

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Développement d'un module d'association de tracklets pour améliorer des
traqueurs dans le suivi de piétons dans un milieu urbain**

RÉMI NAHON

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Décembre 2021

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Développement d'un module d'association de tracklets pour améliorer des
traqueurs dans le suivi de piétons dans un milieu urbain**

présenté par **Rémi NAHON**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Farida CHERIET, présidente

Gilles PESANT, membre et directeur de recherche

Guillaume-Alexandre BILODEAU, membre et codirecteur de recherche

Thomas HURTUT, membre

REMERCIEMENTS

Je tiens à remercier ceux qui m'ont aidé à arriver au bout de ce processus de maîtrise.

Avant tout autre, merci à mon colocataire Nathan d'avoir traversé cette période de Covid avec moi. Son soutien a été primordial à la complétion de ce projet.

Merci à Bilel Omrani pour nos échanges enrichissants et son appui dans le cadre scolaire pendant la durée de mon échange à Montréal.

Merci à Maïdie, Nicolas et à ma mère qui m'ont accompagné sur la phase de rédaction de ce mémoire.

Et enfin et surtout, merci à mes superviseurs pour leur patience, leur soutien et leur aide tout au long de ce projet.

RÉSUMÉ

Le suivi multi-objet est l'une des tâches principales de la vision par ordinateur qui a pour objectif de détecter les positions de divers objets dans des vidéos et de leur affecter une identité unique. Si les avancées récentes en Intelligence Artificielle ont permis un énorme gain de performance pour la phase de détection, la phase d'association, c'est-à-dire de construction de trajectoires, reste un problème épineux à résoudre notamment du fait de sa complexité combinatoire.

Nous proposons dans ce travail de recherche un module basé sur la Programmation par Contraintes (*CP*) dont la vocation est de se greffer à n'importe quel traqueur existant pour en améliorer les résultats d'association en milieu urbain.

Nous avons pris le parti de développer un algorithme modulaire dont chacune des trois parties est indépendante et pourra donc être améliorée à l'avenir ou remplacée par d'autres modules tenant les mêmes rôles.

La première phase consiste à récupérer les tracklets, ou sous-trajectoires, fournies par le traqueur initial, et à les couper dans les zones où des associations douteuses sont repérées, notamment lorsque des tracklets se croisent en se superposant, ce qui risque d'occasionner des interversions d'identité.

Lors de la deuxième phase, nous ré-associions les tracklets construites précédemment à l'aide de notre modèle de programmation par contraintes à propagation de croyances (*miniCPBP*). Nous proposons ici diverses contraintes qui affectent des scores à chacune des associations de tracklets en vertu de multiples caractéristiques telles que leurs dynamiques et la distance qui les sépare dans le temps et dans l'espace.

Enfin nous proposons un modèle d'interpolation pour combler a posteriori les trous présents dans les trajectoires finales que nous avons obtenues lors des phases précédentes.

Notre modèle obtient des résultats améliorants pour l'ensemble des traqueurs de l'état de l'art sur lesquels nous l'avons testé (gain de 3 à 4 points de *HOTA* et d'*IDF1*, métriques phares dans le domaine du suivi multi-objet). Nous exposons d'autre part une analyse détaillée de l'effet de chaque contrainte et paramètre du modèle sur les améliorations des résultats en terme de qualité du suivi.

ABSTRACT

Multiple object tracking (MOT) is one of the main tasks of Computer Vision which aims at detecting the positions of various objects in videos and assigning them a unique identity. While recent advances in machine learning have allowed a huge performance gain for the detection phase, lots of improvements remain to be made in the association phase that remains a thorny problem, especially because of its combinatorial complexity.

In this thesis, we propose a module based on Constraint Programming (CP) whose goal is to be grafted to any existing tracker in order to improve its association results in tracking pedestrians in an urban environment.

We have decided to develop a modular algorithm divided into three independent parts that can therefore be improved in future work or replaced by other modules holding the same functions. The first part consists in recovering the tracklets provided by an initial tracker and to cut them at the places where doubtful associations are spotted, especially when tracklets overlap which may cause identity switches. In the second phase, we associate the previously constructed tracklets using a Belief Propagation Constraint Programming model (miniCPBP), where we propose various constraints that assign scores to each of the tracklets based on multiple characteristics such as their dynamics or the distance between them in time and space. Finally, we propose a rudimentary interpolation model to fill the holes remaining in the trajectories we built.

Our model leads to improvements in the results for all three of the state-of-the-art trackers on which we have tested it (3 to 4 points gained of HOTA and IDF1). We also propose a detailed analysis of the effect of each constraint and parameter of the model on tracking quality.

TABLE DES MATIÈRES

REMERCIEMENTS	iii
RÉSUMÉ	iv
ABSTRACT	v
TABLE DES MATIÈRES	vi
LISTE DES TABLEAUX	ix
LISTE DES FIGURES	x
LISTE DES SIGLES ET ABRÉVIATIONS	xiii
CHAPITRE 1 INTRODUCTION	1
1.1 Contexte	1
1.2 Étapes du suivi multi-objet	1
1.3 Description des objets d'intérêt	2
1.4 Défis à relever	3
1.4.1 Problématiques physiques	3
1.4.2 Difficultés de détection	4
1.4.3 Répercussions et erreurs dans la phase d'association	6
1.5 Objectifs de recherche	6
1.6 Plan du mémoire	7
CHAPITRE 2 REVUE DE LITTÉRATURE	8
2.1 Détecteurs	8
2.2 Traqueurs	8
2.2.1 Des modèles pour représenter les détections	9
2.2.2 Modélisation et résolution de la phase d'association	17
CHAPITRE 3 DÉMARCHE DU TRAVAIL DE RECHERCHE	25
3.1 Réflexion initiale	25
3.1.1 Améliorer des traqueurs simples	25
3.1.2 Découpage en tracklets	26
3.2 Modélisation des tracklets	29

3.2.1	Caractéristiques retenues	29
3.2.2	Moyennage des vitesses et des boîtes englobantes	29
3.3	Modèle d'association des tracklets	32
3.3.1	Modélisation du problème d'association	32
3.3.2	Description des contraintes	38
3.3.3	Reconstruction des trajectoires à partir de la liste des successeurs	49
3.4	Interpolation : combler les trous dans les trajectoires	52
CHAPITRE 4 RÉSULTATS ET DISCUSSIONS		56
4.1	Métriques et scores choisis	56
4.2	Définition des concepts	56
4.2.1	Détections	56
4.2.2	Type d'erreurs	57
4.2.3	Similarité $\mathcal{S}$ et seuil α :	57
4.2.4	Classification des détections : TP, FP, FN	58
4.3	Une mesure d'apparence : MOTA et MOTP	58
4.3.1	Exactitude et précision (Accuracy and precision) :	58
4.3.2	MOTA	58
4.3.3	Erreurs de localisation : MOTP	59
4.4	Une mesure d'association : IDF1	59
4.4.1	Classification des détections selon la superposition des trajectoires	59
4.4.2	Calcul du <i>IDF1</i>	60
4.5	Le meilleur des deux mondes : HOTA	60
4.5.1	Classification selon les associations	60
4.5.2	Calcul du HOTA	61
4.6	Jeux de données étudiés : MOT Challenge	61
4.7	Calibration du modèle	62
4.7.1	Procédure de calibration	63
4.7.2	Calibration du modèle d'association	63
4.7.3	Calibrations du <i>TrackletCutter</i> et du module d'interpolation	68
4.8	Résultats et discussion	69
4.8.1	Résultats en ablation sur <i>IOU-Tracker</i> , <i>SORT</i> et <i>Tracktor</i> et discussion sur la contribution des composantes du modèle	69
4.8.2	Discussion sur la sensibilité des résultats aux paramètres des contraintes	71
4.8.3	Discussion sur la sensibilité du <i>TrackletCutter</i>	78
4.8.4	Discussion sur la sensibilité du modèle d'interpolation	80

CHAPITRE 5 CONCLUSION	86
5.1 Synthèse des travaux	86
5.2 Limitations de la solution proposée	86
5.3 Améliorations futures	88
RÉFÉRENCES	91

LISTE DES TABLEAUX

Tableau 3.1	Exemple de trajectoires avant interpolation	53
Tableau 3.2	Exemple de trajectoires après interpolation avec $maxGapSize = 5$ sur les trajectoires du tableau 3.1	54
Tableau 4.1	Valeurs par défaut des paramètres des contraintes <i>TimeDistance</i> , <i>PredictedCenterDistance</i> et <i>PredictedIOU</i>	67
Tableau 4.2	Valeurs par défaut des paramètres des contraintes <i>TimeDistance</i> , <i>PredictedCenterDistance</i> et <i>PredictedIOU</i>	68
Tableau 4.3	Résultats des différentes composantes du modèle en ablation sur trois traqueurs différents appliqués aux vidéos du MOT17	70
Tableau 4.4	Configurations des différents modules du modèle final employés dans les tests en ablation	71
Tableau 4.5	Valeurs par défaut des paramètres des contraintes <i>TimeDistance</i> , <i>PredictedCenterDistance</i> , <i>PredictedIOU</i> utilisées pour les tests de sensibilité du modèle aux contraintes	72
Tableau 4.6	Scores obtenus pour la meilleure configuration du <i>TrackletCutter</i> . .	79

LISTE DES FIGURES

Figure 1.1	Schéma de changement de pose démontrant la variation de taille de la boîte englobante selon la posture de l'individu	4
Figure 1.2	Schéma représentant des occlusions dans le suivi d'un piéton	5
Figure 2.1	Exemple de résultats renvoyés par le détecteur FRCNN sur la séquence MOT17-02 - Présence de rectangles, ou boîtes englobantes, qui encadrent les individus repérés par le détecteur dans un milieu urbain .	9
Figure 2.2	Exemple de résultats renvoyés par un traqueur pour les détections fournies par un FRCNN sur la séquence MOT17-02 - Présence de boîtes englobantes colorées dont chacune représente un objet particulier . .	10
Figure 2.3	Schéma explicatif du concept d'IOU : Intersection over Union	11
Figure 2.4	Exemple démontrant l'intérêt d'une représentation par les centres des détections plutôt que par les boîtes englobantes	12
Figure 2.5	Exemple d'histogramme des couleurs d'une image	14
Figure 2.6	Exemple de l'histogramme de gradients orientés d'une image	15
Figure 2.7	Exemple de modélisation du problème d'association par un graphe complet dont les noeuds représentent les détections	18
Figure 2.8	Exemple de modélisation du problème d'association par un graphe N-parti dont chaque partie représente les détections d'une trame, et d'une résolution en succession de graphes bipartis	19
Figure 2.9	Exemple de modélisation du problème d'association par un graphe biparti dont les deux parties représentent chacune l'ensemble des détections et les arêtes le lien de succession dans une trajectoire	20
Figure 3.1	Exemple occasionnant une scission de trajectoire pour <i>IOU-Tracker</i> : détection manquante	26
Figure 3.2	Exemple occasionnant une scission de trajectoire pour <i>IOU-Tracker</i> : détection partielle	27
Figure 3.3	Exemple d'application du découpeur de tracklets <i>TrackletCutter</i> avec un seuil de coupe $T_{TC} = 0.3$	28
Figure 3.4	Exemple de changements de taille et de forme d'une boîte englobante dans une situation d'occlusion dus au passage de l'objet d'intérêt derrière un obstacle	30
Figure 3.5	Exemple de changement de taille et de forme d'une boîte englobante dus aux changements de posture de l'individu suivi	31

Figure 3.6	Exemple de modélisation du problème d'association par graphe biparti associant les tracklets à leur successeur	33
Figure 3.7	Exemple d'arbre des solutions dans un cas à deux tracklets	34
Figure 3.8	Exemple d'arbre des solutions dans un cas à deux tracklets	35
Figure 3.9	Visualisation du format du score appliqué à chacune des contraintes du modèle avec $max_{score} = 0.85$ et $min_{score} = 0.2$ pour faciliter la visualisation	41
Figure 3.10	Exemple d'application de la contrainte AngleDifference : Visualisation de l'écart angulaire entre les directions d'une tracklet et de deux candidats à sa succession	44
Figure 3.11	Exemple d'application de la contrainte PredictedIOU : Visualisation de la projection de la dernière boîte englobante d'une tracklet sur la trame de début de ses successeurs envisagés	47
Figure 3.12	Exemple d'application de la contrainte PredictedCenterDistance : Visualisation de la projection du centre de la dernière détection d'une trajectoire sur la trame de début de ses successeurs envisagés	49
Figure 3.13	Exemple de modélisation du problème d'association par graphe binaire associant les tracklets à leur successeur	51
Figure 3.14	Exemple d'application du modèle d'interpolation dans un cas simple .	55
Figure 4.1	Exemple de carte de chaleur des scores HOTA pour une contrainte fictive X	64
Figure 4.2	Cartes de chaleur sur l'ensemble des vidéos pour les contraintes <i>AngleDifference</i> , <i>TimeDistance</i> et de <i>TimeDistance</i> utilisées seules	65
Figure 4.3	Cartes de chaleur sur l'ensemble des vidéos pour les contraintes <i>PredictedCenterDistance</i> , <i>PredictedIOU</i> utilisées seules	66
Figure 4.4	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>PredictedCenterDistance</i> utilisée seule	72
Figure 4.5	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>PredictedCenterDistance</i> utilisée en conjonction avec <i>PredictedIOU</i>	73
Figure 4.6	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>PredictedCenterDistance</i> utilisée en conjonction avec <i>TimeDistance</i>	74
Figure 4.7	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>PredictedIOU</i> utilisée seule	75

Figure 4.8	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>PredictedIOU</i> utilisée en conjonction avec <i>PredictedCenterDistance</i>	76
Figure 4.9	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>PredictedIOU</i> utilisée en conjonction avec <i>TimeDistance</i>	77
Figure 4.10	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>TimeDistance</i> utilisée seule	78
Figure 4.11	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>TimeDistance</i> utilisée en conjonction avec <i>PredictedCenterDistance</i>	79
Figure 4.12	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>TimeDistance</i> utilisée en conjonction avec <i>PredictedIOU</i>	80
Figure 4.13	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>AngleDifference</i> utilisée seule	81
Figure 4.14	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>AngleDifference</i> utilisée en conjonction avec <i>TimeDistance</i> et <i>PredictedCenterDistance</i>	82
Figure 4.15	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>AngleDifference</i> utilisée en conjonction avec <i>TimeDistance</i> et <i>PredictedIOU</i>	82
Figure 4.16	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>SpeedNormDifference</i> utilisée seule	83
Figure 4.17	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>SpeedNormDifference</i> utilisée en conjonction avec <i>TimeDistance</i> et <i>PredictedCenterDistance</i>	83
Figure 4.18	Cartes de chaleur des résultats <i>HOTA</i> de l'ensemble des vidéos de MOT17 pour <i>SpeedNormDifference</i> utilisée en conjonction avec <i>TimeDistance</i> et <i>PredictedIOU</i>	84
Figure 4.19	Score <i>HOTA</i> sur l'ensemble des vidéos du MOT17 en faisant varier les valeurs du seuil du TrackletCutter	84
Figure 4.20	Score <i>HOTA</i> sur l'ensemble des vidéos du MOT17 en faisant varier la taille maximale des trous comblés par l'interpolation	85
Figure 4.21	Score <i>HOTA</i> de chacune des vidéos du MOT17 en faisant varier la taille maximale des trous comblés par l'interpolation	85

LISTE DES SIGLES ET ABRÉVIATIONS

CP	Constraint Programming
BP	Belief Propagation
IOU	Intersection over Union
MOTA	Multi Object Tracking Accuracy
MOTP	Multi Object Tracking Precision
IDSW	Identity Switches
TP	True Positives
FN	False Negatives
FP	False Positives
gtDets	ground truth Detections
prDets	predictor Detections
gtID	ground truth ID
prID	predictor ID

CHAPITRE 1 INTRODUCTION

1.1 Contexte

Le suivi multi-objet est un sujet en plein essor qui présente de nombreuses solutions dont la performance a largement été améliorée par les avancées récentes en terme d'intelligence artificielle. Cette tâche consiste à localiser l'ensemble des objets d'intérêt d'une vidéo et à les suivre sur toute la durée de leur présence dans celle-ci, c'est-à-dire affecter et maintenir un identifiant unique à chacun de ces objets. Elle est dans l'écrasante majorité des cas résolue par le paradigme du "tracking-by-detection" (ou suivi par détections) qui consiste à séparer le problème en deux tâches distinctes, celles de la détection et de l'association.

Le suivi multi-objet peut être utile à des visées d'étude de comportements de déplacements de toute sorte d'objets d'intérêt, qu'il s'agisse de piétons ou de véhicules par exemple, dont l'observation des comportements dangereux peut permettre de modifier les infrastructures et réglementations sur le long terme pour limiter les accidents. Il est aussi utilisé en temps réel dans les domaines de la surveillance, militaire et aussi évidemment dans celui des véhicules autonomes qui doivent, en temps réel, être capables de détecter les objets qui les entourent, leur nature et de suffisamment prédire leur comportement pour pouvoir limiter autant que possible les risques pour les passagers et les autres usagers de la route. Cet outil peut aussi s'avérer utile dans le cadre de la reconnaissance de visages ou de plaques d'immatriculation dans la mesure où si la reconnaissance de l'objet n'est pas possible dans une trame donnée, elle peut l'être plus facilement dans des trames subséquentes.

1.2 Étapes du suivi multi-objet

Le suivi multi-objet est généralement décomposé en deux phases distinctes, celle de la **détection** et celle de l'**association** que l'on peut décrire comme suit :

- **La phase de détection** consiste à repérer et à classifier dans chacune des *trames* (c'est comme cela que l'on se réfère aux images individuelles d'une vidéo) l'ensemble des *objets d'intérêt* présents dans celle-ci. Les objets d'intérêt représentent l'ensemble des classes d'objets que l'on souhaite suivre. Dans le cadre de cette étude par exemple, il s'agit des piétons présents dans la vidéo (mais cela peut correspondre à n'importe quel type d'objet). Cela peut représenter une première difficulté et de nombreux types de méthodes existent. Les algorithmes qui remplissent cette tâche seront appelés *détecteurs* dans la suite du mémoire.

- **La phase d'association** consiste à construire les trajectoires des objets d'intérêt, c'est-à-dire à affecter à chaque objet un identifiant unique qu'il doit conserver de la première trame dans laquelle il apparaît à celle qui précède sa disparition de la vidéo. C'est sur cette phase que nous allons nous concentrer tout au long de ce mémoire. Cette phase peut donc être représentée comme un problème d'optimisation combinatoire dans lequel il convient d'affecter à chaque détection un identifiant. On cherche alors à définir (ou à laisser l'algorithme définir) des caractéristiques qui permettent de rendre compte du réalisme de l'association de détections entre elles. On cherche à répondre au mieux à la question "Ces deux détections correspondent-elles au même objet?". Les algorithmes qui remplissent cette tâche seront appelés *traqueurs* dans la suite du mémoire.

