the original image that is covered by the segmented regions of the occurrences of the cluster in frame t. Then, we can compare

the visual appearance of the different image segments in each frame and decide which one(s) best represents the object by selecting, for example, the segments that are the most similar to the others in the cluster. Then this set of segments can be added to a cloud of image tags representing the video. The size of the tag representing each object can vary with respect to the importance of the object, i.e., the number of frames covered by the cluster it has been extracted from. Such an application could be useful to efficiently describe the visual content of a video before watching it. Being able to represent a video by a set of image segments also offers the possibility to perform queries, on a database of videos, by supplying an image of an object we are interested in and retrieve videos in which it is important.

CHAPTER 6

Conclusion et Perspectives

6.1 Conclusion

Dans cette thèse nous nous sommes intéressés à l'utilisation de techniques de fouille de graphes pour résoudre certains des problèmes rencontrés dans le cadre de la tâche de suivi d'objets dans les vidéos acquises à partir d'une caméra en mouvement. Cette approche nous a permis de répondre au problème général de suivi d'objets ouvrant la porte à de nombreuses applications, comme le résumé ou l'indexation automatique de vidéos en fonction de leur contenu visuel qui peut être décrit de manière plus pertinente en se basant sur les objets de la scène et leurs interactions. Plutôt que de rechercher des objets correspondant à des modèles cibles, pré-entrainés ou sélectionnés par l'utilisateur, nous nous sommes intéressés à ce qui constitue un objet intéressant afin de pouvoir découvrir automatiquement les objets principaux d'une vidéo et les suivre. Nous sommes partis de l'hypothèse que lorsque la caméra est en mouvement, les objets intéressants doivent apparaitre plus souvent que l'arrière plan.

Pour identifier et suivre les objets principaux, nous nous sommes basés sur l'hypothèse que la topologie des objets ne change pas brutalement au cours d'une vidéo. A l'aide d'un algorithme de segmentation couleur, nous avons segmenté en régions chacune des images des vidéos traitées. Nous avons ensuite modélisé la topologie de chaque image sous forme de graphes plans. Le problème d'identification et de suivi des objets principaux d'une vidéo est alors analogue à celui de l'extraction de sous graphes fréquents dans la base de données de graphes représentant les images d'une vidéo.

Dans la première partie de cette thèse, nous avons présenté un état de l'art du suivi d'objets (chapitre 2) et de la fouille de graphes (chapitre 3). Ceci nous a permis d'identifier les faiblesses des méthodes actuelles de suivi et d'étudier les algorithmes les plus efficaces en fouille de graphes.

Dans la deuxième partie, nous avons présenté nos contributions dans les domaines de la fouille de graphes (chapitre 4) et du suivi d'objets (chapitre 5).

Dans le domaine de la fouille de graphes, notre première contribution (*Prado et al.* (2011)), publiée dans la conférence française "Conférence d'Apprentissage" en 2011, a été de développer PLAGRAM, un algorithme de fouille de graphes plans basé sur le très populaire algorithme de fouille de graphes généraux nommé GSPAN (*Yan and Han* (2002)). PLAGRAM tire parti des travaux de *Damiand et al.* (2009) sur l'isomorphisme de graphes plans pour compter efficacement le support des sous graphes pendant le processus de fouille. Nous avons montré expérimentalement que l'exploitation de ces travaux, couplé à notre stratégie d'extension consistant à ajouter des faces complètes, permettait à PLAGRAM d'être significativement plus efficace que GSPAN pour extraire les sous graphes fréquents d'un ensemble de graphes plans.

Notre seconde contribution au domaine de la fouille de graphes, publiée dans le journal international "Intelligent Data Analysis" en 2013 (*Prado et al.* (2013)), a été le développement d'une technique permettant d'introduire des contraintes spatio-temporelles au sein du processus de fouille afin de limiter le nombre de sous-graphes fréquents générés. Le nouvel algorithme résultant, appelé DYPLAGRAM, étend PLAGRAM en introduisant une nouvelle mesure de fréquence freq_{seq} qui permet d'exploiter une contrainte temporelle réduisant le nombre de motifs fréquents découverts. Une étape de post-traitement utilise les occurrences de ces motifs fréquents pour construire un graphe des occurrences dans lequel deux occurrences sont connectées si elles respectent les deux contraintes de temps et d'espace. Les composantes connexes de ce graphe des occurrences sont ensuite calculées pour générer des motifs spatio-temporels. Les expérimentations conduites ont démontré que l'utilisation de la contrainte temporelle pendant l'étape de fouille augmente significativement l'efficacité de l'algorithme.

Notre troisième contribution dans le domaine de la fouille de graphes, publiée dans la conférence internationale "European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases " en 2012 (*Diot et al.* (2012)), a été le développement de l'algorithme DYPLAGRAM_ST, qui étend PLAGRAM pour extraire directement des motifs-spatio temporels dont la mesure de fréquence freq_{st} est supérieure à un seuil minimum. Les résultats expérimentaux montrent que l'exploitation des contraintes spatiales et temporelles rend DYPLAGRAM_ST beaucoup plus efficace en temps de calcul que DYPLAGRAM qui n'exploite que la contrainte temporelle.

Dans le domaine du suivi d'objets nous pouvons énumérer 2 principales contributions. Dans un premier temps nous avons présenté, dans notre article *Diot et al.*

(2012), une méthode enrichissant le graphe des occurrences avec des arêtes de similarité entre les occurrences de motifs différents ayant une forte proportion de nœuds en commun. Les arêtes de ce graphe des occurrences enrichi ont été pondérées en fonction de la similarité des occurrences (pour les arêtes de similarité) ou de leur distance spatio-temporelle (pour les arêtes du graphe des occurrences original). Nous avons montré expérimentalement, sur des vidéos synthétiques et sur une vidéo réelle simple, qu'il est possible de suivre un objet en suivant le chemin le plus court dans le graphe des occurrences enrichi. Cette première méthode nécessite toutefois l'intervention de l'utilisateur pour sélectionner la cible. De plus, les chemins spatio-temporels doivent se terminer dans la dernière image de la vidéo alors que la cible n'y est pas nécessairement présente.