1.3 Description des objets d'intérêt

Pour pouvoir associer efficacement les objets d'intérêt entre chaque trame, il est utile, une fois qu'ils sont détectés, de conserver un certain nombre de caractéristiques descriptives pour les représenter et les comparer entre eux. Le plus souvent, le détecteur fournit au minimum une **boîte englobante** (ou *bounding box*) qui encadre chaque objet d'intérêt. Ainsi la description la plus simple de chaque détection correspond au vecteur (x, y, w, h) où :

- x correspond à l'abscisse du coin haut-gauche de la boîte englobante
- y correspond à l'ordonnée de ce coin
- w (pour *width*) correspond à sa largeur
- h (pour *height*) correspond à sa hauteur

On se base ensuite principalement sur deux types de modèles pour décrire ces objets plus précisément, les modèles de mouvement et ceux d'apparence, que l'on peut distinguer comme suit :

- **Les modèles de mouvement** consistent à décrire la détection à partir de sa position décrite dans le vecteur précédent. On suppose qu'un même objet dans des trames peu espacées aura tendance à conserver une forme, des dimensions et une position proches d'une trame à une autre (ce qui est d'autant plus vrai que les caméras fournissent des trames espacées de quelques centièmes de seconde) et que les détecteurs sont particulièrement performants aujourd'hui. De plus certains modèles s'appuient sur des caractéristiques telles que la vitesse ou l'accélération qu'on peut extraire dès lors qu'on a déjà associé des détections entre elles.
- **Les modèles d'apparence** sont composés de caractéristiques visuelles de la détection, qu'il s'agisse de contours, de textures, de couleurs, de contrastes. Comme pour

les modèles de mouvement, on suppose que l'apparence d'un objet ne change pas du tout au tout d'une trame à la suivante. Les modèles les plus simples font donc l'hypothèse d'une continuité ou de changements progressifs. D'autres vont jusqu'à rechercher des motifs qui se répètent. Par exemple, dans le cas de la marche d'un piéton, le balancement des bras peut aider à reconnaître un individu.

1.4 Défis à relever

L'objectif du suivi multi-objet est donc d'obtenir une représentation aussi proche de la réalité que possible de la position des objets d'une ou plusieurs classes spécifiées tout au long d'une vidéo et d'être capable à chaque instant de suivre et de différencier chacun de ces objets. Nous détaillerons de façon formelle au chapitre 4 les différents types d'erreurs auxquels on peut faire face mais nous devons en parler ici pour pouvoir comprendre les défis à relever pour mener à bien notre tâche de suivi. Par simplicité, nous détaillerons ici un certain nombre de cas concernant uniquement le suivi de piétons, mais ils peuvent s'appliquer à de nombreux types d'objets.

1.4.1 Problématiques physiques

Les objets d'intérêt suivis se déplaçant dans le monde physique, leur observation est soumise à diverses problématiques :

- Les êtres humains, lorsqu'ils se déplacent, ont tendance à changer de posture, notamment en balançant les bras ce qui change assez drastiquement leur apparence. Ainsi on va parler de **déformation** de l'objet d'intérêt, ce qui complique le suivi : en effet si on caractérise l'objet par le fait qu'il est en forme de croix (au moment où il a un bras devant et l'autre derrière lui), on aura du mal à le reconnaître lorsque ses bras passent le long de son buste. Comme on peut le voir sur la figure 1.1, selon la phase de la course de l'individu, la taille de la boîte englobante qui l'encadre peut être augmentée de moitié.
- Lorsque l'individu suivi se rapproche de la caméra (ou resp. s'en éloigne), sa taille augmente (ou resp. diminue) dans l'image. On parle alors de **changement d'échelle**, ce qui rend impossible dans de nombreux cas d'affecter la taille comme une caractéristique fixe pour un objet.
- Selon les sources de lumière dans la vidéo, un même individu peut connaître au cours de son déplacement des **variations d'illumination** qui vont agir sur son apparence en changeant notamment la répartition des couleurs par lesquelles on peut le décrire.
- Les **occlusions** représentent la difficulté qui reviendra le plus souvent tout au long de

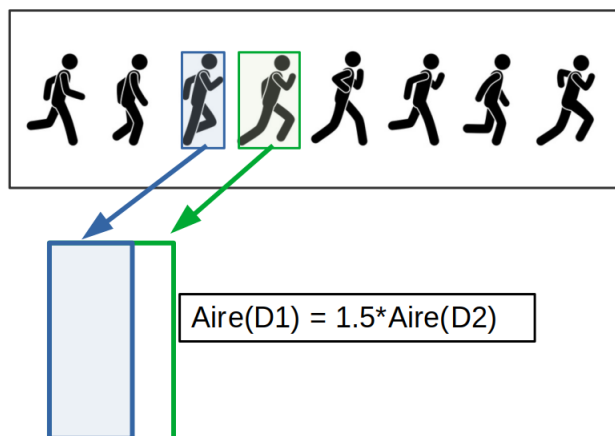


Figure 1.1 Schéma de changement de pose démontrant la variation de taille de la boîte englobante selon la posture de l'individu

ce mémoire et à laquelle on s'intéressera le plus. En effet, dans le cas de piétons, on peut facilement envisager des cas où un individu passe devant un autre, le dissimulant en partie. De même, tout objet positionné entre la caméra et un individu peut poser problème en le cachant en partie ou complètement. Ces occlusions mènent à différents écueils selon les cas. Si l'individu est suffisamment dissimulé, il peut simplement ne pas être détecté, auquel cas on a une *détection manquante*. C'est le cas dans la figure 1.2, dans laquelle l'individu est perdu par le détecteur lorsqu'il passe derrière l'arbre à ($t=4$), comme on peut le noter en l'absence de boîte englobante orange à cette trame. Un traqueur qui associerait les détections trame par trame serait contraint d'arrêter la trajectoire à ($t=3$) et d'en commencer une autre à ($t=5$), créant par la même un nouvel objet.

Si maintenant l'individu n'est dissimulé que partiellement, on peut selon les cas soit obtenir une description aux bonnes dimensions mais qui comportera l'objet obstruant la vue dans ses caractéristiques (et donc une mauvaise description de l'objet), soit ne repérer qu'une partie de l'individu dissimulé ou encore le découper en plusieurs détections. Dans la figure 1.2, on observe ainsi à ($t=9$) que seule la moitié arrière de l'individu est repérée lorsqu'il passe derrière le poteau, ce qui rendra son association avec la détection à la trame suivante plus difficile.

1.4.2 Difficultés de détection

A ces différentes difficultés physiques s'ajoutent les imperfections des divers détecteurs pour créer les erreurs suivantes au niveau des détections :

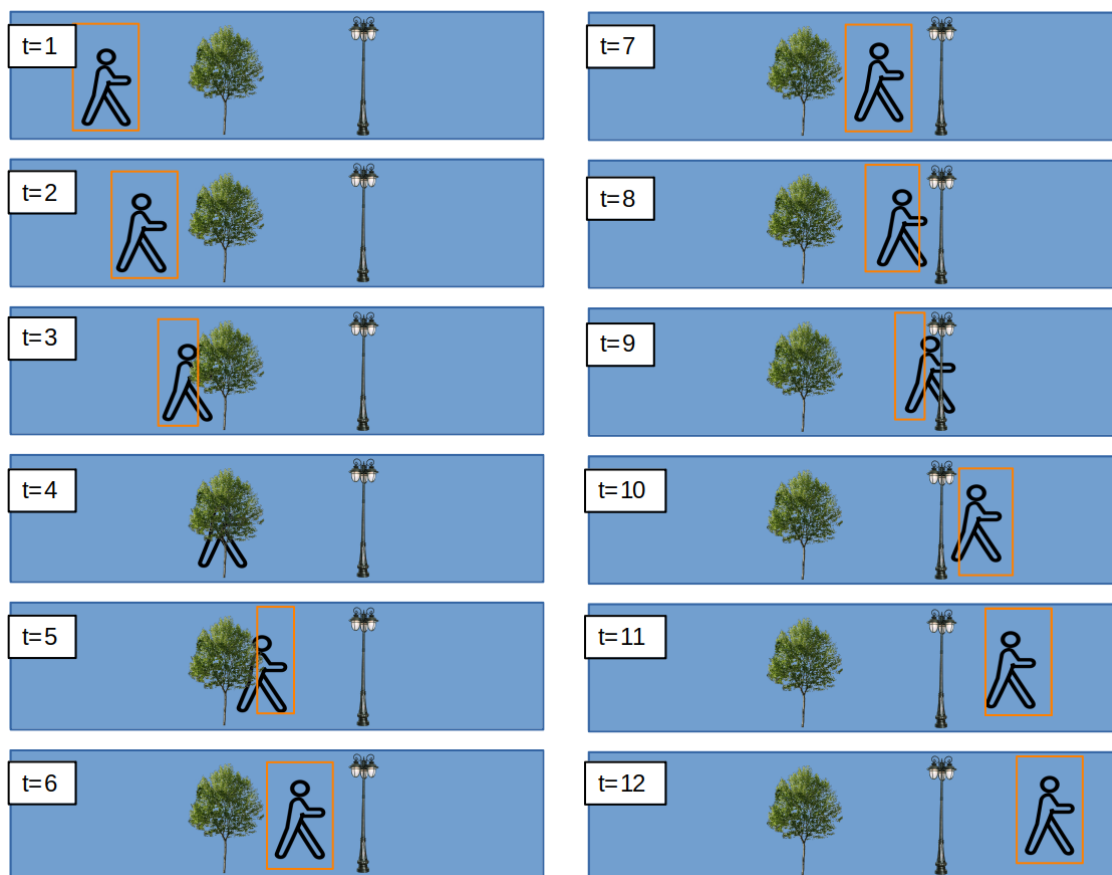


Figure 1.2 Schéma représentant des occlusions dans le suivi d'un piéton

- Un objet d'intérêt peut simplement ne pas être détecté même s'il est complètement visible. Ce cas est différent du cas des **détections manquantes** dues à des occlusions. Il s'agit d'un **faux négatif** et on pourra se reporter au chapitre 4 pour entrer plus en détail dans les notions de vrais et faux positifs et négatifs.
- Un objet d'une autre classe peut être classifié comme objet d'intérêt. En plus des erreurs grossières telles que prendre un panneau de signalisation pour un être humain, d'autres erreurs plus subtiles existent. Un problème récurrent est par exemple de différencier les êtres humains de leurs *reflets* dans divers surfaces telles que des vitres. On parle ici de **faux positifs**.
- Il arrive qu'un détecteur dédouble un objet d'intérêt. On parle de **doublons**. Ceci crée une situation particulièrement précaire dans la mesure où cela augmente le nombre d'objets détectés (et augmente par la même le nombre de faux positifs), mais où on s'expose en tentant de supprimer les doublons à supprimer des objets réels non dédoublés.

- Il arrive que le détecteur découpe un objet d'intérêt en plusieurs détections. On parle alors de **fragmentation** et on notera que ce cas est différent de celui des doublons dans la mesure où l'on n'a pas forcément de recouvrements entre les détections. Il présente toutefois des écueils similaires tels que la génération de faux positifs mais aussi les siens propres tels que le fait qu'aucune des détections obtenues ne représente efficacement l'objet suivi.

1.4.3 Répercussions et erreurs dans la phase d'association

Les difficultés précédentes présentes lors de la phase de détection sont évidemment héritées lors de la phase d'association qui y ajoute alors les siennes. Ainsi on se retrouve avec des détections manquantes, des détections superflues (qu'il s'agisse ou non réellement d'objets d'intérêt) et des descriptions de celles-ci parfois relativement éloignées de la réalité. Cela mène donc à la construction de trajectoires qui peuvent présenter les défauts suivants :

- Certaines trajectoires comportent des **trous**, c'est-à-dire des discontinuités temporelles, des trames dans lesquelles l'objet qu'elles sont supposées suivre disparaît. On conviendra aisément que cela n'est pas envisageable dans la réalité et c'est pourquoi on cherche à y remédier.
- La problématique qui va nous intéresser le plus dans cette étude est celle des **interversions d'identité** (ou *ID Switches*) auxquelles on se référera dans la suite par **IDSW**. Il s'agit des cas où deux détections sont intégrées à la même trajectoire alors qu'elles ne correspondent pas au même objet réel ou à l'inverse lorsqu'un même objet apparaît dans différentes trajectoires. Ce phénomène peut être causé par un manque de discrimination dans l'association des détections (dû soit au détecteur qui fournit de mauvais descripteurs, soit au traqueur) mais aussi par une fin prématurée de trajectoires souvent due à des occlusions.

1.5 Objectifs de recherche

Dans le cadre de ce projet, nous développons un module qui a vocation de se greffer à d'autres traqueurs (tels qu'*IOU-Tracker*, *SORT* ou *Tracktor*) pour améliorer leurs résultats notamment concernant la gestion des interversions d'identité. Nous visons à atteindre cet objectif en récupérant les trajectoires qu'ils obtiennent avant de les découper en trajectoires partielles (ou *tracklets* comme nous nous y référerons dans la suite) pour les associer grâce à notre modèle basé sur la Programmation par contraintes et la propagation de croyances [1]. Ainsi nous chercherons à obtenir une méthode efficace de découpage puis de représentation des tracklets. Nous déterminerons une façon de les associer entre elles, ainsi qu'une méthode

pour combler les trous présents dans les trajectoires résultantes. Enfin nous évaluerons nos résultats sur différents jeux de données et différents traqueurs existants.

1.6 Plan du mémoire

Dans le cadre de ce mémoire, notre étude sera développée en cinq chapitres dont la présente introduction constitue le premier. Le deuxième chapitre consistera en une revue de la littérature principalement axée sur les algorithmes de suivi multi-objet (qui se concentrent sur la phase d’association des données). Dans le troisième chapitre, nous détaillerons la démarche de ce travail de recherche ainsi qu’expliciterons le fonctionnement de notre module *CP-Associator*, de la récupération et du découpage des trajectoires initiales à l’interpolation pour combler les trous dus aux occlusions en passant par le modèle de Programmation par Contraintes employé. Le quatrième chapitre vise quant à lui à présenter les méthodes d’évaluation de notre travail, la calibration de notre modèle ainsi que de discuter l’ensemble de nos résultats appliqués aux traqueurs *IOU-Tracker* [2], *SORT* [3] et *Tracktor* [4] sur le jeu de données MOT17 [5] du MOT-Challenge [5–7], avant de faire un point sur la sensibilité de notre modèle à ses divers paramètres. Dans le cinquième chapitre, nous nous proposons enfin de conclure ce mémoire en présentant un tour d’horizon des objectifs atteints ainsi que des écueils à éviter à l’avenir tout en proposant quelques idées de solutions et d’améliorations.

CHAPITRE 2 REVUE DE LITTÉRATURE

Le suivi multi-objet est donc divisé en deux phases. La première consiste à repérer des objets d'intérêt dans l'ensemble des trames d'une vidéo. Ici nous nous intéressons à ce suivi en milieu urbain et particulièrement au suivi de piétons. Une fois les objets d'intérêts repérés, l'objectif de la seconde phase, dite *d'association*, est de les associer entre eux pour obtenir des trajectoires et c'est sur cette phase que se concentre ce travail de recherche. Avant de détailler les méthodes existantes qui permettent de réaliser cette association, un court préambule présentera rapidement les méthodes de détection qui nous ont été utiles. Nous nous étendrons ensuite sur les divers modèles employés pour représenter les détections et leur trajectoire pour faciliter l'étape d'*association* dont nous présenterons en détails diverses approches.

2.1 Détecteurs

Les méthodes de détection en IA ont récemment permis de grandes avancées dans le domaine. C'est le cas par exemple des détecteurs de la famille *R-CNN* [8], *Region-based Convolutional Neural Network* dont le principe consiste à extraire des régions d'intérêt, puis via des réseaux neuronaux convolutifs à extraire des caractéristiques de ces zones d'intérêts pour en trouver le type d'objet et la position exacte de l'objet dans l'image. Les types d'objets d'intérêt repérés par ce détecteurs peuvent par exemple être des piétons ou des automobilistes. Des améliorations ont été apportées à cette méthode, telle que détaillées dans [9] avec le *Mask R-CNN* et dans le détecteur *Faster R-CNN* [10] (auquel on se référera pour la suite en parlant de *FRCNN*). Pour ce dernier, la méthode de proposition de régions est elle-même un réseau de neurones appelé *RPN* (pour *Region Proposal Network*). Puisque la détection n'est pas le sujet de ce mémoire, nous n'élaborerons pas plus sur la détection. La figure 2.1 montre une sortie typique d'un détecteur FRCNN pour une vidéo de la base de données MOT17 du MOT-Challenge [5–7]. Les boîtes englobantes visibles dans la figure sont les sorties sur lesquelles les traqueurs sont appliqués.

2.2 Traqueurs

Si quelques méthodes actuelles ont tendance à effectuer d'une seule traite les deux tâches de détection et d'association pour construire les trajectoires [11, 12], la majorité des traqueurs se focalisent principalement sur la phase d'association des données pour obtenir à terme un résultat ressemblant à celui présenté dans la figure 2.2. Dans cette figure, les aplats rouges

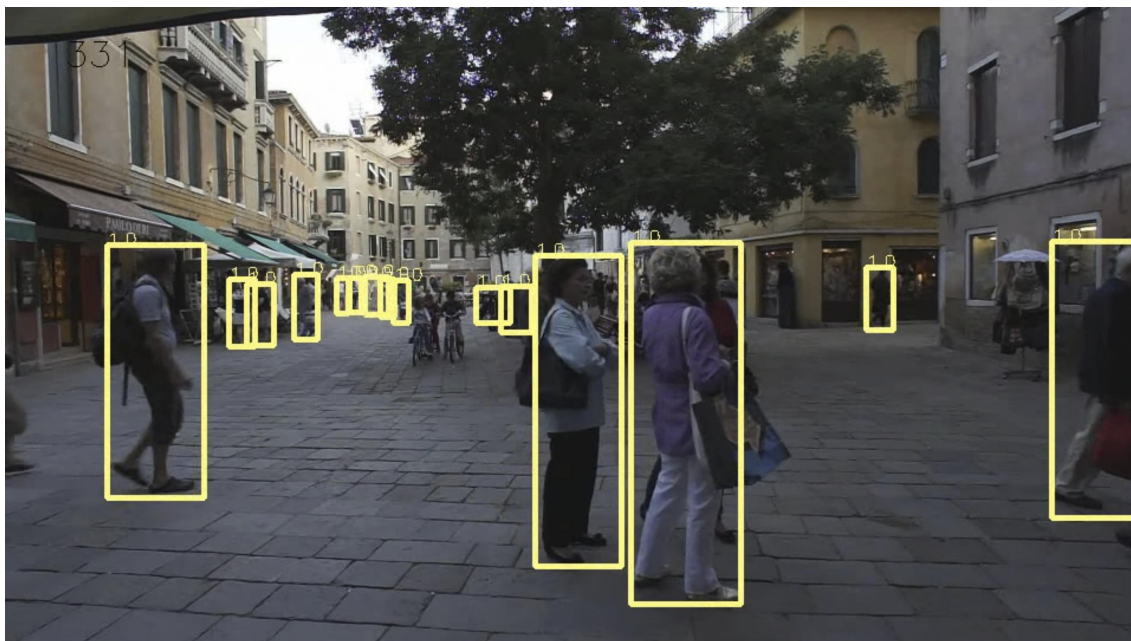


Figure 2.1 Exemple de résultats renvoyés par le détecteur FRCNN sur la séquence MOT17-02 - Présence de rectangles, ou boîtes englobantes, qui encadrent les individus repérés par le détecteur dans un milieu urbain

représentent les objets qui créent des occlusions. Pour effectuer efficacement cette phase d'association, plusieurs questions apparaissent :

- Comment représenter les détections de façon à pouvoir reconnaître et différencier les différents objets d'intérêt ?
- Comment représenter le problème d'association globale ?
- Comment explorer efficacement l'ensemble des solutions et atteindre une solution satisfaisante en un temps raisonnable ?