Nous avons alors développé une autre stratégie permettant de résoudre cet inconvénient. Cette deuxième méthode, publiée dans la conférence internationale "International Conference in Pattern Recognition" en 2014 (*Diot et al.* (2014)), consiste à mesurer la similarité de deux motifs spatio-temporels en comparant leur trajectoire. En utilisant cette mesure de similarité dans un algorithme de clustering hiérarchique, nous avons pu construire des ensemble de motifs spatio-temporels ayant des trajectoires similaires. Nous avons ensuite trié ces ensembles en fonction de leur longueur (en nombre d'images couvertes) et de leur taille en nombre de motifs spatio-temporels. Nous avons montré expérimentalement, sur des vidéos réelles, que parmi ces ensembles de motifs spatio-temporels, les mieux classés correspondent souvent aux objets principaux.

6.2 Perspectives

Les travaux exposés dans cette thèse présentent bien sûr un certain nombre de limitations. Tout d'abord, notre hypothèse de base qui consiste à affirmer que les objets intéressant apparaissent plus souvent que l'arrière plan n'est valide que si le mouvement de la caméra est suffisamment grand, ce qui n'est pas nécessairement le cas de toutes les vidéos que l'on trouve par exemple sur YouTube. Ensuite, le fait qu'il soit difficile d'établir le nombre d'objets présents et à quel niveau découper la hiérarchie pour produire les clusters de motifs spatio-temporels limite également l'utilisation générique de notre méthode. Une troisième limitation de notre approche réside dans la stratégie d'extension employée par notre algorithme de fouille de graphes plans. Celle-ci consiste à ajouter des faces complètes, ce qui impose que les objets soient segmentés en suffisamment de régions pour pouvoir être représentés par au minimum une face. Cette contrainte rend notre approche très dépendante de l'algorithme de

segmentation utilisé. De plus, les variations d'illumination peuvent conduire à des segmentations très différentes d'un même objet ce qui influence beaucoup la structure des graphes représentant la vidéo. Les techniques de segmentation vidéo comme celle de *Grundmann et al.* (2010) devraient permettre à l'avenir de limiter le bruit introduit dans les graphes par la segmentation. En effet, les récentes approches en matière de segmentation vidéo prennent en compte la segmentation des frames antérieures pour segmenter les frames suivantes. Ceci permet d'obtenir une segmentation plus stable, malgré les variations d'illumination, d'orientation de la scène ou le flou cinétique dû au déplacement des objets et de la caméra. Grâce à une segmentation plus stable, notre algorithme de fouille pourrait extraire des motifs plus gros en terme d'arêtes, et donc plus discriminants. Avec une segmentation plus stable, il serait également possible de diminuer le seuil de la contrainte temporelle. Les occurrences des motifs spatio-temporels extraits seraient ainsi séparés par moins de frames ce qui rendrait l'estimation de leur trajectoire plus robuste.

Une approche alternative consisterait à s'abstraire de la segmentation en construisant différemment les graphes représentant les images d'une vidéo. Une possibilité serait d'extraire des points d'intérêts dans les frames et de se servir de ceux-ci, au lieu des régions qui sont moins robuste aux variations dans les images, pour construire un graphe modélisant leur topologie. Cependant, la relation de topologie entre points d'intérêts n'est pas aussi claire que celle entre les régions segmentées qui sont liées par leur relation d'adjacence. Une possibilité serait de trianguler ces points d'intérêts en les connectant de manière à obtenir des graphes plans composés de faces triangulaires. Même si nos expérimentations ont montré que, pour un même ensemble de nœuds de départ basés sur la même segmentation en régions, la fouille des graphes issus d'une triangulation de Delaunay est plus couteuse en temps de calcul que la fouille des graphes d'adjacence des régions, les motifs extraits restent pertinents dans le contexte du suivi d'objets.

Une intéressante piste de recherche future serait l'introduction de mesures de similarité d'apparence entre occurrences. La distance que nous avons utilisé pour regrouper les motifs spatio-temporels ne prend en compte que la trajectoire des motifs. Il serait envisageable d'introduire un terme de similarité d'apparence des occurrences, à base d'histogrammes de couleurs par exemple. Nous pourrions aussi exploiter les travaux sur les trackers tenant compte du contexte par l'intermédiaire de *supporters* dont nous avons discuté dans la section 2.3.4 comme ceux de *Yang et al.* (2009) ou *Grabner et al.* (2010). Les *supporters* pourraient par exemple être utilisés pour regrouper deux motifs spatio-temporels si les occurrences de l'un couvrent des *supporters* de l'autre et inversement. L'utilisation d'une mesure de similarité de l'apparence pourrait

aussi permettre de regrouper des clusters représentant le même objet mais disjoints à cause d'occlusions de longue durée. Enfin, l'apparence des clusters pourrait être utilisée afin de mieux identifier les bons clusters en favorisant ceux ayant une forte cohérence visuelle sur toute leur durée.

Il pourrait également être intéressant d'utiliser les motifs spatio-temporels en combinaison des travaux de *Lee et al.* (2011) dont nous avons discutés en section 2.3.6. En effet, comme nous l'avons vu, leur méthode se base sur les segments d'image produits par *Endres and Hoiem* (2010) qu'elle regroupe en clusters en fonction de leur affinité. Les segments d'image produits par *Endres and Hoiem* (2010) pourraient être utilisés comme base pour extraire des motifs spatio-temporels. L'appartenance à un même motif spatio-temporel pourrait surement être utilisée pour raffiner la mesure d'affinité entre les segments. En effet, si deux segments d'image ont tendance à apparaitre dans les même motifs spatio-temporels, cela indique qu'ils sont liés par une relation de topologie qui est consistante à travers la vidéo. Ceci constitue un fort indice de leur appartenance au même élément ou objet de la scène et peut donc être exploité pour mieux mesurer l'affinité entre deux segments d'image et mieux regrouper ceux ci en vue d'extraire l'objet principal d'une vidéo.

Une autre perspective serait d'utiliser les contraintes spatio-temporelles pour améliorer des algorithmes de fouille inexacte comme celui de *Holder et al.* (1994) et *Zhang and Yang* (2008) ou encore *Jia et al.* (2011) afin de leur permettre de traiter de plus gros jeux de données. Ces techniques de fouille inexacte ont l'avantage de pouvoir intégrer le bruit introduit dans les données par la segmentation et de compenser, dans une certaine mesure, les variations d'apparence dues au changement d'orientation des objets. Leur défaut principal est toutefois qu'elles doivent considérer de nombreux motifs lors de l'exploration de l'espace de recherche. Ceci les rend donc difficile à appliquer sur de gros jeux de données comme des vidéos, qui peuvent être composées de milliers de frames. Introduire des contraintes spatio-temporelles pourrait limiter le nombre de motifs évalués par les algorithmes de fouille et donc permettre un réel passage à l'échelle.

Enfin, les approches que nous avons développées lors de nos travaux nécessitent pour l'instant d'analyser toutes les images des vidéos pendant le processus de fouille. Ceci empêche nos méthodes d'être appliquées en temps réel. Les travaux de *Yang et al.* (2009) montrent, entre autre, que la fouille de données peut être utilisée de manière pertinente dans un contexte de suivi d'objets en temps réel si elle est appliquée sur une fenêtre d'images consécutives plutôt que sur l'ensemble de celles de la vidéo. Leur technique utilise la fouille d'itemsets pour détecter des *supporters* apparaissant fréquemment avec la cible. Leur approche pourrait donc certainement être amélio-

rée en utilisant les contraintes spatio-temporelles pour détecter des *supporters* plus robuste. D'autre part, nous pourrions utiliser le même principe de fenêtre d'images consécutives pour ne fouiller qu'un sous ensemble des images de la vidéo. Ainsi, nos approches pourraient traiter les vidéos en temps réel avec un délai de quelques images en fonction de la taille de la fenêtre d'images fouillées. L'application en temps réel de la méthode de clustering de motifs spatio-temporels que nous avons présenté dans *Diot et al.* (2014) est fortement limitée par le coût en temps de calcul de l'étape de clustering. En effet, celle-ci nécessite le calcul de nombreuses similarités entre les différents motifs spatio-temporels. En appliquant la fouille sur une fenêtre d'images, un plus petit nombre de motifs spatio-temporels serait détecté à chaque étape, ce qui nécessiterait moins de calculs de similarités. L'étape de clustering pourrait aussi être améliorée en regroupant, dans un premier temps, uniquement les motifs spatio-temporels les plus longs, puis en affectant, dans un deuxième temps, chaque motifs spatio-temporel restant au cluster dont il est le plus proche, comme le font les auteurs de l'approche présentée dans *Fragkiadaki and Shi* (2011) dont nous avons discuté dans la section 2.3.6. Ceci permettrait de diminuer drastiquement la combinatoire des calculs de similarités et de clustering, les rendant compatibles avec une approche temps réel.

BIBLIOGRAPHY

Agarwal, P. K., H. Edelsbrunner, O. Schwarzkopf, and E. Welzl (1991), Euclidean minimum spanning trees and bichromatic closest pairs, *Discrete & Computational Geometry*, *6*(1), 407–422.

Agrawal, R., T. Imieliński, and A. Swami (1993), Mining association rules between sets of items in large databases, in *ACM SIGMOD Record*, vol. 22, pp. 207–216, ACM.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993), *Network Flows : Theory, Algorithms, and Applications*, 1 ed., Prentice Hall.

Ana, L., and A. K. Jain (2003), Robust data clustering, in *Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. II–128, IEEE.

Anderberg, M. R. (1973), Cluster analysis for applications, *Tech. rep.*, DTIC Document.

Andriluka, M., S. Roth, and B. Schiele (2008), People-tracking-by-detection and people-detection-by-tracking, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, IEEE.

Avidan, S. (2007), Ensemble tracking, *Pattern Analysis and Machine Intelligence (PAMI)*, *29*(2), 261–271.

Babel, L., I. N. Ponomarenko, and G. Tinhofer (1996), The isomorphism problem for directed path graphs and for rooted directed path graphs, *Journal of Algorithms*, *21*(3), 542–564.

Babenko, B., M.-H. Yang, and S. Belongie (2011), Robust object tracking with online multiple instance learning, *Pattern Analysis and Machine Intelligence (PAMI)*, *33*(8), 1619–1632.

Baker, S., D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski (2011), A database and evaluation methodology for optical flow, *International Journal of Computer Vision (IJCV)*, *92*(1), 1–31.

Bar-Shalom, Y. (1987), *Tracking and data association*, Academic Press Professional, Inc.

Barnich, O., and M. Van Droogenbroeck (2011), Vibe : A universal background subtraction algorithm for video sequences, *IEEE Transactions on Image Processing*, *20*(6), 1709–1724.

Bay, H., T. Tuytelaars, and L. Van Gool (2006), Surf : Speeded up robust features, in *European Conference on Computer Vision (ECCV)*, pp. 404–417, Springer.

Benezeth, Y., P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger (2008), Review and evaluation of commonly-implemented background subtraction algorithms, in *International Conference on Pattern Recognition (ICPR)*, pp. 1–4, IEEE.

Berclaz, J., F. Fleuret, E. Turetken, and P. Fua (2011), Multiple object tracking using k-shortest paths optimization, *Pattern Analysis and Machine Intelligence (PAMI)*, *33*(9), 1806–1819.

Bergholm, F. (1987), Edge focusing, *Pattern Analysis and Machine Intelligence (PAMI)*, (6), 726–741.

Birchfield, S. T., and S. J. Pundlik (2008), Joint tracking of features and edges, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1–6, IEEE.

Bonnici, V., R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro (2013), A subgraph isomorphism algorithm and its application to biochemical data, *BMC Bioinformatics*, *14* (Suppl 7), S13.

Borgelt, C., and M. R. Berthold (2002), Mining molecular fragments : Finding relevant substructures of molecules, in *International Conference on Data Mining (ICDM 2003)*, pp. 51–58, IEEE.

Bouguet, J.-Y. (2001), Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm, *Intel Corporation*, *2*, 3.

Boyd, S. P., and L. Vandenberghe (2004), *Convex optimization*, Cambridge university press.

Boykov, Y., and G. Funka-Lea (2006), Graph cuts and efficient nd image segmentation, *International Journal of Computer Vision (IJCV)*, *70*(2), 109–131.

Brendel, W., M. Amer, and S. Todorovic (2011), Multiobject tracking as maximum weight independent set, in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1273–1280, IEEE.