2.2.1 Des modèles pour représenter les détections

Pour la première question concernant la représentation des détections, pour une association efficace, on cherche généralement à maximiser une (ou plusieurs) métrique de similarité entre les détections associées, et ces métriques sont calculées à partir de descripteurs qui se divisent principalement en deux catégories :

- les modèles de mouvement
- les modèles d'apparence

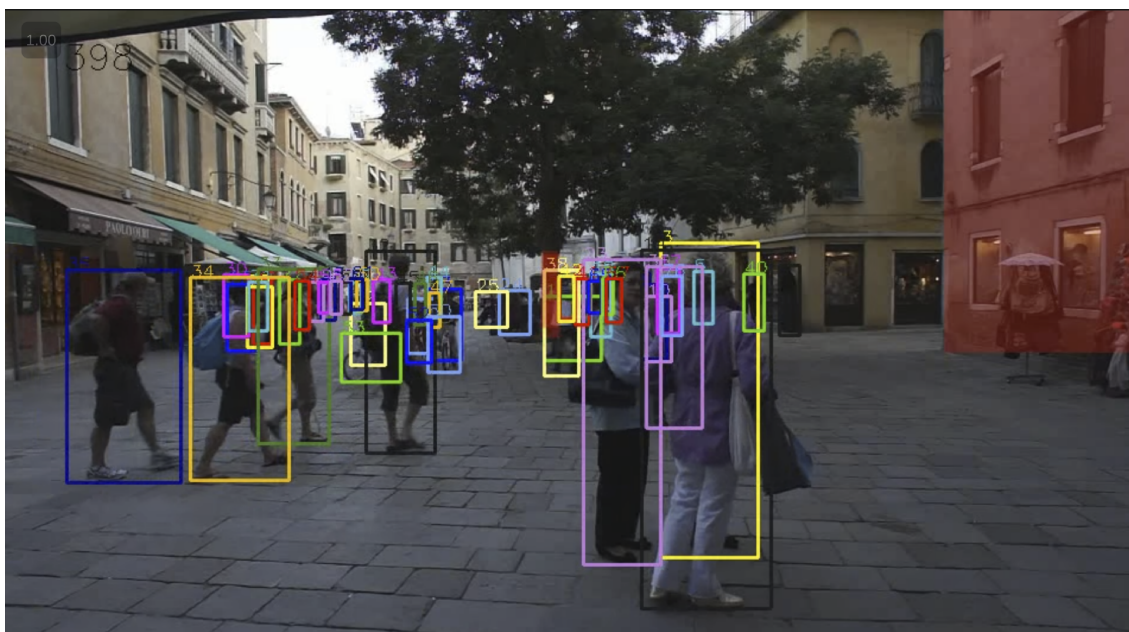


Figure 2.2 Exemple de résultats renvoyés par un traqueur pour les détections fournies par un FRCNN sur la séquence MOT17-02 - Présence de boîtes englobantes colorées dont chacune représente un objet particulier

Modèles de mouvement

Les modèles de mouvements reposent généralement sur le fait que les objets d'intérêt suivis répondent à un certain nombre de contraintes physiques :

- Il leur est impossible de se téléporter dans l'espace, c'est-à-dire de changer instantanément de position.
- Ils ne changent généralement pas de taille ou de dimensions de façon instantanée.
- Leurs vitesses de déplacement peuvent changer mais leurs changements de normes et de direction sont continus.

De plus, l'amélioration constante de la technique en matière de captation vidéo permet d'obtenir des vidéos allant jusqu'à plus de 30 trames par seconde, ce qui est beaucoup plus élevé que la capacité de déplacement des classes d'objets qui nous intéressent. Dans le cas des piétons notamment, la vitesse de marche moyenne d'un être humain est de 6.4 km/h soit 1.8 m/s, soit 6 cm par trame pour une vidéo de 30 trames par seconde.

Partant de ce constat, on peut envisager ce que l'on appellera des *modèles de position* qui font l'hypothèse qu'à l'échelle de trames successives, l'objet d'intérêt suivi est immobile, ainsi que de taille et de forme fixe. Cela permet la conception de modèles très simples et donc très rapides tels que celui sur lequel se base une grande partie de notre étude : *IOU-Tracker* [2], qui

consiste à chercher les meilleures associations de détections frame par frame en maximisant l'IOU de celles-ci.

Comme on peut le voir sur la figure 2.3, l'IOU de deux détections correspond au quotient de leurs surfaces d'intersection et d'union. IOU est un acronyme pour "Intersection over Union". Cette métrique permet de mesurer la superposition de deux détections, et pénalise les différences de taille et de position. Dans l'hypothèse où les objets d'intérêt sont quasi-statiques et ne changent pas de taille d'une frame à l'autre, cette métrique est efficace pour effectuer un suivi.

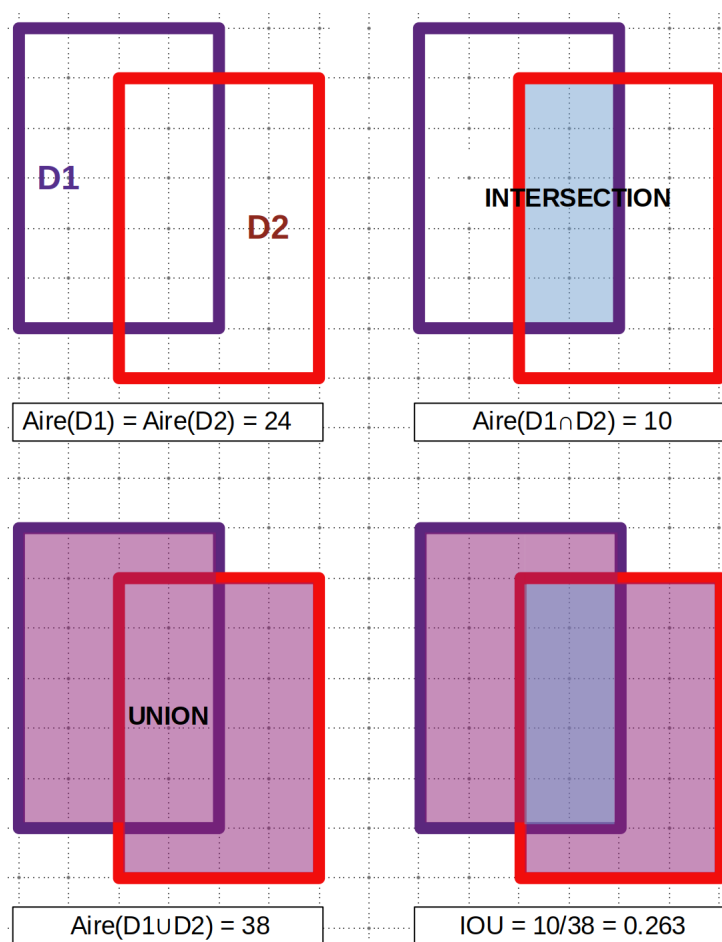


Figure 2.3 Schéma explicatif du concept d'IOU : Intersection over Union

Cette méthode est à la base du traqueur qui sera au centre de nos expérimentations : le *IOU-Tracker* [2]. Celui-ci consiste simplement à balayer l'ensemble des trames et parmi celles-ci l'ensemble des détections une à une pour trouver à chaque fois la meilleure détection à leur associer dans la trame suivante (celle maximisant l'IOU).

Pour augmenter la précision de ce genre de modèles de position, on prend souvent en consi-

dération la vitesse de déplacement de l'objet d'intérêt et parfois sa vitesse de déformation. Ainsi la façon la plus répandue d'utiliser ce paramètre de vitesse est de la considérer de norme et de direction fixes, ce qui se vérifie à l'échelle de quelques trames d'écart entre les boîtes englobantes, et de projeter alors les boîtes englobantes sur les trames suivantes et de comparer les boîtes englobantes des détections réelles avec ces projections. Par ailleurs, un travail sur le centre de la détection permet de mieux gérer la problématique des dimensions erronées des boîtes englobantes dans le cas d'occlusions notamment. C'est le choix qui est fait dans une partie de la méthode CenterTrack [11] dont nous reparlerons un peu plus loin. Dans la figure 2.4, la détection D2' est plus fautive, plus petite que la détection D2 de la réalité-terrain et un peu décalée par rapport à elle. On y observe que les IOU sont beaucoup plus impactés que l'alignement des centres ou les distances qui les séparent.

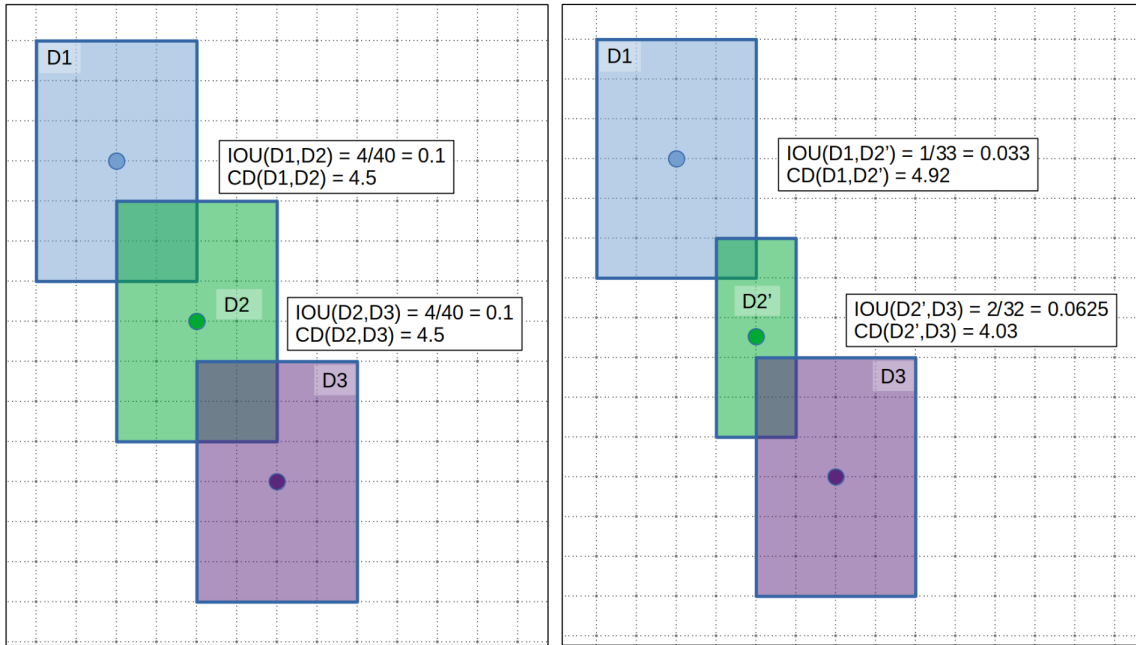


Figure 2.4 Exemple démontrant l'intérêt d'une représentation par les centres des détections plutôt que par les boîtes englobantes

Il existe des modèles de plus en plus complexes de la vitesse qui se basent notamment sur des filtres de Kalman [13] utilisés par exemple dans [14] et dans SORT (pour *Simple Online and Realtime Tracker*, pour *traqueur simple online en temps réel* en français) [3], un des traqueurs sur lesquels nous testerons notre modèle. Le filtre de Kalman se base sur le modèle plus complet de la boîte englobante décrit dans le chapitre 1, (x, y, w, h) , en y ajoutant la dérivée temporelle de chacune des variables. Pour la trame t , l'état X de chaque objet

d'intérêt est donc décrit par :

$$X(t) = (x_t, y_t, w_t, h_t, \dot{x}_t, \dot{y}_t, \dot{w}_t, \dot{h}_t) \quad (2.1)$$

On effectue une première prédiction de l'état de chacune des 8 variables X à $t + 1$ comme suit :

$$\hat{X}_t = AX_{t-1} + BU_{t-1} + \epsilon_{X_t} \quad (2.2)$$

où U correspond à l'action, ou la variable qui va avoir un effet sur la prédiction de X . A et B sont des coefficients appartenant au modèle et ϵ_{X_t} est une gaussienne qui permet de marquer l'incertitude de la prédiction. Par exemple pour la position en abscisse x_t , on aura : $\hat{x}_t = x_{t-1} + \dot{x}_{t-1} + \epsilon_{x_t}$. On ajoute à cela l'incertitude de la mesure en créant une prédiction de la mesure $\hat{Z}_t$:

$$\hat{Z}_t = C\hat{X}_t + \epsilon_{Z_t} \quad (2.3)$$

Et on compare tout cela à l'observation réelle, c'est-à-dire aux boîtes englobantes proposées par le détecteur. Une fois une association faite, on retiendra une nouvelle estimation de la variable X :

$$\hat{X}_{est_t} = \hat{X}_t + K(Z_t - \hat{Z}_t) \quad (2.4)$$

La valeur affectée à K permet de moduler le poids donné à l'observation réelle par rapport à la confiance que l'on a du modèle. D'autres modèles encore plus complexes cherchent à représenter la totalité de la dynamique des trajectoires comme dans [15].

Certains traqueurs vont aussi prendre en compte un potentiel mouvement de caméra. En effet, dans certaines vidéos, la captation est faite par une caméra mobile et la tâche de suivi peut-être simplifiée en conservant en parallèle des modèles de mouvement des objets d'intérêt, un modèle du mouvement de la caméra. C'est notamment ce propose la méthode de maximisation du coefficient de corrélation [16], que l'on retrouve dans de nombreuses méthodes [4, 17].

Modèles d'apparence Les modèles d'apparence consistent en la description de certaines caractéristiques visuelles des détections (ou plutôt de leurs boîtes englobantes) et se basent sur le fait que les objets ne changent généralement pas d'apparence instantanément, ce qui permettra a posteriori d'associer les détections avec des descripteurs similaires. Les descripteurs les plus utilisés, hors apprentissage automatique, sont les histogrammes de couleur et ceux des gradients orientés (HOG pour "Histogram of Oriented Gradients") [18].

Les *histogrammes des couleurs* [19] consistent à représenter chaque détection par un ou plusieurs histogrammes de 256 colonnes (qui représentent les niveaux d'intensité que peuvent

prendre les couleurs rouge, vert et bleu pour chaque pixel dans une image en couleurs, ou les niveaux d'intensité de gris pour une image en tons de gris). On aura un histogramme par dimension de l'espace des couleurs représenté et cet histogramme représentera pour chaque couleur (rouge, vert et bleu dans un espace RGB) la quantité ou la proportion de pixels de la détection à atteindre chaque niveau d'intensité pour la couleur étudiée. Il est possible de simplifier ces histogrammes pour limiter la quantité d'information transportée pour chaque détection en regroupant les niveaux d'intensité par groupes. On peut voir un exemple d'histogramme obtenu pour une image donnée dans la figure 2.5.

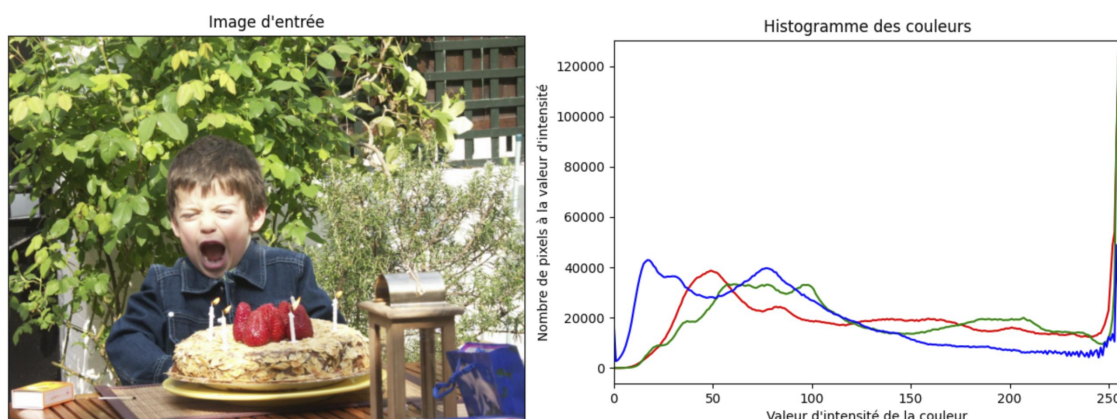


Figure 2.5 Exemple d'histogramme des couleurs d'une image

Les *histogrammes de gradients orientés* permettent de décrire les objets d'intérêt par des statistiques sur leurs gradients. Pour décrire cette méthode, nous allons nous aider de la figure 2.6 qui permet de visualiser un histogramme de ce type pour l'image originale (à gauche de la figure) :

- On calcule le gradient de l'intensité de l'image pour chaque pixel grâce à deux filtres de Sobel (un en x , un en y). Ceux-ci sont visibles pour l'ensemble des pixels au milieu de la figure. On y voit clairement qu'une sorte de contours des objets se dégage des gradients mais que, par exemple, les bougies du gâteau sont beaucoup plus visibles dans le cas du gradient horizontal que dans celui du gradient vertical. Ainsi le fait d'avoir ces deux directions de gradient permet de mieux visualiser les contours des objets dans l'image.
- On obtient pour chaque pixel une direction (parmi 9 possibilités : $0^\circ, 20^\circ, \dots, 160^\circ$) et une norme de gradient, à partir de la valeur de gradient qu'ils ont obtenu selon x et y .
- L'image est alors divisée en cellules (de 128×128 pixels pour l'image du haut à droite ou 32×32 pixels en bas) et pour chaque cellule on compile le nombre de pixels de la cellule ayant un gradient dans chacune des 9 directions pondéré par la norme de ce

gradient.

- On a alors un histogramme par cellule et celui-ci est représenté par une étoile dont la direction d'une branche représente la direction du gradient et la longueur de la branche représente le nombre de pixels de la cellule dont le gradient est dans ladite direction (pondéré par la norme de ces gradients).

Ainsi si une cellule se situe au niveau d'un aplat de couleur, comme au niveau de la veste ou du visage du petit garçon de la figure, les normes des gradients sont très faible et donc l'histogramme de la cellule est presque vide. Si la cellule se situe à l'interface entre deux aplats, sur un contour, comme au niveau du col du garçon, on observera des normes élevées regroupées dans les mêmes directions. Les arbres derrière la scène n'étant ni des aplats ni des contours ont des gradients qui partent dans tous les sens.

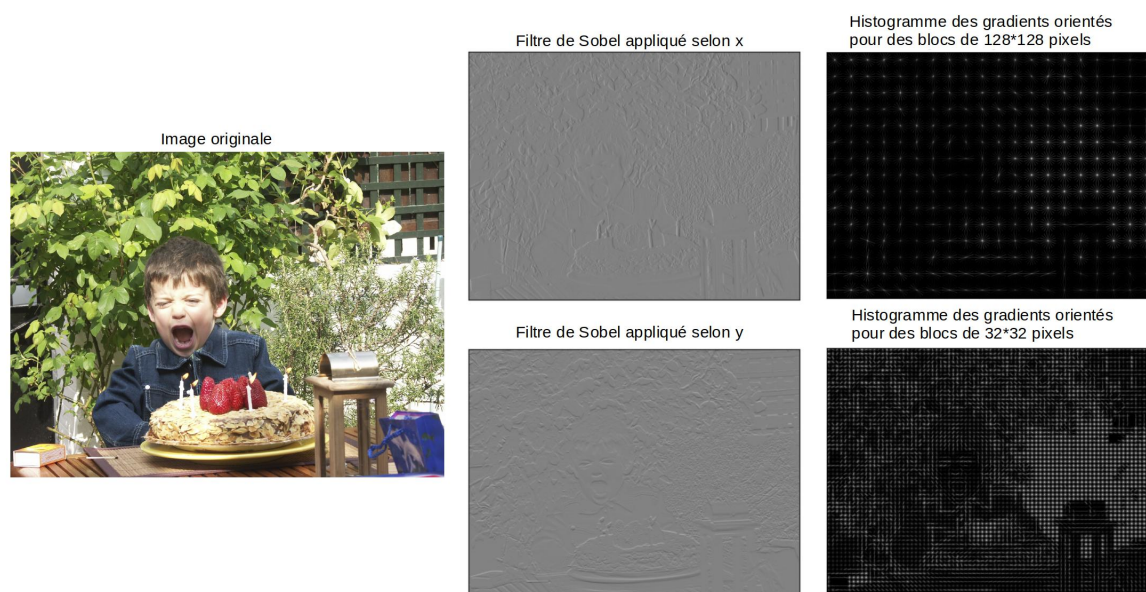


Figure 2.6 Exemple de l'histogramme de gradients orientés d'une image

D'autres méthodes telles que les matrices de covariance sont aussi utilisées pour décrire les textures présentes dans une détection. On peut aussi citer la méthode *KCF* (pour *Kernalized Correlation Filter*) [20] qui consiste à construire un filtre à partir de la détection tel que le produit de ce filtre et de la détection doit produire un résultat maximal. Ainsi en effectuant le produit de ce filtre avec les détections présentes dans les trames suivantes, on arrive à retrouver la détection la plus similaire en apparence. Cette méthode est principalement employée en *SOT* (*Single Object Tracking* ou suivi d'objet singulier en français) mais est aussi employée en suivi multi-objet notamment dans le traqueur *VIOU* [21] qui est une amélioration du *IOU-Tracker* qui va appliquer ce traqueur singulier à chacune des extrémités

des trajectoires construites par *IOU-Tracker* pour essayer de les allonger ou de les fusionner (*IOU-Tracker* étant très sensible aux occlusions).

Toutefois les méthodes d'apprentissage profond sont devenues depuis quelques années les méthodes les plus performantes pour ce qui est des modèles d'apparence. La définition de l'ensemble de caractéristiques les plus pertinentes pour décrire les détections est alors laissée à l'algorithme. Parmi les méthodes récentes qui constituent l'état de l'art, on peut compter des réseaux de neurones à convolution [22–28], ainsi que récemment des transformeurs comme dans les méthodes *TransCtr* et *MOTer* [29], des réseaux de neurones à mélange de densité (Mixture Density Network) comme dans [30]. La méthode *Tracktor* proposée par [4] consiste justement en une régression de boîtes englobantes d'une trame à l'autre pour prolonger les trajectoires. Si la boîte régressée de la trame $t - 1$ à la trame t a un *IOU* supérieur à un certain seuil, la détection est ajoutée à la trajectoire et sinon on en commence une nouvelle. Sur le même modèle, *CenterTrack* [11] fonctionne de façon similaire mais en travaillant sur le centre des détections, ce qui lui permet d'être plus robuste aux occlusions comme on l'a vu avec la figure 2.4.

Modèles mixtes Un grand nombre des algorithmes existants aujourd'hui sont mixtes et emploient à la fois des modèles de mouvement et des modèles d'apparence. Parmi eux, on trouve évidemment encore de nombreux algorithmes d'apprentissage profond qui mélangent caractéristiques visuelles à celles des mouvements. De nombreux modèles emploient des méthodes de régression se basant sur celle proposée par *Tracktor* tout en y ajoutant des méthodes d'apprentissage profond plus orientées sur l'apparence [17, 23]. Il existe aussi une méthode améliorant *SORT* [28] par l'ajout d'un réseau de neurones qui travaille à la construction de descripteurs visuels. De même on peut trouver cette idée de modèle mixte poussée à l'extrême dans les méthodes qui s'appuient sur la minimisation de fonctions d'énergies (dont on décrit plus en détail le fonctionnement dans la partie 2.2.2). Dans ces méthodes [31, 32], la présence de multiples fonctions d'énergie dont on cherche à minimiser la somme permet d'augmenter à volonté la quantité de caractéristiques sur lesquelles se baser pour représenter les trajectoires. Comme on le verra plus loin, il y est fait appel à la fois à une énergie de mouvement E_{dyn} qui représente donc un modèle de mouvement mais aussi à une énergie d'apparence dans [32] ou encore à une énergie de persistance qui minimise le nombre de trous présents dans les trajectoires.

Une pratique très employée réside aussi dans ce que l'on appelle la ré-identification. Celle-ci, particulièrement utile dans le cadre de caméras multiples, a pour priorité de retrouver des individus repérés au préalable à partir de diverses caractéristiques (le plus souvent visuelles). La méthode a fait ses preuves dans le cadre du suivi multi-objet notamment dans le cadre de

VIOW [21] décrit précédemment, mais aussi dans [23] et pour le traqueur *UnsupTrack* [33]. Ce dernier s'appuie aussi sur les résultats obtenus par des traqueurs simples, tels qu'*IOU-Tracker* [2], *SORT* [3] ou *Tracktor* [4] mais sans les conserver a posteriori. En effet, il récupère les identifiants de trajectoires affectés aux détections par le traqueur initial, entraîne un réseau de neurones à convolution, *ResNet50* [34], à affecter ces labels aux détections auxquelles ils appartiennent, puis reprend les détections vierges et effectue cette affectation lui-même. Ainsi les trajectoires découpées peuvent être fusionnées lorsque les descripteurs visuels qu'elles possèdent sont suffisamment similaires.

2.2.2 Modélisation et résolution de la phase d'association

Une fois les modèles de représentation des détections choisis, on va chercher à les associer entre elles pour construire nos trajectoires. L'objectif est alors d'affecter un identifiant de trajectoire à chaque détection pour s'assurer de suivre l'ensemble des objets d'intérêt de leur entrée dans le champ de la caméra à leur sortie.

Modélisation du problème d'association

Il convient d'abord de modéliser ce problème d'association et la façon la plus simple de le faire est de construire un graphe $G(D, A)$ tel que :

- D est l'ensemble des détections retenues par le détecteur
- $A = D \times D$ est l'ensemble des paires de détections

Et le travail consiste à trouver une bonne façon d'évaluer le poids de chacune des arêtes de A , pour construire le meilleur graphe orienté $H(D, B)$ tel que :

- H est un sous-graphe couvrant de G : c'est-à-dire que tous les sommets (détections) appartiennent à H et B appartient à A .
- H est acyclique, de sorte à ce qu'une détection ne fasse pas partie de ses propres successeurs.
- tout sommet de H est au maximum de degré entrant 1 et de degré sortant 1 : c'est-à-dire que chaque détection ne peut avoir qu'un prédécesseur et qu'un successeur.

Dans l'exemple de la figure 2.7, on a à gauche notre graphe complet G dont les noeuds représentent chacun une détection. Et après résolution, on obtient à droite H (en rouge et noir gras) qui contient l'ensemble des détections, est orienté, sans cycle et dont chacun des sommets est de degré (entrant et sortant) inférieur ou égal à 1. Ainsi les sommets de degré entrant nul sont le début des trajectoires et celles-ci se terminent dans les sommets de degré sortant nul. On obtient donc après résolution deux trajectoires : $[D_1, D_2, D_5, D_6]$ et $[D_3, D_4, D_7, D_8]$.

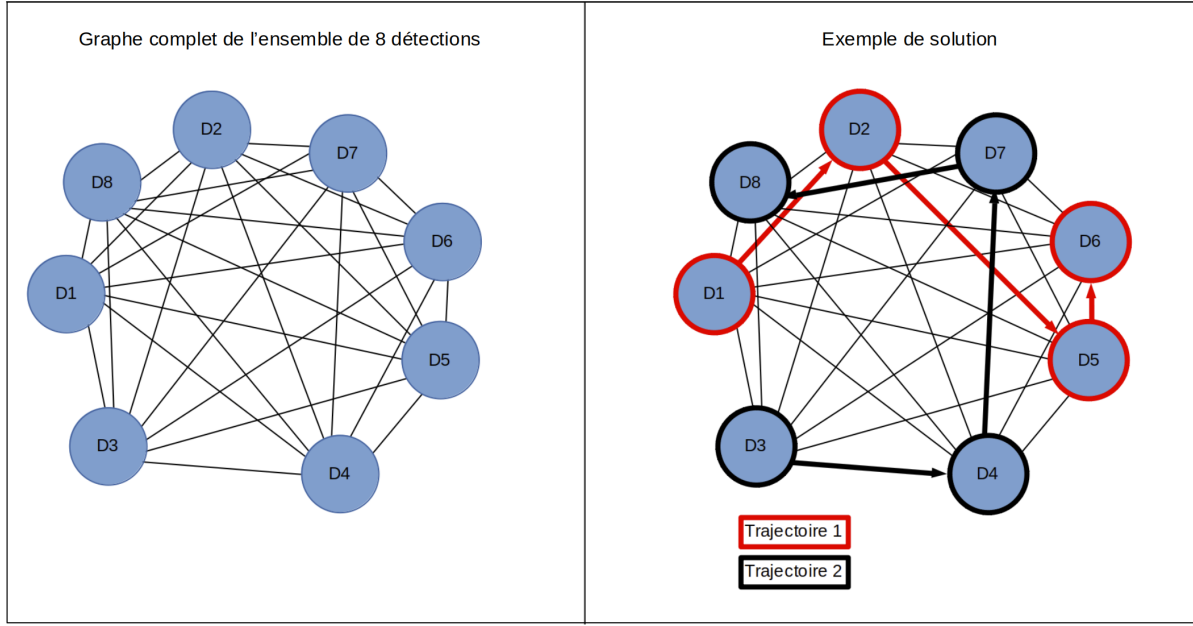


Figure 2.7 Exemple de modélisation du problème d'association par un graphe complet dont les noeuds représentent les détections

Les algorithmes *online* (dont on verra le fonctionnement plus en détail plus loin dans la section 2.2.2) utilisent souvent une modélisation plus contrainte qui consiste à résoudre pour chaque paire de trames (t, u) successives un graphe biparti $G(D_t, D_u, A_{t,u})$ tel que :

- D_t est l'ensemble des détections présentes dans la trame t
- D_u est l'ensemble des détections présentes dans la trame u
- $A_{t,u}$ est l'ensemble des paires de détections entre les détections dans t et celles dans u

Dans l'exemple de la figure 2.8, on part donc d'un graphe N -parti (pour N trames) visible dans l'image (a) de sorte que pour toute trame d'identifiant compris entre 1 et $N - 1$, les détections qui lui appartiennent ne sont reliées qu'à celles de la trame suivante et celles de la trame précédente. Il est donc aisé de diviser ce graphe en $N - 1$ graphes bipartis. Les meilleures associations sont effectuées dans chaque sous-graphe biparti avant de recomposer des trajectoires. Ainsi on résout le problème d'association pour chaque graphe biparti (comme montré pour deux graphes bipartis en (c)) et enfin on associe toutes ces résolutions pour construire des trajectoires (comme on peut le voir en (d)). Selon le traqueur, les détections restantes, c'est-à-dire qu'elles n'appartiennent à aucune trajectoire, sont le plus souvent supprimées ou déclarées comme étant elles-mêmes des trajectoires (d'une seule détection). On notera toutefois que cette façon de modéliser le problème ne permet pas de gérer les détections manquantes dans une ou plusieurs trames.

Une autre modélisation équivalente à la première (en ce qu'elle permet encore l'association

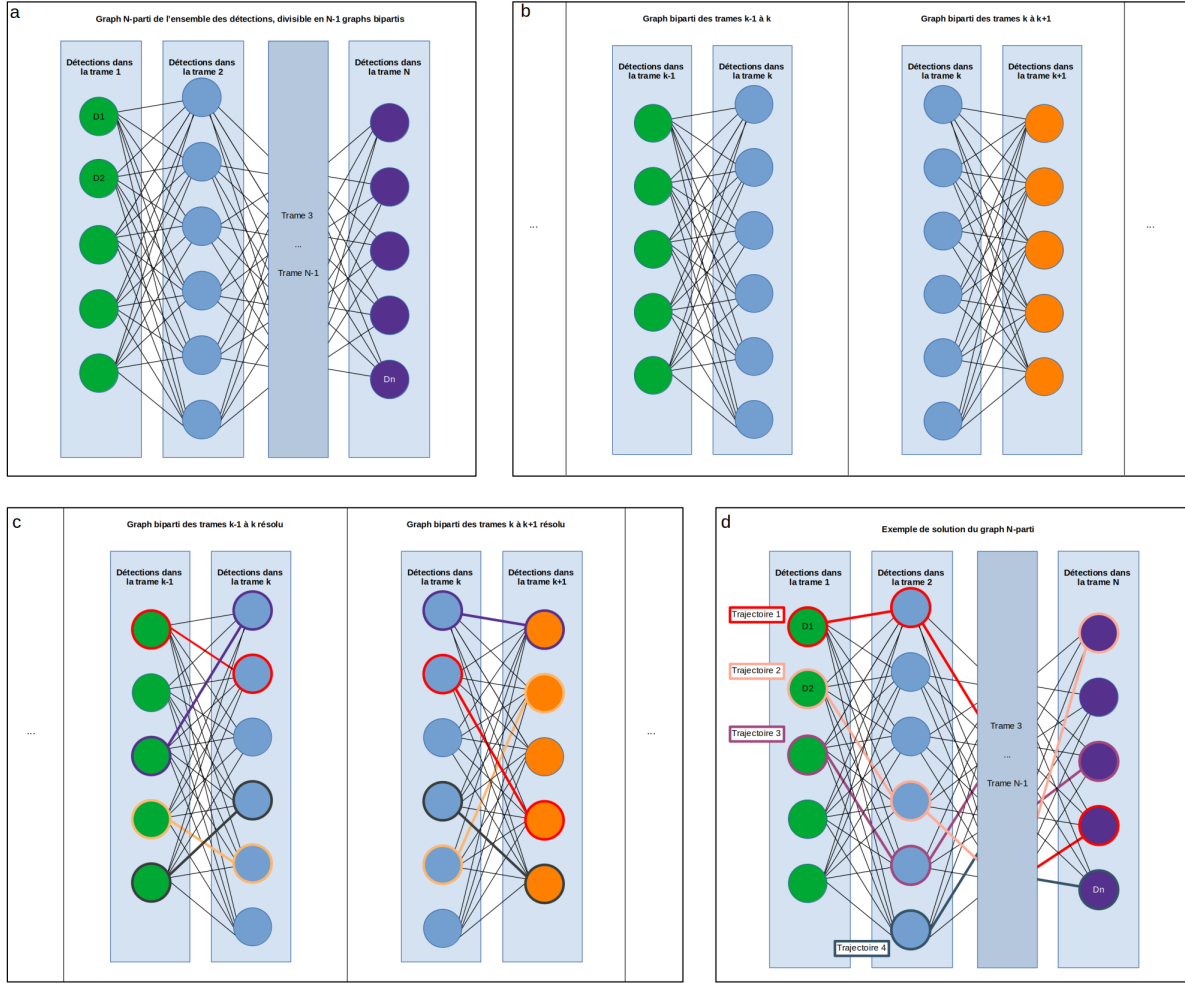


Figure 2.8 Exemple de modélisation du problème d'association par un graphe N-parti dont chaque partie représente les détections d'une trame, et d'une résolution en succession de graphes bipartis

de toutes les détections entre elles contrairement à la précédente) [35,36] est celle d'un *graphe biparti* $G(P, S, A)$ tel que :

- P est l'ensemble des détections *prédécesseurs*.
- S est l'ensemble des détections *successeurs*.
- $A = P \times S$ est l'ensemble des paires prédécesseur-successeur.

La différence avec les cas du graphe N-parti décrit dans la figure 2.8 est qu'ici il reste possible d'associer des détections éloignées de plusieurs trames. Dans le cas précédent, toute détection manquante sur une trame entraînera une fin de trajectoire ou une association erronée. C'est notamment le cas pour *IOU-Tracker* [2].

Le travail est le même, on va chercher à obtenir le meilleur sous-graphe couvrant de G .