Bringmann, B., and S. Nijssen (2008), What is frequent in a single graph ?, in *Advances in Knowledge Discovery and Data Mining (PAKDD)*, pp. 858–863, Springer.

Brox, T., A. Bruhn, and J. Weickert (2006), Variational motion segmentation with level sets, in *European Conference on Computer Vision (ECCV)*, pp. 471–483, Springer.

Brutzer, S., B. Hoferlin, and G. Heidemann (2011), Evaluation of background subtraction techniques for video surveillance, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1937–1944, IEEE.

Bunke, H., and G. Allermann (1983), Inexact graph matching for structural pattern recognition, *Pattern Recognition Letters*, *1*(4), 245–253.

Bunke, H., P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento (2002), A comparison of algorithms for maximum common subgraph on randomly connected graphs, in *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 123–132, Springer.

Burkard, R. E., M. Dell'Amico, S. Martello, et al. (2009), *Assignment Problems, Revised Reprint*, Society for Industrial and Applied Mathematics (SIAM).

Buss, S. R. (1997), Alogtime algorithms for tree isomorphism, comparison, and canonization, in *Computational Logic and Proof Theory*, pp. 18–33, Springer.

Butt, A. A., and R. T. Collins (2013), Multi-target tracking by lagrangian relaxation to min-cost network flow, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1846–1853, IEEE.

Cai, Z., L. Wen, J. Yang, Z. Lei, and S. Z. Li (2013), Structured visual tracking with dynamic graph, in *Asian Conference on Computer Vision (ACCV)*, pp. 86–97, Springer.

Calders, T., and B. Goethals (2007), Non-derivable itemset mining, *Data Mining and Knowledge Discovery*, *14*(1), 171–206.

Cannons, K. (2008), A review of visual tracking, technical Report CSE-2008-07, York University.

Canny, J. (1986), A computational approach to edge detection, *Pattern Analysis and Machine Intelligence (PAMI)*, (6), 679–698.

Carletti, V., P. Foggia, and M. Vento (2013), Performance comparison of five exact graph matching algorithms on biological databases, in *New Trends in Image Analysis and Processing (ICIAP)*, pp. 409–417, Springer.

Caviar (2004), Context aware vision using image-based active recognition, `http://homepages.inf.ed.ac.uk/rbf/CAVIAR/`.

Chang, I., S.-Y. Lin, et al. (2010), 3d human motion tracking based on a progressive particle filter, *Pattern Recognition*, *43*(10), 3621–3635.

Chang, R.-F., C.-J. Chen, and C.-H. Liao (2004), Region-based image retrieval using edgeflow segmentation and region adjacency graph, in *International Conference on Multimedia and Expo (ICME)*, vol. 3, pp. 1883–1886, IEEE.

Chen, C., X. Yan, F. Zhu, and J. Han (2007), gapprox : Mining frequent approximate patterns from a massive network, in *International Conference on Data Mining (ICDM)*, pp. 445–450, IEEE.

Chi, Y., Y. Yang, and R. R. Muntz (2003), Indexing and mining free trees, in *International Conference on Data Mining (ICDM)*, pp. 509–512, IEEE.

Chockalingam, P., N. Pradeep, and S. Birchfield (2009), Adaptive fragments-based tracking of non-rigid objects using level sets, in *International Conference on Computer Vision (ICCV)*, pp. 1530–1537, IEEE.

Collins, R. T. (2012), Multitarget data association with higher-order motion models, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1744–1751, IEEE.

Comaniciu, D., and P. Meer (1999), Mean shift analysis and applications, in *International Conference on Computer Vision (ICCV)*, vol. 2, pp. 1197–1203 vol.2.

Comaniciu, D., V. Ramesh, and P. Meer (2003), Kernel-based object tracking, *Pattern Analysis and Machine Intelligence (PAMI)*, *25*(5), 564–577.

Conte, D., P. Foggia, C. Sansone, and M. Vento (2004), Thirty years of graph matching in pattern recognition, *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(03), 265–298.

Cordella, L. P., P. Foggia, C. Sansone, and M. Vento (2004), A (sub) graph isomorphism algorithm for matching large graphs, *Pattern Analysis and Machine Intelligence (PAMI)*, *26*(10), 1367–1372.

Damiand, G. (2001), Définition et étude d'un modèle topologique minimal de représentation d'images 2d et 3d., Thèse de doctorat, Université Montpellier II.

Damiand, G., C. De La Higuera, J.-C. Janodet, É. Samuel, and C. Solnon (2009), A polynomial algorithm for submap isomorphism, in *Graph-based Representation in Pattern Recognition (GBR)*, pp. 102–112, Springer.

De La Higuera, C., J.-C. Janodet, É. Samuel, G. Damiand, and C. Solnon (2013), Polynomial algorithms for open plane graph and subgraph isomorphisms, *Theoretical Computer Science*, *498*, 76–99.

Dinh, T. B., N. Vo, and G. Medioni (2011), Context tracker : Exploring supporters and distracters in unconstrained environments, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1177–1184, IEEE.

Diot, F., E. Fromont, B. Jeudy, E. Marilly, and O. Martinot (2012), Graph mining for object tracking in videos, in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, pp. 394–409, Springer.

Diot, F., E. Fromont, B. Jeudy, E. Marilly, and O. Martinot (2014), Unsupervised tracking from clustered graph patterns, in *International Conference on Pattern Recognition (ICPR)*, p. to appear.

Dollar, P., C. Wojek, B. Schiele, and P. Perona (2012), Pedestrian detection : An evaluation of the state of the art, *Pattern Analysis and Machine Intelligence (PAMI)*, *34*(4), 743–761.

Duan, G., H. Ai, S. Cao, and S. Lao (2012), Group tracking : exploring mutual relations for multiple object tracking, in *European Conference on Computer Vision (ECCV)*, pp. 129–143, Springer.

Endres, I., and D. Hoiem (2010), Category independent object proposals, in *European Conference on Computer Vision (ECCV)*, pp. 575–588, Springer.

Erdem, E., S. Dubuisson, and I. Bloch (2012), Fragments based tracking with adaptive cue integration, *Computer Vision and Image Understanding*, *116*(7), 827–841.

Ess, A., B. Leibe, and L. Van Gool (2007), Depth and appearance for mobile scene analysis, in *International Conference on Computer Vision, (ICCV)*, pp. 1–8, IEEE.

Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2008), The PASCAL Visual Object Classes Challenge(VOC2008) Results, http ://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

Felzenszwalb, P. F., and D. P. Huttenlocher (2004), Efficient graph-based image segmentation, *International Journal of Computer Vision (IJCV)*, *59*(2), 167–181.

Ferryman, J., A. Shahrokni, et al. (2009), An overview of the pets 2009 challenge, IEEE.

Fischler, M. A., and R. A. Elschlager (1973), The representation and matching of pictorial structures, *IEEE Transactions on Computers*, *22*(1), 67–92.

Fragkiadaki, K., and J. Shi (2011), Detection free tracking : Exploiting motion and topology for segmenting and tracking under entanglement, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 2073–2080, IEEE.

Frawley, W. J., G. Piatetsky-Shapiro, and C. J. Matheus (1992), Knowledge discovery in databases : An overview, *AI magazine*, *13*(3), 57.

Freeman, H. (1961), On the encoding of arbitrary geometric configurations, *Electronic Computers, IRE Transactions on*, (2), 260–268.

Fukunaga, K., and L. Hostetler (1975), The estimation of the gradient of a density function, with applications in pattern recognition, *IEEE Transactions on Information Theory*, *21*(1), 32–40.

Gabriel, K. R., and R. R. Sokal (1969), A new statistical approach to geographic variation analysis, *Systematic Biology*, *18*(3), 259–278.

Geng, L., and H. J. Hamilton (2006), Interestingness measures for data mining : A survey, *ACM Computing Surveys (CSUR)*, *38*(3), 9.

Glantz, R., M. Pelillo, and W. G. Kropatsch (2004), Matching segmentation hierarchies, *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(03), 397–424.

Goldberg, A. V. (1997), An efficient implementation of a scaling minimum-cost flow algorithm, *Journal of algorithms*, *22*(1), 1–29.

Gosselin, S., G. Damiand, , and C. Solnon (2011), Frequent submap discovery, in *Annual Symposium on Combinatorial Pattern Matching (CPM)*.

Grabner, H., J. Matas, L. Van Gool, and P. Cattin (2010), Tracking the invisible : Learning where the object might be, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1285–1292, IEEE.

Grundmann, M., V. Kwatra, M. Han, and I. Essa (2010), Efficient hierarchical graph-based video segmentation, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 2141–2148, IEEE.

Gu, S., and C. Tomasi (2011), Branch and track, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1169–1174, IEEE.

Gu, S., Y. Zheng, and C. Tomasi (2011), Efficient visual object tracking with online nearest neighbor classifier, in *Asian Conference on Computer Vision (ACCV)*, pp. 271–282, Springer.

Gudes, E., S. E. Shimony, and N. Vanetik (2006), Discovering frequent graph patterns using disjoint paths, *IEEE Transactions on Knowledge and Data Engineering*, *18*(11), 1441–1456.

Han, J., and M. Kamber (2006), *Data Mining : Concepts and Techniques, 2nd ed*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Harris, C., and M. Stephens (1988), A combined corner and edge detector., in *Alvey vision conference*, vol. 15, p. 50, Manchester, UK.

He, S., Q. Yang, R. W. Lau, J. Wang, and M.-H. Yang (2013), Visual tracking via locality sensitive histograms, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 2427–2434.

Holder, L. B., D. J. Cook, S. Djoko, et al. (1994), Substucture discovery in the subdue system., in *Workshop on Knowledge Discovery in Databases (KDD)*, pp. 169–180.

Horn, B. K., and B. G. Schunck (1981), Determining optical flow, in *1981 Technical Symposium East*, pp. 319–331, International Society for Optics and Photonics.

Horváth, T., J. Ramon, and S. Wrobel (2010), Frequent subgraph mining in outerplanar graphs, *Data Mining and Knowledge Discovery*, *21*(3), 472–508.

Hu, J.-S., and T.-M. Su (2007), Robust background subtraction with shadow and highlight removal for indoor surveillance, *EURASIP Journal on Applied Signal Processing*, *2007*(1), 108–108.

Huan, J., W. Wang, and J. Prins (2003a), Efficient mining of frequent subgraphs in the presence of isomorphism, in *International Conference on Data Mining (ICDM)*, pp. 549–552, IEEE.

Huan, J., W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropsha (2003b), Accurate classification of protein structural families using coherent subgraph analysis, in *Proceedings of the Ninth Pacific Symposium on Biocomputing (PSB)*, pp. 411–422.

Huan, J., W. Wang, J. Prins, and J. Yang (2004a), Spin : mining maximal frequent subgraphs from graph databases, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 581–586, ACM.

Huan, J., W. Wang, J. Prins, and J. Yang (2004b), Spin : mining maximal frequent subgraphs from graph databases, *Tech. Rep. UNC Technical Report TR04-018*.

Inokuchi, A., T. Washio, and H. Motoda (2000), An apriori-based algorithm for mining frequent substructures from graph data, in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, pp. 13–23.

Isard, M., and A. Blake (1996), Contour tracking by stochastic propagation of conditional density, in *European Conference on Computer Vision (ECCV)*, pp. 343–356, Springer.

Iverson, L. A., and S. W. Zucker (1995), Logical/linear operators for image curves, *Pattern Analysis and Machine Intelligence (PAMI)*, *17*(10), 982–996.

Jia, Y., J. Zhang, and J. Huan (2011), An efficient graph-mining method for complicated and noisy data with real-world applications, *Knowledge and Information Systems*, *28*(2), 423–447.

Jiang, C., F. Coenen, and M. Zito (2013), A survey of frequent subgraph mining algorithms, *Knowledge Engineering Review*, *28*(1), 75–105.

Jin, R., L. Liu, and C. C. Aggarwal (2011), Discovering highly reliable subgraphs in uncertain graphs, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 992–1000, ACM.

Jung, Y., H. Park, D.-Z. Du, and B. L. Drake (2003), A decision criterion for the optimal number of clusters in hierarchical clustering, *Journal of Global Optimization*, *25*(1), 91–111.

Kalal, Z., J. Matas, and K. Mikolajczyk (2009), Online learning of robust object detectors during unstable tracking, in *International Conference on Computer Vision Workshops (ICCV)*, pp. 1417–1424, IEEE.