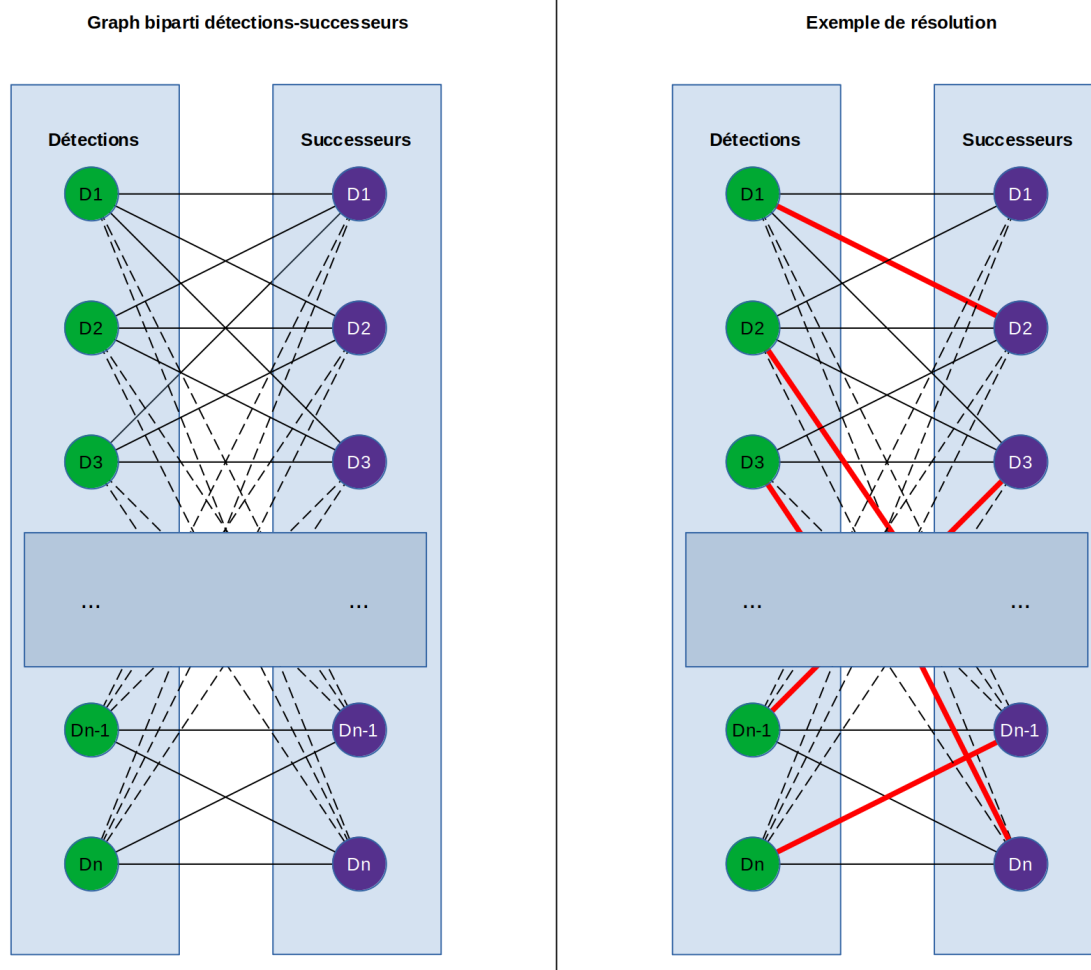


Figure 2.9 Exemple de modélisation du problème d'association par un graphe biparti dont les deux parties représentent chacune l'ensemble des détections et les arêtes le lien de succession dans une trajectoire

On en a un exemple dans la figure 2.9 et on y obtient après résolution une suite de paires (prédécesseurs-successeurs) qu'il faudra arranger en trajectoires : assez simplement, si $D1$ a pour successeur direct $D2$ qui a lui-même pour successeur $D17$, alors ces trois détections se suivent directement dans la même trajectoire.

Une autre modélisation consiste à créer (encore) un graphe biparti entre l'ensemble des détections et un ensemble d'identifiants de trajectoires. L'ensemble des détections associées à une trajectoire constituent alors les détections de ladite trajectoire. Cette modélisation a l'avantage de bien gérer les détections manquantes et de diminuer la complexité du problème, mais nécessite de fixer par avance le nombre de trajectoires attendues, ce qui n'est pas toujours possible.

Dans d'autres travaux, des modélisations similaires aux deux précédentes sont employées mais en prenant pour sommets des *tracklets* (courtes trajectoires souvent construites par d'autres traqueurs *online* tels que *IOU-Tracker* [2] ou *Tracktor* [4]). Nous détaillerons plus ce cas dans la partie 3.3.1, car il s'agit de l'axe que nous avons décidé d'emprunter dans ce travail.

Résolution du problème d'association

Avant d'entrer dans la façon dont on va résoudre le problème d'association, nous allons distinguer deux types de traqueurs dont les objectifs diffèrent et dont les façons de gérer cette résolution vont donc largement différer.

Traqueurs online et traqueurs offline, des objectifs distincts Certains effectuent la tâche d'association trame par trame dans l'ordre chronologique d'apparition de celles-ci [2,3,33]. On parle alors d'association *online* et si celle-ci se fait dans un temps suffisamment court, elle peut être appliquée en *temps réel* à des vidéos. C'est notamment le cas du traqueur auquel nous avons décidé de greffer notre module pendant la majorité de ce travail, le *IOU-Tracker* [2], ainsi que *SORT* et *Tracktor* dont nous avons détaillé les fonctionnements dans la partie 2.2.1. Ces traqueurs sont souvent très rapides (jusqu'à 100 000 trames traitées par seconde pour l'*IOU-Tracker*) ce qui les rend particulièrement éligibles à des ajouts modulaires pour les améliorer [21, 28, 33, 37].

En revanche, cette caractéristique diminue aussi leur performance dans la mesure où ils ne peuvent pas s'appuyer sur des informations disponibles dans les trames postérieures à celle étudiée. Et c'est exactement ce qui est fait dans la seconde catégorie de traqueurs, qui effectuent des associations dites *offline*. Certains vont ajouter au travail *online* du passé vers le futur un travail symétrique du futur vers le passé [17, 21] quand de nombreux autres vont chercher à résoudre le problème comme un problème d'optimisation dont la modélisation en elle-même peut varier. Ces traqueurs *offline* travaillent donc sur l'ensemble des données et peuvent donc s'appuyer sur de plus nombreuses et robustes caractéristiques. Ils ne peuvent toutefois pas travailler en temps réel dans la mesure où, par définition, les trames apparaissent les unes après les autres.

Traqueurs online Un traqueur *online* a pour objectif de construire ses trajectoires en temps réel et donc trame par trame dans leur ordre d'apparition chronologique. La méthode se doit donc d'être incrémentale, et trame par trame on ajoute des détections aux trajectoires déjà construites, on ferme des trajectoires existantes et on en crée de nouvelles. L'algorithme

utilisé doit donc déterminer un critère pour débiter une trajectoire, la fermer et pour y ajouter une détection donnée. Dans le cas le plus simple, celui d'*IOU-Tracker* [2], l'association de la trame t à la trame $t + 1$ se fait comme suit :

- Pour chaque trajectoire encore active, on cherche la détection de $t + 1$ qui a le IOU le plus élevé avec sa détection à t .
- Si leur IOU est supérieur au seuil fourni par l'algorithme, on ajoute la détection à la trajectoire. Sinon on ferme la trajectoire car on suppose qu'elle se finit à la trame précédente.
- Les détections restantes (n'étant associées à aucune trajectoire) constituent le début de nouvelles trajectoires.

Dans des cas un peu plus sophistiqués, les informations obtenues sur les trajectoires que l'on cherche à prolonger permettent de prédire des informations sur les détections que l'on cherche. C'est à ces *détections prédites* que l'on compare les détections réelles de la trame. Ainsi pour les méthodes basées sur le mouvement, on projette l'objet d'intérêt dans la trame suivante et on mesure la distance entre cet objet projeté et les détections présentes dans la trame. Dans le cas de *Tracktor*, cette mesure est l'IOU entre une boîte englobante non pas projetée mais détectée par régression de la boîte englobante que l'on avait à la trame précédente et les boîtes englobantes réelles. Dans d'autres méthodes basées sur de l'apprentissage comme *MFI_tst* [23] ou *DAN* [24] (pour "*Deep Affinity Network*"), les distances sont calculées entre des vecteurs qui prennent aussi en compte l'apparence des objets d'intérêt. Cette tendance rigide à arrêter les trajectoires dès qu'on n'arrive pas à leur trouver de bon successeur dans une trame est assez dommageable. Pour être plus robuste à ce problème, des techniques de contournement existent. Dans le cas de plusieurs méthodes précédemment citées, les trajectoires qui n'ont pas trouvé de suite sont conservées en mémoire pour être aussi comparées pendant un certain nombre de trames après leur dernière apparition dans l'image. C'est le cas pour *SORT* ou *DAN* qui maintiennent ces trajectoires *en vie* pour quelques trames ou encore pour *PermaTrack* [12] qui le fait (se basant sur *CenterTrack*) en priorité pour les objets centrés dans l'image : il part du principe qu'un objet qui disparaît au bord de l'écran en est probablement sorti, tandis qu'un objet qui disparaît au milieu de l'image le fait sans doute pour cause d'occlusion et donc risque d'apparaître de nouveau.

Traqueurs offline Les traqueurs *online*, s'ils ont l'avantage d'être utilisés en temps réel, se privent de quantités d'informations et d'outils sur lesquels ils pourraient s'appuyer pour améliorer grandement leurs résultats. Ainsi le fait de travailler sur l'ensemble des détections à la fois permet de pouvoir résoudre le problème comme un problème d'optimisation global. Toutefois, chercher à explorer l'ensemble de l'espace des solutions est une tâche impossible

dès lors qu'on a affaire à un grand nombre de détections et il convient de trouver des façons de simplifier le problème tout en conservant une partie des avantages trouvés en supprimant la contrainte du temps réel. Certaines méthodes s'éloignent d'ailleurs assez peu des méthodes online et certaines maintiennent la contrainte de *découverte chronologique* des détections. Par exemple, le traqueur *GOG* (pour "*Globally Optimal Greedy*") [38] consiste en un algorithme glouton qui résout le problème d'association en employant la programmation dynamique : il considère que des changements locaux à une solution optimale pour k trames permettent d'obtenir une solution optimale pour $k + 1$ trames. Il résout donc le problème d'association de proche en proche en cherchant à conserver des dynamiques cohérentes pour chacune de ses trajectoires. La caractéristique qui empêche cet algorithme d'être qualifié d'*online* réside dans le fait qu'il modifie les choix faits précédemment au fur et à mesure de la révélation des trames. Ainsi il peut travailler en temps réel mais les trajectoires données changeront jusqu'à la fin de son exploration des trames. D'autres conservent aussi ce travail chronologique mais le font dans les deux sens : du passé vers le futur **et** du futur vers le passé (ce qui est difficilement faisable en temps réel) ! Ainsi la méthode *TMOH* de [17] propose d'employer son traqueur (qui effectue des régressions comme *Tracktor*) dans les deux directions temporelles pour rendre ses résultats plus robustes.

De nombreuses autres approches ne se forcent pas à artificialiser cette *découverte chronologique des trames* et résolvent des problèmes d'affectation plus globaux mais découpent l'ensemble des détections en lots (*batches*, ce qui rend la tâche d'association plus tractable. C'est notamment le cas du travail proposé par Alexandre Pineault [35, 36], premier à appliquer la Programmation par Contraintes au problème du suivi multi-objet et qui, du fait de l'explosion en complexité du problème a proposé de le résoudre pour des lots restreints de trames avant de relier ses solutions entre elles. La méthode *H2T* [39] (pour "*Hierarchical relation Hypergraph based Tracker*" ou en français "*Traqueur basé sur un hypergraphe de relation hiérarchique*") travaille elle aussi du local vers le global en découpant l'ensemble des trames en sous-ensembles dans lesquels elle cherche à minimiser une somme pondérée de fonctions d'affinités (qui s'assurent de la proximité en apparence des détections dans une même trajectoire, de leur proximité spatiale ainsi que de la fluidité de la trajectoire), puis rassemble ces sous-ensembles pour résoudre de nouveau le même problème, encore et encore jusqu'à arriver à l'ensemble complet des trames.

Certaines méthodes s'attaquent toutefois directement à l'ensemble du problème d'association. C'est le cas de la méthode employée par la méthode *Discrete Continuous Tracking (DCT)* [31] et *Continuous Energy Minimization* [32] déjà évoquée dans la section 2.2.1 qui cherche à minimiser une somme de *fonctions d'énergie* non convexes par descente de gradient. Comme expliqué précédemment, ces fonctions prennent en compte de nombreux aspects des

trajectoires. La fonction globale de la méthode *DCT* se décompose comme la somme de sous-fonctions suivante :

$$E(X) = E_{det} + \alpha E_{dyn} + \beta E_{exc} + \gamma E_{per} + \delta E_{reg} \quad (2.5)$$

où :

- E_{det} correspond à l'énergie de détection, qui est minimale lorsque le score des détections des trajectoires est maximal
- E_{dyn} correspond à l'énergie de mouvement, qui ici correspond à un modèle de vitesse constante et diminue le plus les trajectoires collent à ce modèle : si on a des changements brusques de vitesse ou de direction, cette énergie augmente.
- E_{exc} correspond à l'énergie d'exclusion mutuelle, qui est minimale si aucune détection n'est partagée par plusieurs trajectoires
- E_{per} correspond à l'énergie de persistance, vise à minimiser le nombre de trous dans les trajectoires
- E_{reg} correspond au régulateur qui a pour objectif de minimiser le nombre de trajectoires et de maximiser leurs longueurs respectives

L'association est ainsi guidée par la minimisation de cette somme de fonctions par descente de gradient. L'algorithme *CEM* (pour Continuous Energy Minimization) ajoute à celles-ci E_{app} une énergie d'apparence, qui vise à optimiser la similarité visuelle des détections appartenant à des mêmes trajectoires.

Enfin une approche régulièrement employée dans l'état de l'art consiste à s'appuyer sur les forces des traqueurs online, particulièrement rapides, et de chercher à améliorer les résultats qu'ils fournissent. Dans le cas des traqueurs se basant sur *IOU-Tracker* [21, 37], l'objectif est souvent de combler les trous créés par les occlusions auxquelles *IOU-Tracker* est très sensible. Ainsi le traqueur *VIOU* [21] (pour "Visual Intersection over Union) se base sur les trajectoires construites par *IOU-Tracker* et applique un *SOT* (pour *Single Object Tracker* ou traqueur d'objet singulier par opposition aux traqueurs multi-objets) à la fin des trajectoires construites par *IOU-Tracker* pendant plusieurs trames pour voir s'il retrouve un objet et donc peut le relier à une trajectoire éloignée de plusieurs trames. Ce processus est aussi effectué du futur vers le passé pour s'appuyer sur d'éventuelles meilleures descriptions de la trajectoire subséquente.

Une autre approche récemment explorée par *SP_Con* [40] consiste à découper les trajectoires fournies par *IOU-Tracker* dans les zones de discontinuité de vitesse pour construire des tracklets et à les associer entre elles en fonction des distances entre des caractéristiques d'apparence fournies par des réseaux de neurones récurrents.

CHAPITRE 3 DÉMARCHE DU TRAVAIL DE RECHERCHE

3.1 Réflexion initiale

Au commencement de ce travail de recherche, nous avons décidé de poursuivre dans la lancée entamée par Alexandre Pineault [36] qui avait le premier tenté d’appliquer la Programmation par Contraintes au suivi multi-objet. Le choix de travailler sur une association de détections deux-à-deux a créé les principales limitations de ce modèle. En effet, cela a rendu l’espace de recherche extrêmement grand et donc le temps de calcul assez élevé. Cela a donc forcé la décision de travailler par lots de trames et de restreindre fortement les distances entre boîtes englobantes considérées, ce qui augmente la sensibilité du modèle aux occlusions.

Pour remédier à cette difficulté, notre idée pour ce projet a été de travailler non pas sur les détections mais sur des *tracklets* ou *mini-trajectoires* que l’on chercherait à associer, ce qui diminuerait grandement la taille du problème : on passe par exemple pour les exemplaires de MOT17 (avec le détecteur FRCNN) de l’ordre de 10^3 à 10^4 détections à associer à celui de 10 à 100 tracklets à associer par exemplaire. Ce choix permet de plus d’augmenter la finesse du modèle, qui peut maintenant s’appuyer sur les caractéristiques des tracklets, dynamiques donc, et non plus seulement celles des détections, statiques par essence.

Ici on parlera donc de *tracklet* pour toute association de détection fournie par un traqueur initial, qu’on l’ait coupée ou non et de *trajectoire* pour parler soit des associations de tracklets que nous construisons, soit des associations de détections fournies par la réalité-terrain comme représentant un unique objet.

3.1.1 Améliorer des traqueurs simples

Pour mener à bien cette idée, il est nécessaire de construire en premier lieu des tracklets et, comme on en a parlé dans la revue de littérature, il existe des traqueurs simples qui sont extrêmement performants en terme de temps d’exécution (notamment *IOU-Tracker* [2] qui parvient à traiter jusqu’à 100 000 trames par seconde, ce qui le rend des centaines de fois plus rapide que la majorité des autres traqueurs de l’état de l’art) et nous avons décidé de partir de ceux-ci pour capitaliser sur leur performance et essayer de corriger leurs défauts a posteriori, notamment ceux qui interviennent au niveau des occlusions. À terme le modèle est aussi applicable à toutes sortes de traqueurs mais les améliorations sont les plus notables dans le cas des traqueurs simples, qui travaillent souvent de façon locale et sur peu de caractéristiques.

3.1.2 Découpage en tracklets

Un découpage n'est pas toujours nécessaire La première approche que nous avons décidé de suivre est de ne pas effectuer nous-mêmes de découpage. En effet, pour *IOU-Tracker*, traqueur sur lequel nous avons basé la majorité de notre étude, les occlusions risquent de très rapidement occasionner une scission de la trajectoire, comme on peut le voir dans les figures 3.1 et 3.2. Dans la figure 3.1, le passage de l'individu derrière l'arbre fait que le détecteur ne le reconnaît plus à ($t=4$). Comme *IOU-Tracker* fonctionne au frame par frame, il sera obligé de clore la trajectoire à ($t=3$) et d'en commencer une nouvelle à ($t=5$). Dans la figure 3.2, l'individu est détecté à chaque frame. Toutefois, selon le seuil d'acceptation du traqueur, la détection partielle due à l'occlusion par le lampadaire à ($t=9$) (qui entraîne un IOU de 0.37) risque déjà d'occasionner une coupe à cette frame. Le phénomène de coupe est certain au passage à ($t=10$) où l'on n'a plus aucune superposition ($\text{IOU} = 0$) entre les boîtes englobantes qui encadrent l'individu.