Kalal, Z., J. Matas, and K. Mikolajczyk (2010), Pn learning : Bootstrapping binary classifiers by structural constraints, in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 49–56, IEEE.

Kalal, Z., K. Mikolajczyk, and J. Matas (2012), Tracking-learning-detection, *Pattern Analysis and Machine Intelligence (PAMI)*, *34*(7), 1409–1422.

Kalman, Rudolph, and Emil (1960), A New Approach to Linear Filtering and Prediction Problems, *Transactions of the ASME–Journal of Basic Engineering*, *82*(Series D), 35–45.

Kenji, A., S. Kawasoe, H. SAKAMOTO, H. Arimura, and S. Arikawa (2004), Efficient substructure discovery from large semi-structured data, *Transactions on Information and Systems (IEICE)*, *87*(12), 2754–2763.

Kotropoulos, C., A. Tefas, and I. Pitas (2000), Frontal face authentication using morphological elastic graph matching, *Image Processing, IEEE Transactions on*, *9*(4), 555–560.

Kramer, S., L. De Raedt, and C. Helma (2001), Molecular feature mining in hiv data, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 136–143, ACM.

Kropatsch, W. G. (1995), Building irregular pyramids by dual-graph contraction, *IEE Proceedings-Vision, Image and Signal Processing*, *142*(6), 366–374.

Kuramochi, M., and G. Karypis (2001), Frequent subgraph discovery, in *International Conference on Data Mining (ICDM)*, pp. 313–320, IEEE.

Kuramochi, M., and G. Karypis (2004), Grew-a scalable frequent subgraph discovery algorithm, in *International Conference on Data Mining (ICDM)*, pp. 439–442, IEEE.

Kuramochi, M., and G. Karypis (2006), Finding topological frequent patterns from graph datasets, in *Mining graph data*, chap. 6, John Wiley & Sons.

Lee, D.-T., and B. J. Schachter (1980), Two algorithms for constructing a delaunay triangulation, *International Journal of Computer & Information Sciences*, *9*(3), 219–242.

Lee, Y. J., J. Kim, and K. Grauman (2011), Key-segments for video object segmentation, in *International Conference on Computer Vision (ICCV)*, pp. 1995–2002, IEEE.

Li, F., T. Kim, A. Humayun, D. Tsai, and J. M. Rehg (2013), Video segmentation by tracking many figure-ground segments, in *International Conference on Computer vision (ICCV)*.

Lienhardt, P. (1991), Topological models for boundary representation : a comparison with n-dimensional generalized maps, *Computer-aided design*, *23*(1), 59–82.

Lowe, D. G. (2004), Distinctive image features from scale-invariant keypoints, *International Journal of Computer Vision (IJCV)*, *60*(2), 91–110.

Lucas, B. D., T. Kanade, et al. (1981), An iterative image registration technique with an application to stereo vision., in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 81, pp. 674–679.

Ma, T., and L. J. Latecki (2012), Maximum weight cliques with mutex constraints for video object segmentation, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 670–677, IEEE.

Malcolm, J., Y. Rathi, and A. Tannenbaum (2007), Multi-object tracking through clutter using graph cuts, in *International Conference on Computer Vision (ICCV)*, pp. 1–5, IEEE.

Martin, D., C. Fowlkes, D. Tal, and J. Malik (2001), A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in *International Conference on Computer Vision (ICCV)*, vol. 2, pp. 416–423, IEEE.

Menger, K. (1927), Zur allgemeinen kurventheorie, *Fundamenta Mathematicae*, *10*(1), 96–115.

Michael, R. G., and D. S. Johnson (1979), Computers and intractability : A guide to the theory of np-completeness, *WH Freeman & Co., San Francisco*.

Mikolajczyk, K., and C. Schmid (2005), A performance evaluation of local descriptors, *Pattern Analysis and Machine Intelligence (PAMI)*, *27*(10).

Moravec, H. P. (1979), Visual mapping by a robot rover, in *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 598–600, Morgan Kaufmann Publishers Inc.

Nalwa, V. S., and T. O. Binford (1986), On detecting edges, *Pattern Analysis and Machine Intelligence (PAMI)*, (6), 699–714.

Nejhum, S., J. Ho, and M.-H. Yang (2010), Online visual tracking with histograms and articulating blocks, *Computer Vision and Image Understanding*, *114*(8), 901–914.

Neuhaus, M., and H. Bunke (2007), *Bridging the gap between graph edit distance and kernel machines*, World Scientific Publishing Co., Inc.

Newman, M. E. (2004), Detecting community structure in networks, *The European Physical Journal B-Condensed Matter and Complex Systems*, *38*(2), 321–330.

Nijssen, S., and J. N. Kok (2004), A quickstart in frequent structure mining can make a difference, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 647–652.

Nowak, E., F. Jurie, and B. Triggs (2006), Sampling strategies for bag-of-features image classification, in *European Conference on Computer Vision (ECCV)*, pp. 490–503, Springer.

Osher, S., and J. A. Sethian (1988), Fronts propagating with curvature-dependent speed : Algorithms based on hamilton-jacobi formulations, *Journal of Computational Physics*, *79*(1), 12–49.

Oskoei, M. A., and H. Hu (2010), A survey on edge detection methods, *Tech. rep.*, Technical Report CES.

Papadakis, N., A. Bugeau, et al. (2011), Tracking with occlusions via graph cuts, *Pattern Analysis and Machine Intelligence (PAMI)*, *33*(1), 144–157.

Perona, P., and W. Freeman (1998), A factorization approach to grouping, in *European Conference on Computer Vision (ECCV)*, pp. 655–670.

Piatetsky-Shapiro, G., and C. J. Matheus (1994), The interestingness of deviations, in *Association for the Advancement of Artificial Intelligence (AAAI)*, vol. 94, pp. 25–36.

Piccardi, M. (2004), Background subtraction techniques : a review, in *Systems, Man and Cybernetics*, vol. 4, pp. 3099–3104 vol.4.

Pirsiavash, H., D. Ramanan, and C. C. Fowlkes (2011), Globally-optimal greedy algorithms for tracking a variable number of objects, in *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1201–1208.

Prado, A., B. Jeudy, E. Fromont, and F. Diot (2011), Plagram : un algorithme de fouille de graphes plans efficace, *Conférence Francophone sur l'Apprentissage Automatique (CAp)*, p. 343.

Prado, A., B. Jeudy, E. Fromont, and F. Diot (2013), Mining spatiotemporal patterns in dynamic plane graphs, *Intelligent Data Analysis (IDA)*, *17*(1), 71–92.

Ramírez, E., D. Martinez, and R. Carmona (2012), Segmentación de imágenes a color basada en el algoritmo de grabcut, *TekhnÃĺ, Revista de la Facultad de IngenierÃŋa*, (15), 21–37.

Rückert, U., and K. Stefan (2004), Frequent free tree discovery in graph data., in *Advances in Databases : Concepts, Systems and Applications*, pp. 564–570.

Samet, H. (1984), The quadtree and related hierarchical data structures, *ACM Computing Surveys (CSUR)*, *16*(2), 187–260.

Samuel, E. (2011), Recherche de motifs dans des images : apport des graphes plans, Ph.D. thesis, Université Jean Monnet de Saint Etienne.

Sebesta, K., and J. Baillieul (2012), Animal-inspired agile flight using optical flow sensing, in *Annual COnference on Decision and Control (CDC)*, pp. 3727–3734, IEEE.

Shi, J., and J. Malik (2000), Normalized cuts and image segmentation, *Pattern Analysis and Machine Intelligence (PAMI)*, *22*(8), 888–905.

Silberschatz, A., and A. Tuzhilin (1996), What makes patterns interesting in knowledge discovery systems, *IEEE Transactions on Knowledge and Data Engineering*, *8*(6), 970–974.

Solnon, C. (2010), Alldifferent-based filtering for subgraph isomorphism, *Artificial Intelligence*, *174*(12), 850–864.

Sperduti, A., and A. Starita (1997), Supervised neural networks for the classification of structures, *IEEE Transactions on Neural Networks*, *8*(3), 714–735.

Stalder, S., H. Grabner, and L. Van Gool (2009), Beyond semi-supervised tracking : Tracking should be as simple as detection, but not simpler than recognition, in *International Conference on Computer Vision Workshops (ICCV)*, pp. 1409–1416, IEEE.

Stauffer, C., and W. Grimson (2000), Learning patterns of activity using real-time tracking, *Pattern Analysis and Machine Intelligence (PAMI)*, *22*(8), 747–757.

Sun, D., S. Roth, and M. Black (2010), Secrets of optical flow estimation and their principles, in *Computer Vision and Pattern Recognition (CVPR)*, pp. 2432–2439.

Sundaram, N., T. Brox, and K. Keutzer (2010), Dense point trajectories by gpu-accelerated large displacement optical flow, in *European Conference on Computer Vision (ECCV)*, pp. 438–451, Springer.

Supancic III, J. S., and D. Ramanan (2013), Self-paced learning for long-term tracking, *Computer Vision and Pattern Recognition (CVPR)*, pp. 2379–2386.

Swain, M. J., and D. H. Ballard (1991), Color indexing, *International Journal of Computer Vision (IJCV)*, *7*(1), 11–32.

Toussaint, G. T. (1980), The relative neighbourhood graph of a finite planar set, *Pattern recognition*, *12*(4), 261–268.

Trémeau, A., and P. Colantoni (2000), Regions adjacency graph applied to color image segmentation, *IEEE Transactions on Image Processing*, *9*(4), 735–744.

Tsai, D., M. Flagg, and J. Rehg (2010), Motion coherent tracking with multi-label mrf optimization, *algorithms*.

Tsai, D., M. Flagg, A. Nakazawa, and J. M. Rehg (2012), Motion coherent tracking with multi-label mrf optimization, *International Journal of Computer Vision*, *100*(2), 190–202.

Ullmann, J. R. (1976), An algorithm for subgraph isomorphism, *Journal of the ACM (JACM)*, *23*(1), 31–42.

Umeyama, S. (1988), An eigendecomposition approach to weighted graph matching problems, *Pattern Analysis and Machine Intelligence (PAMI)*, *10*(5), 695–703.

Vanetik, N., E. Gudes, and S. E. Shimony (2002), Computing frequent graph patterns from semistructured data, in *International Conference on Data Mining (ICDM)*, pp. 458–465, IEEE.

Wang, J., Z. Zeng, and L. Zhou (2006), Clan : An algorithm for mining closed cliques from large dense graph databases, in *International Conference on Data Engineering (ICDE)*, pp. 73–73, IEEE.

White, B., and M. Shah (2007), Automatically tuning background subtraction parameters using particle swarm optimization, in *International Conference on Multimedia and Expo (ICME)*, pp. 1826–1829, IEEE.

Wörlein, M., T. Meinl, I. Fischer, and M. Philippsen (2005), *A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston*, Springer.

Wren, C. R., A. Azarbayejani, T. Darrell, and A. P. Pentland (1997), Pfinder : Real-time tracking of the human body, *Pattern Analysis and Machine Intelligence (PAMI)*, *19*(7), 780–785.

Wu, Z., and R. Leahy (1993), An optimal graph theoretic approach to data clustering : theory and its application to image segmentation, *Pattern Analysis and Machine Intelligence (PAMI)*, *15*(11), 1101–1113.

Yan, X., and J. Han (2002), gspan : Graph-based substructure pattern mining, in *International Conference on Data Mining (ICDM)*, pp. 721–724, IEEE.

Yan, X., and J. Han (2003), Closegraph : mining closed frequent graph patterns, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 286–295, ACM.

Yan, X., and J. Han (2006), Discovery of frequent substructures, in *Mining graph data*, chap. 5, John Wiley & Sons.

Yan, X., P. S. Yu, and J. Han (2004), Graph indexing : a frequent structure-based approach, in *International Conference on Management of Data (SIGMOD)*, pp. 335–346, ACM.

Yang, M., Y. Wu, and G. Hua (2009), Context-aware visual tracking, *Pattern Analysis and Machine Intelligence (PAMI)*, *31*(7), 1195–1209.

Yilmaz, A., O. Javed, and M. Shah (2006), Object tracking : A survey, *Acm computing surveys (CSUR)*, *38*(4), 13.

Yu, Q., and G. Medioni (2009), Multiple-target tracking by spatiotemporal monte carlo markov chain data association, *Pattern Analysis and Machine Intelligence (PAMI)*, *31*(12), 2196–2210.