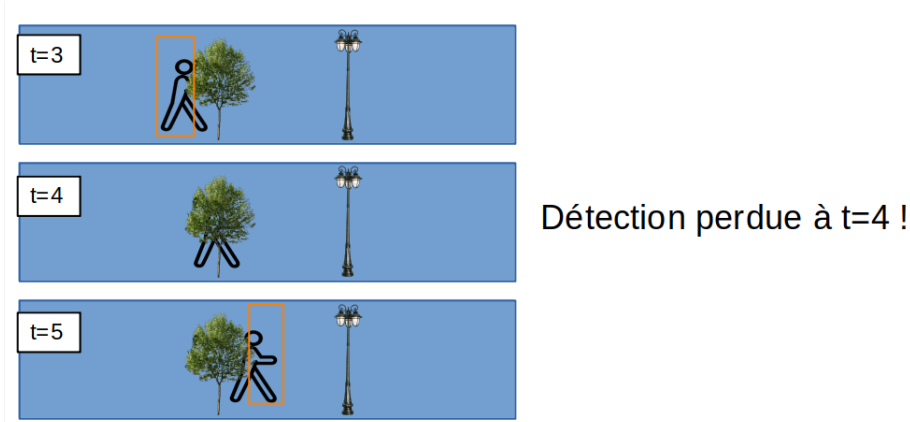


Figure 3.1 Exemple occasionnant une scission de trajectoire pour *IOU-Tracker* : détection manquante

En revanche, on suppose que les tracklets conservées par l'*IOU-Tracker* (dont le seuil d'acceptation est de 0.9 dans ce que proposent ses auteurs pour MOT17) ne présentent pas d'erreurs dans la mesure où même une occlusion très partielle conduit à passer sous le seuil d'acceptation proposé et scinde la trajectoire. On se retrouve avec des trajectoires souvent morcelées en multiples tracklets, trop courtes mais dont la cohérence interne est élevée, c'est-à-dire qu'on conjecture que les tracklets fournies par *IOU-Tracker* ont majoritairement les caractéristiques suivantes :

- Chaque tracklet contient **un et un seul** objet d'intérêt.
- Un objet d'intérêt singulier peut être contenu dans plusieurs tracklets à la fois, mais

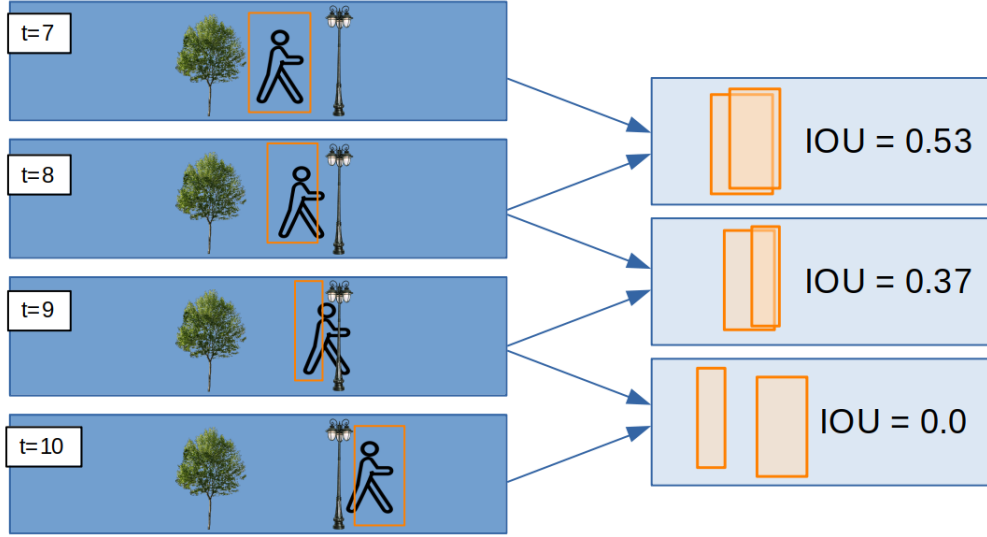


Figure 3.2 Exemple occasionnant une scission de trajectoire pour *IOU-Tracker* : détection partielle

seulement si celles-ci ne se superposent pas dans le temps.

Si cette conjecture se vérifie dans la totalité des cas, on comprend bien qu'il est inutile de découper les trajectoires fournies par *IOU-Tracker*. Comme ce n'est pas totalement le cas, nous avons décidé d'ajouter un module complémentaire qui est décrit ci-dessous.

Emploi d'un découpeur simpliste : *TrackletCutter* Pour les autres traqueurs et pour les cas d'occlusions moins manifestes, nous avons décidé de développer un découpeur de tracklets simple que nous avons baptisé *TrackletCutter* et qui fonctionne de la manière suivante : Dès que deux détections présentes dans une même trame ont un IOU supérieur à un seuil fixé, on coupe la tracklets à laquelle elles appartiennent à l'instant dans le temps où se situent ces détections. Ainsi on atténue le risque d'IDSW un peu plus qu'avant tout en conservant les zones dans lesquelles les risques d'erreur semblent moins élevés, notamment pour *IOU-Tracker* dont on peut assurer que les détections associées renvoyées remplissent les deux critères suivants après passage par le *TrackletCutter* :

$$\forall (d_{t_i, f_k}, d_{t_i, f_{k+1}}) \in Dets, \begin{cases} IOU(d_{t_i, f_k}, d_{t_i, f_{k+1}}) > T_{IOU} \\ \forall d_{t, f_k}, t \neq t_i, IOU(d_{t, f_k}, d_{t_i, f_k}) < T_{TC} \end{cases} \quad (3.1)$$

où :

- *Dets* représente l'ensemble des détections fournies par le détecteur.
- d_{t_i, f_k} représente la détection appartenant à la trajectoire t_i présente dans la trame f_k

- T_{IOU} représente le seuil d'acceptation de *IOU-Tracker* pour associer deux détections appartenant à des trames successives dans une même trajectoire.
- T_{TC} représente le seuil de coupe de *TrackletCutter*, IOU à partir duquel il coupe une trajectoire en deux.

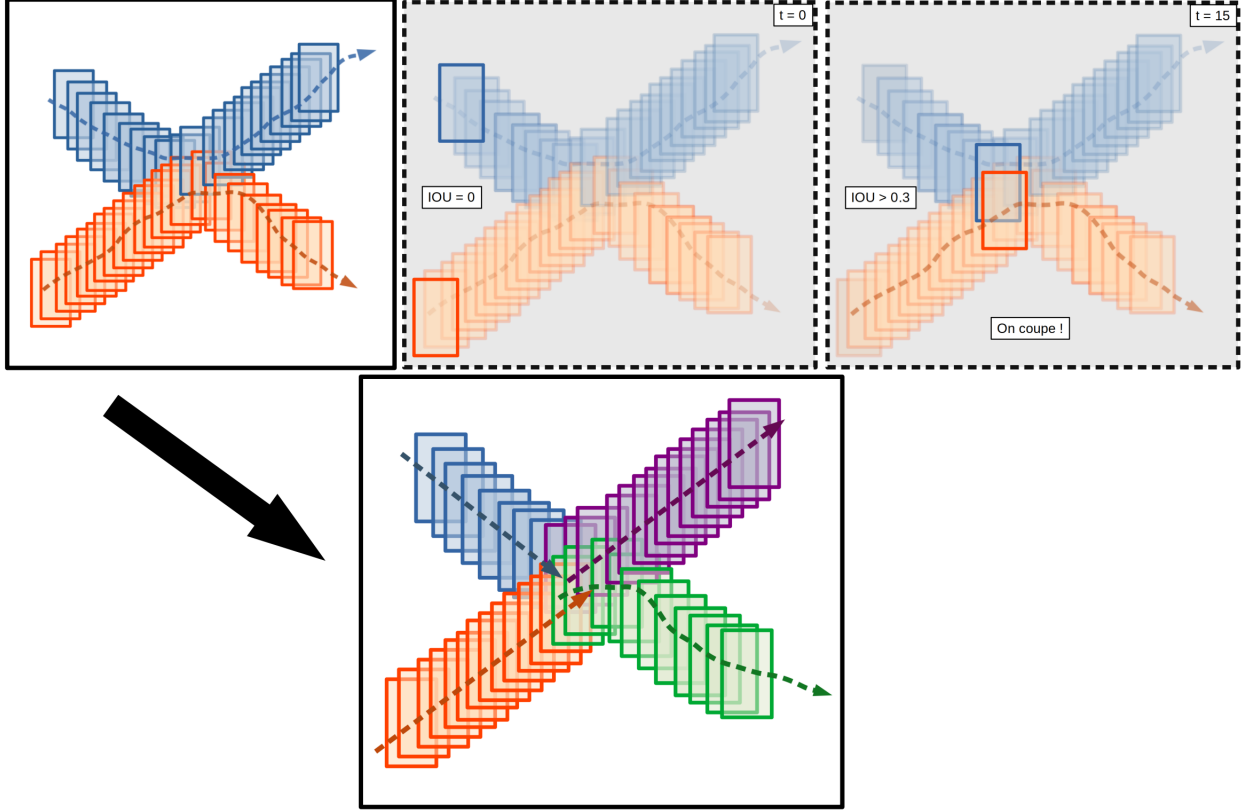


Figure 3.3 Exemple d'application du découpeur de tracklets *TrackletCutter* avec un seuil de coupe $T_{TC} = 0.3$

Ainsi dans le schéma de la figure 3.3, on a d'abord deux tracklets (une représentée en bleu et l'autre en orange). Le *TrackletCutter* va mesurer le niveau de superposition (l'IOU en fait) des boîtes englobantes de ces tracklets trame par trame. On a donc d'abord à ($t=0$) un IOU nul puis on avance dans le temps et à ($t=15$) on observe une superposition supérieure au seuil fixé pour la coupe (posé ici à 0.3). En sortie, on obtient donc quatre tracklets dont l'association restera à faire a posteriori : peut-être que la tracklet orange sera associée à la violette et non pas à la verte comme c'était le cas au départ. Cette méthode de découpage est en revanche assez longue dans la mesure où on va mesurer l'IOU de toutes les détections présentes dans chaque trame de chacune des vidéos. Cela ne nous pose toutefois pas spécialement de problème car nous ne cherchons pas à faire du temps réel avec ce module. Nous n'avons donc pas cherché

à optimiser le temps de calcul ici.

3.2 Modélisation des tracklets

3.2.1 Caractéristiques retenues

Une fois les tracklets obtenues (qu'on les récupère telles quelles au sortir du traqueur ou qu'elles proviennent du *TrackletCutter*), il convient de définir les informations que l'on en retient, car c'est à partir de celles-ci que notre modèle d'association travaillera. Nous avons décidé de retenir les caractéristiques suivantes :

- **trames** de début et de fin : celles-ci sont indispensables pour organiser temporellement les tracklets entre elles et éviter les chevauchements.
- **boîte englobante** en début et en fin de tracklet : on conserve les informations $(x_{haut-gauche}, y_{haut-gauche}, largeur, hauteur)$ fournies par le détecteur, auxquelles on se référera par (x, y, w, h) dans la suite.
- **vecteur vitesse** en début et en fin de tracklets : il s'agit d'une des informations gagnées en travaillant sur des tracklets et non plus sur des détections unitaires. Celle-ci est calculée sur les centres des premières (resp. dernières) boîtes englobantes de la trajectoire et selon la longueur de la trajectoire, on emploie deux méthodes différentes pour le faire que nous allons décrire ci-dessous.

3.2.2 Moyennage des vitesses et des boîtes englobantes

La présence d'occlusions progressives, l'imperfection des détecteurs et les changements de formes potentiels des piétons qui se déplacent rendent moins performante l'idée de se baser sur la première et la dernière détections pour caractériser nos tracklets.

En effet, comme on peut le voir sur le figure 3.4, la boîte englobante orange qui entoure le piéton suivi diminue de taille et sa forme change à mesure que celui-ci passe derrière le lampadaire qui le dissimule. Ainsi les dimensions de la dernière boîte englobante ne sont pas représentatives de celles que l'on peut attendre à la réapparition de l'individu.

De la même façon, la figure 3.5 décrit une assez grande variabilité dans la forme de la boîte englobante en fonction de la posture de l'individu : l'aire de cette boîte englobante est augmentée de 50% par la simple extension des membres de l'individu lors de sa course.

De même, si on décide de calculer la vitesse de déplacement de l'objet d'intérêt à partir de deux détections successives, un petit décalage dû au détecteur risque de nous faire conserver une vitesse très peu représentative de la direction de déplacement de l'objet. Ceci est

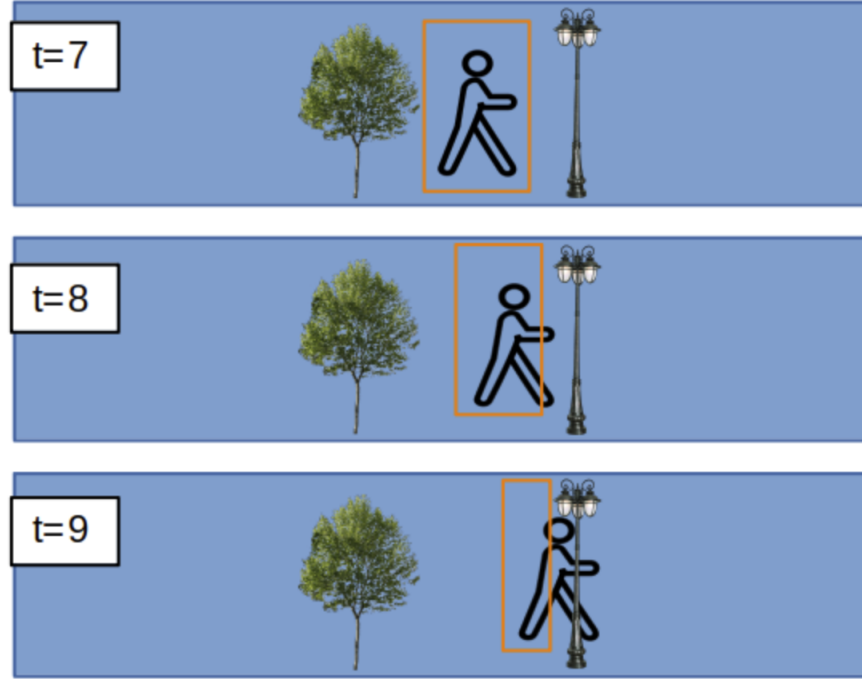


Figure 3.4 Exemple de changements de taille et de forme d’une boîte englobante dans une situation d’occlusion dus au passage de l’objet d’intérêt derrière un obstacle

particulièrement vrai dans la mesure où le nombre de trames par seconde est très élevé.

Pour rendre notre modèle un peu plus robuste, nous avons décidé de calculer les vitesses des trajectoires comme suit (détaillé pour la **vitesse de début** ici, sachant que le même procédé est effectué pour calculer la vitesse de fin) :

- Si la trajectoire T comporte **plus de 10 détections**, on calcule la vitesse sur plusieurs trames au début de la tracklet (pour moins subir les variations dues à l’apparition de l’objet dans le champ de vue). On calcule une boîte englobante moyenne sur les trois premières détections et une seconde sur les trois suivantes. On calcule alors la vitesse sur les centres de ces deux boîtes englobantes plus représentatives de la réalité. Ainsi, si on fixe $(d_0, \dots, d_6)$ les 7 premières détections de T et pour chacune d’entre elles (x_i, y_i, w_i, h_i) leur boîtes englobantes et f_i les trames dans lesquelles elles apparaissent. On a d’abord, pour chacune d’entre elles, leur centre C_i de coordonnées :

$$\begin{aligned} C_i(x) &= x_i + \frac{w_i}{2} \\ C_i(y) &= y_i + \frac{h_i}{2} \end{aligned} \tag{3.2}$$

Ensuite on calcule D_0 et D_1 , les deux super-boîtes englobantes, de centre C_{D_i} et trames

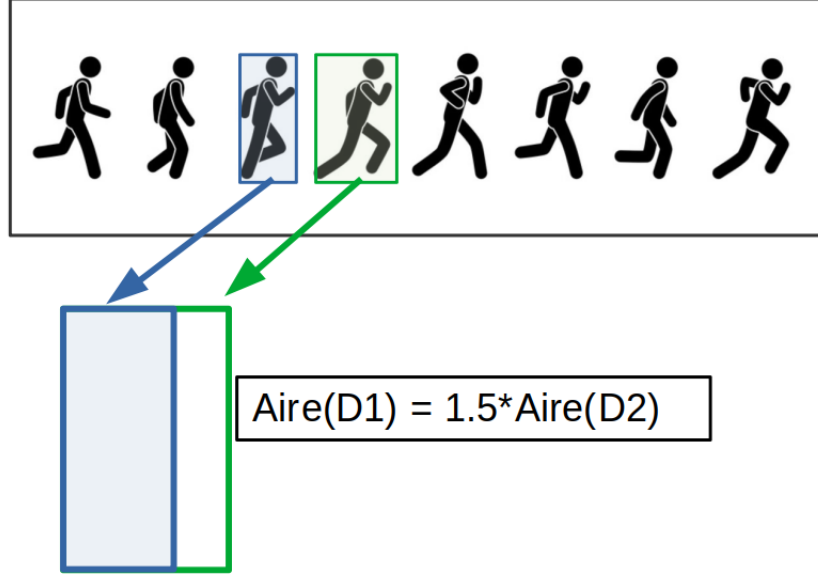


Figure 3.5 Exemple de changement de taille et de forme d'une boîte englobante dus aux changements de posture de l'individu suivi

d'apparition fictives F_{D_i} :

$$\begin{aligned}
 D_0 : & \begin{cases} C_{D_0}(x) = \frac{1}{3}[C_1(x) + C_2(x) + C_3(x)] \\ C_{D_0}(y) = \frac{1}{3}[C_1(y) + C_2(y) + C_3(y)] \\ F_{D_0} = \frac{1}{3}[f_1 + f_2 + f_3] \end{cases} \\
 D_1 : & \begin{cases} C_{D_1}(x) = \frac{1}{3}[C_4(x) + C_5(x) + C_6(x)] \\ C_{D_1}(y) = \frac{1}{3}[C_4(y) + C_5(y) + C_6(y)] \\ F_{D_1} = \frac{1}{3}[f_4 + f_5 + f_6] \end{cases}
 \end{aligned} \tag{3.3}$$

Et enfin on calcule notre vitesse moyenne :

$$\begin{cases} v_{debut_x} = \frac{1}{F_{D_1} - F_{D_0}}(C_{D_1}(x) - C_{D_0}(x)) \\ v_{debut_y} = \frac{1}{F_{D_1} - F_{D_0}}(C_{D_1}(y) - C_{D_0}(y)) \end{cases} \tag{3.4}$$

Ainsi on obtient notre vitesse moyenne, moins soumise aux légères variations du détecteur ou aux occlusions régulièrement présentes en début et fin de trajectoires.

- Si la trajectoire T comporte **moins de 10 détections**, pour calculer sa vitesse de début, on se contente de calculer le déplacement entre sa première et sa deuxième boîtes englobantes (d_0 et d_1 donc et d_{n-1} et d_n pour sa vitesse de fin) et on en fait le

quotient par le nombre de trames écoulées entre ces deux détections :

$$\begin{cases} v_{debut_x} = \frac{1}{f_1 - f_0}(C_1(x) - C_0(x)) \\ v_{debut_y} = \frac{1}{f_1 - f_0}(C_1(y) - C_0(y)) \end{cases} \quad (3.5)$$

3.3 Modèle d'association des tracklets

On arrive maintenant à la phase principale de notre modèle : celle de l'association des tracklets entre elles pour construire nos trajectoires.

3.3.1 Modélisation du problème d'association

Pour notre modèle qui va procéder à l'association des tracklets pour former des trajectoires complètes, il convient de définir :

- les variables du problème
- le domaine de ces variables : l'ensemble des valeurs qu'elles peuvent prendre
- les contraintes que l'on affecte à ces variables : les restrictions que l'on applique à chaque domaine
- la stratégie de branchement ou de résolution

Nous avons décidé de définir comme variables les **successeurs** des tracklets fournies par le traqueur initial (après un potentiel découpage par le *TrackletCutter*). Dire que la tracklet j est le successeur de la tracklet i revient à dire que les tracklets i et j appartiennent à la même trajectoire et que j suit directement i dans cette trajectoire. Ainsi, on obtient à la fin du processus d'association une liste telle que :

$$\forall t_i \in T, S(t_i) = \begin{cases} t_j \in T \setminus \{t_i\} & \text{si } t_j \text{ suit directement } t_i \text{ dans sa trajectoire} \\ \emptyset & \text{si } t_i \text{ est la dernière tracklet de sa trajectoire} \end{cases} \quad (3.6)$$

avec T l'ensemble des tracklets fournies par le traqueur initial et $S(t_i)$ le successeur de t_i dans sa trajectoire. En pratique, la modélisation que nous avons construite nécessite d'affecter à chaque variable une valeur différente. Il était donc nécessaire de définir pour chacune une valeur d'arrêt différente que nous avons fixée à $n + t_i$ pour chaque tracklet t_i où n est le nombre total de tracklets. $n + t_i$ remplace donc $\emptyset$ dans l'équation 3.6. On aurait pu prendre t_i comme valeur d'arrêt pour $S(t_i)$ mais cela aurait risqué de créer des cycles (où t_i est son propre successeur, ce qui n'a pas de sens).