Yu, S. X., and J. Shi (2001), Understanding popout through repulsion, in *Computer Vision and Pattern Recognition (CVPR 2001)*, vol. 2, pp. II–752, IEEE.

Zaki, M. J. (2002), Efficiently mining frequent trees in a forest, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 71–80, ACM.

Zeng, Z., J. Wang, L. Zhou, and G. Karypis (2006), Coherent closed quasi-clique discovery from large dense graph databases, in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 797–802, ACM.

Zhang, D., O. Javed, and M. Shah (2013), Video object segmentation through spatially accurate and temporally dense extraction of primary object regions, *Computer Vision and Pattern Recognition (CVPR)*, pp. 628–635.

Zhang, K., L. Zhang, and M.-H. Yang (2012), Real-time compressive tracking, in *European Conference on Computer Vision (ECCV)*, pp. 864–877, Springer.

Zhang, L., Y. Li, and R. Nevatia (2008), Global data association for multi-object tracking using network flows, *Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8.

Zhang, S., and J. Yang (2008), Ram : Randomized approximate graph mining, in *Scientific and Statistical Database Management*, pp. 187–203, Springer.

Zhang, S., J. Yang, and V. Cheedella (2007), Monkey : Approximate graph mining based on spanning trees, in *International Conference on Data Engineering (ICDE)*, pp. 1247–1249, IEEE.

Zou, Z., J. Li, H. Gao, and S. Zhang (2009), Frequent subgraph pattern mining on uncertain graph data, in *Conference on Information and Knowledge Management (CIKM)*, pp. 583–592, ACM.

Titre Fouille de graphes pour le suivi d'objets dans les vidéos

Résumé Détecter et suivre les objets principaux d'une vidéo est une étape nécessaire en vue d'en décrire le contenu pour, par exemple, permettre une indexation judicieuse des données multimédia par les moteurs de recherche. Les techniques de suivi d'objets actuelles souffrent de défauts majeurs. En effet, soit elles nécessitent que l'utilisateur désigne la cible à suivre, soit il est nécessaire d'utiliser un classifieur pré-entraîné à reconnaitre une classe spécifique d'objets, comme des humains ou des voitures. Puisque ces méthodes requièrent l'intervention de l'utilisateur ou une connaissance a priori du contenu traité, elles ne sont pas suffisamment génériques pour être appliquées aux vidéos amateurs telles qu'on peut en trouver sur YouTube. Pour résoudre ce problème, nous partons de l'hypothèse que, dans le cas de vidéos dont l'arrière plan n'est pas fixe, celui-ci apparait moins souvent que les objets intéressants. De plus, dans une vidéo, la topologie des différents éléments visuels composant un objet est supposée consistante d'une image à l'autre. Nous représentons chaque image par un graphe plan modélisant sa topologie. Ensuite, nous recherchons des motifs apparaissant fréquemment dans la base de données de graphes plans ainsi créée pour représenter chaque vidéo. Cette approche nous permet de détecter et suivre les objets principaux d'une vidéo de manière non supervisée en nous basant uniquement sur la fréquence des motifs. Nos contributions sont donc réparties entre les domaines de la fouille de graphes et du suivi d'objets. Dans le premier domaine, notre première contribution est de présenter un algorithme de fouille de graphes plans efficace, appelé PLAGRAM. Cet algorithme exploite la planarité des graphes et une nouvelle stratégie d'extension des motifs. Nous introduisons ensuite des contraintes spatio-temporelles au processus de fouille afin d'exploiter le fait que, dans une vidéo, les objets se déplacent peu d'une image à l'autre. Ainsi, nous contraignons les occurrences d'un même motif à être proches dans l'espace et dans le temps en limitant le nombre d'images et la distance spatiale les séparant. Nous présentons deux nouveaux algorithmes, DYPLAGRAM qui utilise la contrainte temporelle pour limiter le nombre de motifs extraits, et DYPLAGRAM_ST qui extrait efficacement des motifs spatio-temporels fréquents depuis les bases de données représentant les vidéos. Dans le domaine du suivi d'objets, nos contributions consistent en deux approches utilisant les motifs spatio-temporels pour suivre les objets principaux dans les vidéos. La première est basée sur une recherche du chemin de poids minimum dans un graphe connectant les motifs spatio-temporels tandis que l'autre est basée sur une méthode de clustering permettant de regrouper les motifs pour suivre les objets plus longtemps. Nous présentons aussi deux applications industrielles de notre méthode.

Title Graph mining for object tracking in videos

Abstract Detecting and following the main objects of a video is necessary to describe its content in order to, for example, allow for a relevant indexation of the multimedia content by the search engines. Current object tracking approaches either require the user to select the targets to follow, or rely on pre-trained classifiers to detect particular classes of objects such as pedestrians or car for example. Since those methods rely on user intervention or prior knowledge of the content to process, they cannot be applied automatically on amateur videos such as the ones found on YouTube. To solve this problem, we build upon the hypothesis that, in videos with a moving background, the main objects should appear more frequently than the background. Moreover, in a video, the topology of the visual elements composing an object is supposed consistent from one frame to another. We represent each image of the videos with plane graphs modeling their topology. Then, we search for substructures appearing frequently in the database of plane graphs thus created to represent each video. Our contributions cover both fields of graph mining and object tracking. In the first field, our first contribution is to present an efficient plane graph mining algorithm, named PLAGRAM. This algorithm exploits the planarity of the graphs and a new strategy to extend the patterns. The next contributions consist in the introduction of spatio-temporal constraints into the mining process to exploit the fact that, in a video, the motion of objects is small from on frame to another. Thus, we constrain the occurrences of a same pattern to be close in space and time by limiting the number of frames and the spatial distance separating them. We present two new algorithms, DYPLAGRAM which makes use of the temporal constraint to limit the number of extracted patterns, and DYPLAGRAM_ST which efficiently mines frequent spatio-temporal patterns from the datasets representing the videos. In the field of object tracking, our contributions consist in two approaches using the spatio-temporal patterns to track the main objects in videos. The first one is based on a search of the shortest path in a graph connecting the spatio-temporal patterns, while the second one uses a clustering approach to regroup them in order to follow the objects for a longer period of time. We also present two industrial applications of our method.