Ainsi, pour un ensemble de tracklets $T = \{1, 2, 3\}$ on aura que :

- notre ensemble de variables est $X = \{S(1), S(2), S(3)\}$

- l'ensemble des valeurs est $D = \{1, 2, 3, 4, 5, 6\}$ avec 4, 5 et 6 les valeurs d'arrêt respectivement de $S(1)$, $S(2)$ et $S(3)$.

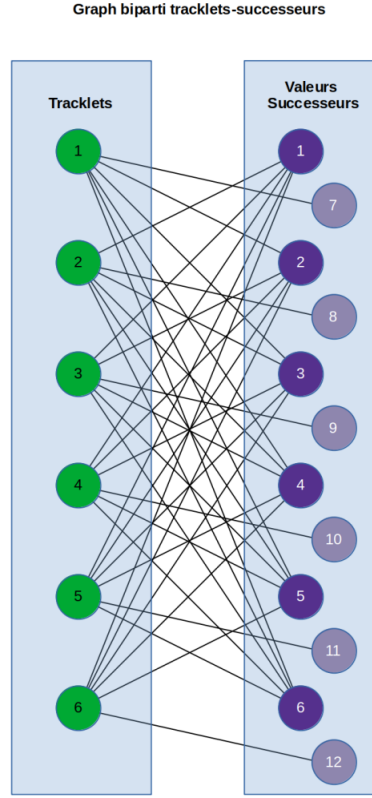


Figure 3.6 Exemple de modélisation du problème d'association par graphe biparti associant les tracklets à leur successeur

Dans l'exemple de la figure 3.6, on a 6 tracklets d'identifiants i allant de 1 à 6 et de valeurs d'arrêt $6 + i$. On observe bien que chaque tracklet peut donc avoir pour successeur chacune des autres tracklets ainsi que sa valeur d'arrêt.

La résolution du problème se fait alors par exploration d'un arbre binaire (construit lors de l'exploration) dont chaque noeud a deux branches ordonnées comme suit :

- une branche de gauche : $S(t_i) = t_j$
- une branche de droite : $S(t_i) \neq t_j$

Une solution est un chemin dans cet arbre, de la racine à une feuille, qui permet d'obtenir une unique affectation à chacun des $S(t_i)$. Toute solution valide doit affecter à chaque tracklet un successeur. Dans le cas simple de la figure 3.7 par exemple, on a deux tracklets d'identifiants 1 et 2 donc deux variables $S(1)$ et $S(2)$ qui représentent leurs successeurs, de domaines respectifs $S1 = \{2, 3\}$ et $S2 = \{1, 4\}$ (3 et 4 étant respectivement les valeurs d'arrêt des

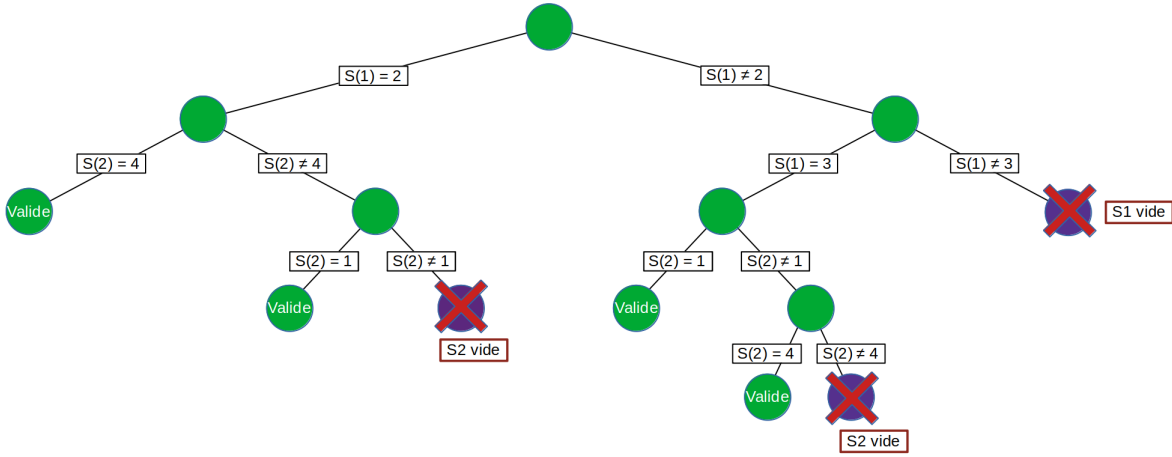


Figure 3.7 Exemple d'arbre des solutions dans un cas à deux tracklets

tracklets 1 et 2). On observe donc que :

- chaque noeud a deux branches $S(i) = j$ et $S(i) \neq j$
- les solutions valides (les feuilles vertes) affectent une valeur aux deux variables $S(1)$ et $S(2)$.
- toute solution qui réduit le domaine d'une variable à l'ensemble vide est invalide. Il s'agit dans ce schéma des feuilles barrées d'une croix rouge. Par exemple, pour celle la plus à droite, on a $S(1) \neq 2$ et $S(1) \neq 3$, donc il n'y a plus de successeur disponible pour 1.

Filtrage par contraintes Pour ne pas avoir à construire un arbre trop grand et effectuer des explorations inutiles, il est utile de l'élaguer à l'avance (avant sa construction donc) en réduisant les domaines des variables par le biais de ce qu'on appelle des *contraintes*, relations que l'on fixe sur les variables et qui restreignent les valeurs qu'elles peuvent prendre. Une contrainte peut porter sur une variable indépendamment des autres (on dira alors qu'elle est d'*arité 1*) ou sur plusieurs variables à la fois (et leur arité correspondra au nombre de variables auxquelles elles s'appliquent à la fois dans une même relation). Dans notre cas, si l'on reprend l'exemple 3.3.1 cité plus haut (avec trois tracklets $\{1, 2, 3\}$), on aura a minima trois contraintes pour obtenir une solution valide :

1. une contrainte d'arité 1 sur les valeurs d'arrêt :

$$\forall i \in \{1, 2, 3\}, S(i) \in \{1, 2, 3, i + 3\} \quad (3.7)$$

2. une autre contrainte d'arité 1 sur le fait qu'une tracklet ne peut pas être son propre successeur ni le successeur d'un de ses descendants (successeur de successeur de successeur et ainsi de suite) pour ne pas créer de trajectoires à boucles temporelles :

$$\forall i \in \{1, 2, 3\}, \forall k \in \mathbb{N}, S^k(i) \neq i \quad (3.8)$$

Dans notre cas, ce problème est réglé par une contrainte qui gère la cohérence temporelle : j ne peut pas succéder à i si i se termine après le début de j .

3. une contrainte d'arité 3 (en général, le nombre de variables successeur) nommée *allDifferent*, sur le fait que les valeurs prises pour chaque successeur doivent être différentes les unes des autres :

$$\forall (i, j) \in \{(k, l) \in \{1, 2, 3\}^2 | k \neq l\}, S(i) \neq S(j) \quad (3.9)$$

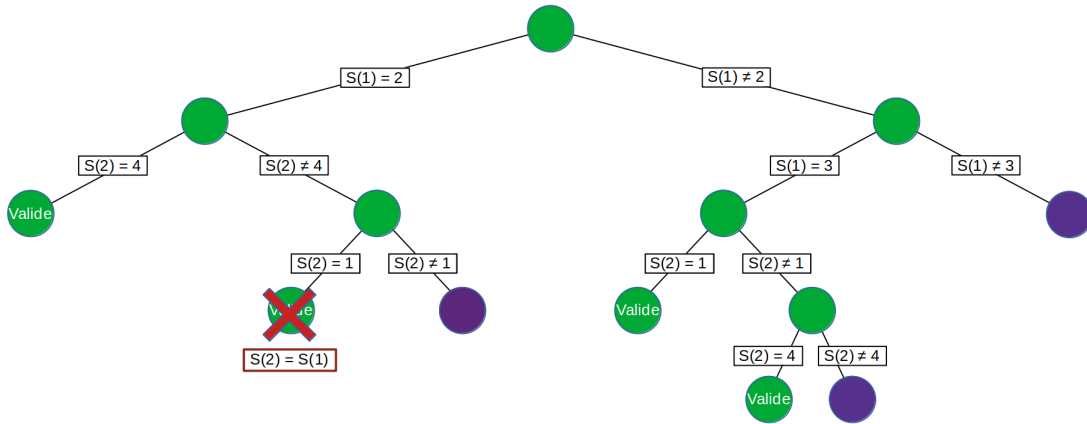


Figure 3.8 Exemple d'arbre des solutions dans un cas à deux tracklets

Si l'on affecte au cas de la figure 3.7 la contrainte qui interdit à une tracklet d'être le successeur d'un de ses descendants, cette contrainte peut être reformulée en : $(S(1), S(2)) \neq (2, 1)$ ce que l'on peut voir dans la figure 3.8, qui supprime le chemin $(S(1) = 2, S(2) \neq 4, S(2) = 1)$ qui ici était une solution. Dans un cas à plus de variables ou de valeurs cette contrainte aurait permis de couper une ou plusieurs branches. Les contraintes choisies pour ce modèle sont détaillées un peu plus loin dans la section 3.3.2.

Heuristiques de branchement : choix de variables et de valeurs Comme l'objectif de notre recherche est de ne pas avoir à énumérer l'ensemble des choix pour les évaluer, il convient de définir :

- une façon d'organiser l'arbre des solutions, tant au niveau de l'ordre des variables dont on va explorer les valeurs que de l'ordre des valeurs que l'on va leur proposer. En effet, selon son organisation, les noeuds que les contraintes vont permettre de couper peuvent se situer plus ou moins haut dans l'arbre et donc réduire plus ou moins la dimension de cet arbre. On parle d'**heuristique de choix de variables** et d'**heuristique de choix de valeurs**.
- une façon de parcourir ces solutions : On peut parcourir l'arbre des solutions de façon aléatoire jusqu'à trouver une solution valide ou organiser cette recherche. S'arrête-t-on dans notre exploration à la première solution valide ou à la dixième, à la centième solution améliorante ?

On parle donc d'heuristiques de choix de variables et de choix de valeurs pour décrire cet ordre d'apparition des variables et des valeurs qu'elles prennent dans l'arbre des solutions. Dans l'exemple précédent de la figure 3.7, ces choix sont faits de façon aléatoire mais d'autres types existent. Pour le choix des variables, on peut par exemple chercher à explorer d'abord les variables aux plus petits domaines (nombre de valeurs qu'elles peuvent prendre), ou faire ce choix en rapport à leur degré (le nombre de contraintes qui les impliquent), ou encore l'effet sur la taille de l'espace de recherche qu'a le fait d'affecter une valeur à cette variable. Dans notre cas, ce choix est particulier et nécessite que l'on explore la notion de propagation de croyances avant de l'expliquer, ce que nous faisons dans la partie 3.3.1 au paragraphe "Notre utilisation de MiniCPBP".

Stratégies d'exploration Une fois l'arbre construit, on a plusieurs choix de parcours de cet arbre ainsi que de critères d'arrêt de la recherche. Il est possible de s'arrêter à la première solution valide et c'est notamment ce qui est fait dans le cadre de problèmes comme le Sudoku : dès qu'on obtient une grille qui remplit les différentes conditions du jeu (avoir des valeurs différentes comprises entre 1 et 9 sur chaque ligne, colonne et dans chaque carré), on renvoie la solution. Dans notre cas, nous avons décidé de faire ce choix après expérimentation, pour voir si l'on obtenait des solutions améliorantes en allant jusqu'à la dixième, voire à la centième solution valide. On a donc en premier lieu laissé ce paramètre ouvert.

Il existe quoiqu'il en soit différentes méthodes d'exploration de l'arbre des solutions. Les deux que nous avons implémentées sont les suivantes :

- *Depth First Search* ou *Parcours en profondeur* (DFS) : cet algorithme consiste à explorer l'arbre des solutions en prenant les déviations le plus bas possible dans l'arbre. Il va explorer les solutions comme suit :
 - D'abord on descend dans l'arbre en ne prenant que les branches de gauche (aucune déviation).

- Si on rencontre un échec, on remonte dans l'arbre jusqu'à la première déviation disponible, on la prend (branche de droite) et on recommence à descendre sur des branches de gauche.

Cette recherche a pour avantage de suivre au plus près les choix faits en terme d'heuristiques de choix de valeurs et de choix de variables. Toutefois elle donne une force particulière aux branchements effectués tôt : si un choix erroné (c'est-à-dire qu'il ne permet pas d'obtenir de solution valide ou améliorante) est effectué en haut de l'arbre, il sera très long de le corriger dans la mesure où la totalité du sous-arbre sera explorée.

- *Limited Discrepancy Search* ou *Parcours à déviation limitée* (LDS) : cet algorithme consiste à explorer l'arbre des solutions en augmentant progressivement le nombre de déviations. Ainsi il va explorer comme suit :
 - 0 déviation : on descend dans l'arbre des solutions en prenant systématiquement la branche de gauche.
 - 1 déviation : Si on rencontre un échec, on explore les solutions qui ne présentent qu'une déviation. D'abord on explore en prenant la première branche de droite en partant de la racine puis uniquement des branches de gauche. Ensuite on prend une première branche de gauche, la suivante à droite, puis uniquement les branches de gauche.
 - Et ainsi de suite.

Ce type d'exploration est donc moins sensible aux erreurs effectuées tôt dans l'exploration (en haut de l'arbre).

Notre utilisation de MiniCPBP Notre module d'association est basé sur un logiciel qui permet la construction d'algorithmes de programmation par contraintes avec propagation de croyances (ou *Belief Propagation*) : MiniCPBP, décrit dans [1]. La propagation de croyances (ou *BP*) se produit à l'origine comme suit :

1. A la première itération de *BP*, la marginale donnée par une contrainte c à l'affectation d'une valeur v à une variable x correspond à la fréquence d'affectation de v à x dans l'ensemble des solutions valides pour c . Ainsi si pour c , l'association (x, v) permet d'obtenir la moitié des solutions, la marginale renvoyée par c à x sera de 0.5. Ce calcul est effectué pour l'ensemble des contraintes, ce qui nous permet d'obtenir une marginale pour chaque contrainte pour cette paire variable-valeur. La marginale résultante pour la paire (x, v) à la fin de cette première itération est alors le produit des marginales fournies par chacune des contraintes, normalisé de sorte à ce que la somme des marginales des associations d'une variable avec chaque valeur de son domaine soit

- de 1.
2. La seconde itération va consister à réitérer le même processus mais en croisant cette fois-ci les fréquences d'affectations pour chaque variable avec les marginales calculées à l'itération précédente. Ainsi si pour c , l'association (x, v) permet d'obtenir la moitié des solutions mais que les affectations permettant ces solutions avaient une marginale de 0.2 à l'itération précédente, la nouvelle marginale fournie par c sera de $0.5 \times 0.2 = 0.1$.
 3. Et ainsi de suite, ce processus se répète à mesure des itérations et permet de converger progressivement vers la répartition réelle des solutions.

Dans notre cas, nous avons décidé d'employer la *BP* pour deux objectifs précis. Le premier est de combiner les scores des caractéristiques considérées, exprimées sous forme de contraintes. Par exemple, comme on le décrit plus en détail dans la partie 3.3.2, on a une contrainte *TimeDistance* qui va associer un score à l'affectation $S(i) = j$ d'autant plus élevé que l'écart temporel entre la fin de i et le début de j est faible. Ainsi dans notre modèle, la marginale d'une association pour une contrainte ne représente plus la fréquence de l'affectation dans les solutions de la contrainte, mais bien son score. La marginale résultante pour une association est égale au produit des scores normalisés de sorte à ce que la somme des marginales de chaque variable soit égale à 1 sur l'ensemble de son domaine à une même étape de recherche.

Notre heuristique de choix de valeurs et de variables L'autre intérêt que nous avons à employer la *BP* est qu'elle nous permet de construire une heuristique de choix de valeurs et de variables particulièrement efficace : *maxMarginal*. L'idée de cette heuristique est d'effectuer l'exploration des solutions (et donc la construction de l'arbre des solutions) en allant des marginales les plus élevées aux moins élevées. Ainsi une fois que les scores de chaque association tracklet-successeur sont calculés, on va effectuer les associations dont les marginales sont les plus élevées les unes après les autres. L'emploi des stratégies d'exploration *DFS* et *LDS* permettra de rester au plus près de notre modèle si l'on n'arrive pas directement à une solution.

3.3.2 Description des contraintes

Notre modèle de contraintes s'inspire en quelque sorte de l'idée de minimisation d'une fonction d'énergie proposée par [31, 32] telle que décrite dans la partie 2.2.1. On peut en effet faire un parallèle direct entre plusieurs des fonctions d'énergies sommées dans le modèle de minimisation d'énergie et nos différentes contraintes qui affectent des scores aux associations de tracklets dont on effectue la somme. Obtenir un score proche de 1 dans notre modèle pour un maximum d'associations retenues revient à minimiser la somme des fonctions d'énergie

de leur modèle.

Contrainte *AllDifferent*

Comme décrit précédemment, l'objectif de la contrainte *AllDifferent* est de maintenir la situation suivante :

$$\forall (S(t_1), S(t_2)) \in X^2 | t_1 \neq t_2, S(t_1) \neq S(t_2) \quad (3.10)$$

avec t_1 et t_2 dans des tracklets et $S(t_1)$ et $S(t_2)$ leur successeurs respectifs. Dans notre cas, cela permet d'empêcher que deux tracklets aient le même successeur et donc à terme qu'aucune détection n'appartienne à deux trajectoires différentes. Il s'agit de la mission qui est tenue par la fonction d'énergie d'exclusion E_{exc} dans [32] qui pénalise l'affectation de la même valeur à plusieurs variables.

Calcul du score

Pour chacune des contraintes spécifiques aux caractéristiques des tracklets (*TimeDistance*, *PredictedIOU*, *PredictedCenterDistance*, *AngleDifference*, *SpeedNormDifference*, décrites plus bas), on associe donc à chaque paire tracklet-successeur un score qui permettra de quantifier la qualité des associations de tracklets selon la métrique de la contrainte.

Pour chaque contrainte C , le calcul de ce score se fait selon trois paramètres :

- T_{50}^C : valeur de la métrique d'association pour laquelle on fixe le score à 50% .
- T_{fin}^C : valeur de la métrique d'association de la tracklet fictive qui représente l'arrêt de la trajectoire. Il est en effet primordial de comparer l'ensemble des successeurs possibles à la possibilité de ne s'associer à aucun d'entre eux. Ainsi si la caractéristique évaluée pour l'association d'un successeur à une tracklet est inférieure à T_{fin}^C , la contrainte C favorise l'arrêt de la trajectoire. Dans un cadre à trois contraintes C_1 , C_2 et C_3 , il faudra que le produit des scores de l'association de tous les successeurs disponibles pour une tracklet T sur les trois contraintes soit inférieur au produit des scores obtenus pour $T_{fin}^{C_1}$, $T_{fin}^{C_2}$ et $T_{fin}^{C_3}$ pour que la trajectoire s'arrête.
- T_0^C : valeur de la métrique d'association à partir de laquelle on retire le successeur étudié de la liste des candidats envisagés. On pourrait envisager par exemple qu'un angle supérieur à 90° entre la fin de la trajectoire étudiée et le début du successeur envisagé rend leur association impossible. Ce seuil peut être activé ou non, mais s'il l'est et qu'il est atteint pour une tracklet et un successeur donnés, le successeur en question est instantanément retiré de la liste des successeurs envisagés pour la tracklet, quel que soit le score obtenu via d'autres contraintes.

Ces trois paramètres sont calculés différemment pour chaque contrainte en fonction de divers attributs de la vidéo à laquelle est appliquée le modèle pour permettre une certaine adaptabilité des contraintes comme nous le détaillerons plus bas.

De plus, par construction de *MiniCPBP*, un score de 1 signifie que le successeur est affecté à la variable (et donc associé à la tracklet) et un score de 0 signifie que le successeur est retiré du domaine des successeurs envisagés. Ainsi, tant qu'aucune de ces deux actions n'est voulue, il est nécessaire de maintenir les scores dans un intervalle $[min_{score}, max_{score}]$ que nous avons fixés respectivement à :

$$\begin{aligned} min_{score} &= 10^{-6} \\ max_{score} &= 1 - min_{score} \end{aligned} \tag{3.11}$$

Cela nous permet d'obtenir pour chacune des contraintes un score de la forme dépeinte dans la figure 3.9. Dans cette figure, on a donc le score en rouge dont on peut voir qu'il suit une courbe gaussienne telle qu'un score de 0.5 est atteint quand la caractéristique C étudiée atteint la valeur T_{50} . On observe que le score s'écarte de la gaussienne g (en noir) en trois occasions :

- Si $g(C) \geq max_{score}$, alors $score(C) = max_{score}$
- Si $g(C) \leq min_{score}$, alors $score(C) = min_{score}$
- Si $C > T_0$, alors $score(C) = 0$

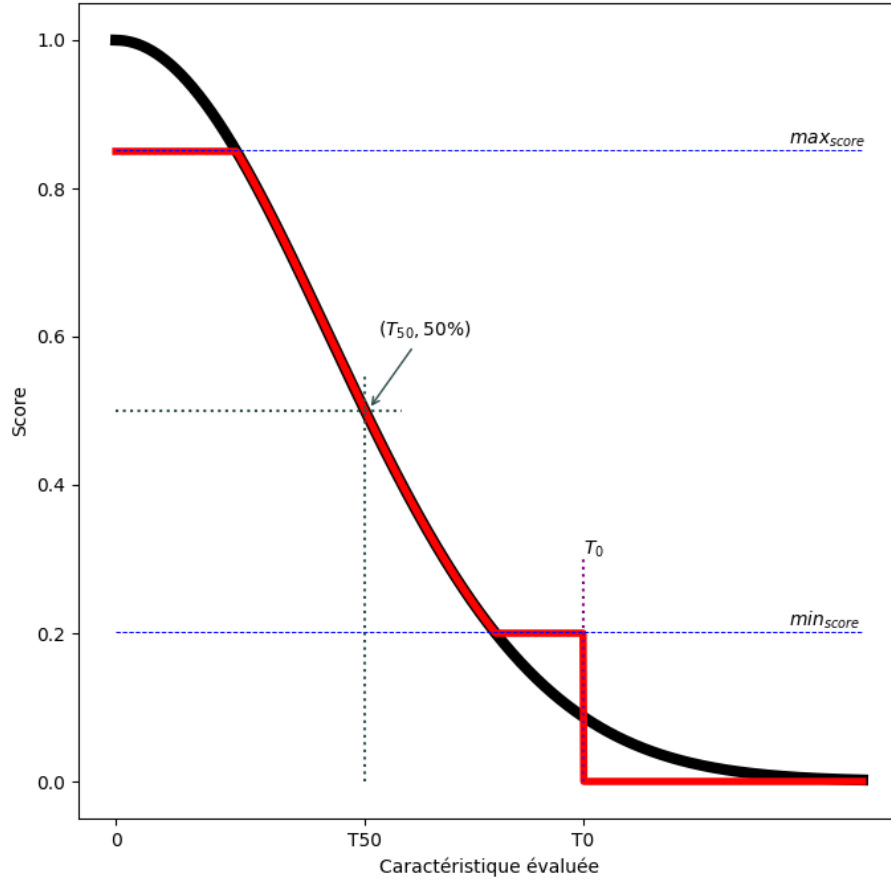


Figure 3.9 Visualisation du format du score appliqué à chacune des contraintes du modèle avec $max_{score} = 0.85$ et $min_{score} = 0.2$ pour faciliter la visualisation

Contraintes originales portant sur les trajectoires

Pour chaque contrainte, une caractéristique est choisie pour évaluer la qualité de l'association d'une tracklet à une trajectoire. Les caractéristiques que nous avons décidé d'évaluer sont divisibles en quatre catégories :

- **Contrainte sur l'espacement dans le temps**
 - **Hypothèse** : Plus deux tracklets sont éloignées l'une de l'autre dans le temps, plus il est probable qu'elles ne représentent pas le même objet ou que si elles représentent un unique objet, elles ne se suivent pas directement dans le temps. Leur association risque donc soit d'entraîner des *IDS* (s'il ne s'agit pas du même

objet) ou de créer des trous dans les trajectoires car notre modèle ne permet pas l'insertion de tracklet à l'intérieur de trajectoires. Il convient donc de diminuer leur score d'association à mesure que leur écart temporel augmente. En effet, dans le cas de voitures par exemple, on peut facilement imaginer deux véhicules d'aspects et de déplacements similaires qui traverseraient la vidéo à deux moments séparés de plusieurs dizaines de secondes. De plus dans le cadre de circulation sur une autoroute, il y a de grandes chances que ces véhicules circulent à la même vitesse, dans la même direction et passent par les mêmes endroits à des instants différents. En diminuant le score d'association de ces deux objets grâce à leur éloignement dans le temps, on devrait pouvoir diminuer ce genre d'associations.

— **Implémentation proposée :**

- **TimeDistance (TD) :** Cette contrainte porte sur l'écart temporel (en nombre de trames) séparant la dernière trame de la trajectoire étudiée de la première trame de son successeur potentiel. Plus cet écart est faible, plus le choix du successeur est probable. Les paramètres d'entrée pour cette contrainte sont les suivants :

- le facteur td_{50} permet de calculer le seuil T_{50}^{TD} d'écart temporel à partir duquel la contrainte affectera à l'association un score inférieur à 50%. On aura :

$$\mathbf{T}_{50}^{TD} = \mathbf{td}_{50} * \mathbf{FPS} \quad (3.12)$$

Ce seuil est indexé sur FPS , le taux de trames par seconde d'une vidéo pour qu'une variation de cette vitesse n'impacte pas la performance du modèle. En effet ce qui nous intéresse tient plus de l'écart temporel réel (en secondes donc) qu'à celui en terme de nombre de trames. Ainsi pour une vidéo présentant un FPS de 15, si on fixe, $td_{50} = 3$, la contrainte *TimeDistance* octroiera un score de 50% aux successeurs qui débutent 45 trames après la trajectoire étudiée.

- le facteur td_{fin} permet de calculer le seuil T_{fin}^{TD} à partir de T_{50}^{TD} . On aura :

$$\mathbf{T}_{fin}^{TD} = \mathbf{td}_{fin} * \mathbf{T}_{50}^{TD} \quad (3.13)$$

Ainsi si l'on reprend l'exemple précédent de $FPS = 15$ et $td_{50} = 3$ et que l'on fixe $td_{fin} = 2$, la contrainte *TimeDistance* favorisera l'arrêt de la trajectoire à une association si le successeur proposé est distant de plus de $2 * 45 = 90$ trames.

- le facteur td_0 permet de calculer le seuil T_0^{TD} de suppression de successeur

du domaine à partir de T_{50}^{TD} . On obtiendra alors :

$$\mathbf{T}_0^{\text{TD}} = \mathbf{td}_0 * \mathbf{T}_{50}^{\text{TD}} \quad (3.14)$$

En conservant l'exemple précédent, si l'on fixe $td_0 = 4$, tout successeur distant de plus de $4 * 45 = 180$ trames de la fin d'une tracklet sera retiré du domaine de celle-ci si ce seuil est activé pour *TimeDistance*.

— **Contraintes sur la dynamique d'une trajectoire**

- **Hypothèse** : Nos objets d'intérêt, qu'il s'agisse de véhicules ou de piétons, ne peuvent pas changer radicalement de direction ou de vitesse de déplacement en l'espace de quelques dixièmes de seconde. Ainsi, tant que l'écart temporel entre deux tracklets n'est pas trop grand, contraindre les variations de vitesse (tant en norme qu'en direction) devrait améliorer l'association. Toutefois, on suppose qu'en l'absence de contrainte sur l'espacement dans le temps ou dans l'espace entre les tracklets, ces contraintes ne seront pas efficaces : aller dans la même direction et à la même vitesse à des localisations ou des moments complètement différents n'est pas un comportement réaliste pour un même objet. Cela résulte en deux contraintes :

— **Implémentations proposées :**

- **AngleDifference (AD)** : Cette contrainte porte sur le changement de direction opéré pour associer une trajectoire à un successeur potentiel. On mesure l'angle entre l'orientation de la vitesse à la fin de la trajectoire et celle au début de la tracklet proposée comme successeur. Plus cet angle est petit, plus le choix du successeur est probable. Les paramètres d'entrée pour cette contrainte sont les suivants :

- le facteur ad_{50} permet de calculer le seuil T_{50}^{AD} d'écart angulaire à partir duquel la contrainte affectera à l'association un score inférieur à 50%. On aura :

$$\mathbf{T}_{50}^{\text{AD}} = \mathbf{ad}_{50} * \pi \quad (3.15)$$

Ce facteur permet donc de déterminer la proportion de π radians à partir de laquelle le score de l'association passe sous 50%.

- la puissance ad_{fin} permet de calculer le seuil T_{fin}^{AD} à partir de ad_{50} . On aura :

$$\mathbf{T}_{fin}^{\text{AD}} = \pi(1 - (1 - \mathbf{ad}_{50})^{\mathbf{ad}_{fin}}) \quad (3.16)$$

Ainsi on peut obtenir en augmentant ad_{fin} un T_{fin}^{AD} qui tend progressivement vers π . Typiquement pour $ad_{50} = 0.25$, soit un score de 50% pour angle à

45° entre deux trajectoires on obtiendra :

$$ad_{fin} = 2 \Rightarrow T_{fin}^{AD} = \pi(1 - (1 - 0.25)^2) = 0.44\pi$$

$$ad_{fin} = 3 \Rightarrow T_{fin}^{AD} = 0.58\pi$$

$$ad_{fin} = 4 \Rightarrow T_{fin}^{AD} = 0.68\pi$$

— idem la puissance ad_0 permet de calculer le seuil T_0^{AD} à partir de ad_{50} . On aura :

$$\mathbf{T}_0^{AD} = \pi(1 - (1 - \mathbf{ad}_{50})^{\mathbf{ad}_0}) \quad (3.17)$$

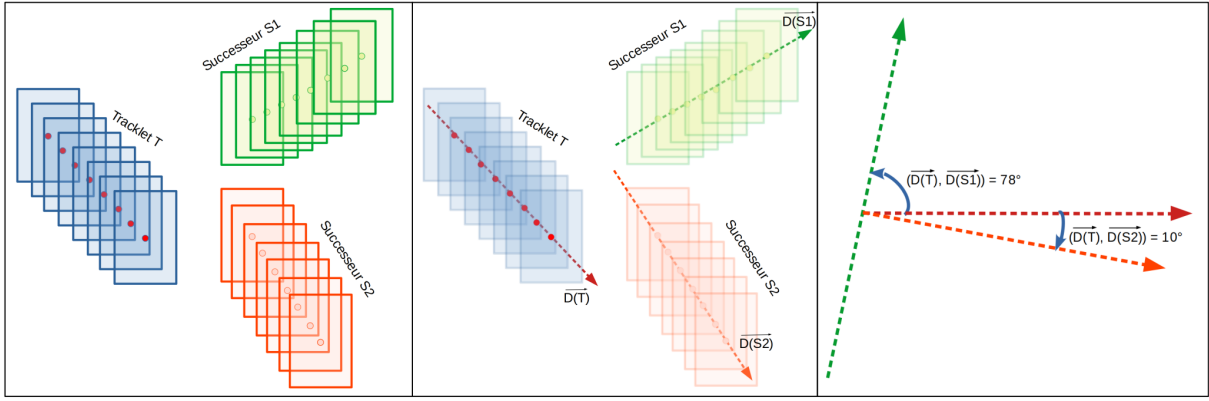


Figure 3.10 Exemple d'application de la contrainte AngleDifference : Visualisation de l'écart angulaire entre les directions d'une tracklet et de deux candidats à sa succession

Ainsi dans le cas décrit dans la figure 3.10, on a une tracklet T de direction $\vec{D}(T)$ et deux candidats successeurs S_1 et S_2 de directions respectives $\vec{D}(S_1)$ et $\vec{D}(S_2)$. Comme on peut le voir, l'angle entre les directions de T et S_1 est beaucoup plus grand que celui entre celles de T et S_2 . La contrainte affectera donc un score plus élevé à la paire (T, S_2) qu'à (T, S_1) . Et par exemple si le T_0^{AD} d'AngleDifference est de $\pi/3 = 60^\circ$, S_1 sera retiré du domaine des successeurs de T .

- **SpeedNormDifference (SD)** : Cette contrainte porte sur la variation de norme de vitesse qu'il faudrait accepter pour associer une trajectoire à un successeur potentiel. On mesure la différence entre les normes de vitesse à la fin de la trajectoire et celle au début de la tracklet proposée comme successeur. Plus cette différence entre les normes est petite, plus le choix du successeur est probable. Les paramètres d'entrée pour cette contrainte sont les suivants :
 - le facteur sd_{50} permet de calculer le seuil T_{50}^{SD} d'écart entre les normes

de vitesse à partir duquel la contrainte affectera à l'association un score inférieur à 50%. On aura :

$$\mathbf{T}_{50}^{\text{SD}} = \text{sd}_{50} \times \text{Diag}(\text{video})/\text{FPS} \quad (3.18)$$

Où $\text{Diag}(\text{video})$ représente la longueur de la diagonale de l'image de la vidéo étudiée (en pixels). Le choix d'indexer la valeur de T_{50}^{SD} sur deux caractéristiques de la vidéo permet de se rapprocher un peu plus d'une vitesse réelle. En effet un déplacement de dix pixels n'a pas le même sens dans une vidéo de 1500 pixels de côté que dans une de 15000 pixels de côté représentant la même scène. De même ce déplacement effectué en cinq trames représente des réalités tout à fait différentes si la vidéo a un taux de trames de 5 par seconde ou de 30 par seconde.

— le facteur sd_{fin} permet de calculer le seuil T_{fin}^{SD} à partir de T_{50}^{SD} . On aura :

$$\mathbf{T}_{fin}^{\text{SD}} = \text{sd}_{fin} * \mathbf{T}_{50}^{\text{SD}} \quad (3.19)$$

— le facteur sd_0 permet de calculer le seuil T_0^{SD} de suppression de successeur du domaine à partir de T_{50}^{SD} .

$$\mathbf{T}_0^{\text{SD}} = \text{sd}_0 * \mathbf{T}_{50}^{\text{SD}} \quad (3.20)$$

— **Contraintes sur la position des boîtes englobantes**

— **Hypothèses :**

1. Les objets d'intérêt que nous suivons ne sont a priori pas sujets à la téléportation. Ainsi, d'une trame à l'autre les détections appartenant à une même trajectoire devraient être très proches l'une de l'autre.
2. La vitesse de déplacement de nos détections varie peu en norme et en direction sur des intervalles de temps courts. Leur forme et leur taille varient elles aussi peu sur ces intervalles de temps. Si cela semble évident dans le cadre d'un véhicule, on peut considérer cela vrai aussi dans le suivi de piétons dans la mesure où l'intervalle de temps entre deux trames est inférieur au dixième de seconde.
3. Les tracklets que l'on cherche à associer sont espacées par des intervalles de temps restreints.
4. Si on accepte les hypothèses 2. et 3., alors 1. s'applique aussi à la projection (via sa vitesse) de la boîte englobante de la détection finale de la trajectoire sur

la trame à laquelle débute la tracklet dont on évalue le potentiel de succession.

— **Implémentations proposées :**

- **PredictedIOU (PIOU)** : Cette contrainte s’inspire du travail proposé par [4] pour le traqueur *Tracktor*. Cette contrainte consiste à projeter la boîte englobante de la dernière détection de la trajectoire étudiée T sur la trame t_{S_0} qui contient la première détection du successeur proposé S_0 . Cette boîte englobante projetée représente celle que l’on s’attendrait à retrouver pour l’objet suivi par notre trajectoire à cette trame t_{S_0} si l’objet avait poursuivi son déplacement avec une accélération nulle. On mesure ensuite l’IoU entre cette boîte englobante projetée et celle de la première détection du successeur. La probabilité du choix du successeur augmente avec cet IoU. Le calcul du score est fait par rapport à $1 - IOU$ pour avoir un score qui augmente avec l’IOU. Les paramètres d’entrée sont alors les suivants :

- le seuil $piou_{50}$ est l’IOU minimal en-dessous duquel le score passe sous les 50%. Il permet de calculer T_{50}^{PIOU} d’écart entre les normes de vitesse à partir duquel la contrainte affectera à l’association un score inférieur à 50%. On aura :

$$\mathbf{T}_{50}^{PIOU} = 1 - \mathbf{piou}_{50} \quad (3.21)$$

Ce calcul est nécessaire pour obtenir un score qui diminue à mesure que l’IOU diminue.

- la puissance $piou_{fin}$ permet de calculer le seuil T_{fin}^{PIOU} à partir de $piou_{50}$. On aura :

$$\mathbf{T}_{fin}^{PIOU} = 1 - (\mathbf{piou}_{50})^{piou_{fin}} \quad (3.22)$$

- la puissance $piou_0$ permet de calculer le seuil T_0^{PIOU} de suppression de successeur du domaine à partir de $piou_{50}$. On aura :

$$\mathbf{T}_0^{PIOU} = 1 - (\mathbf{piou}_{50})^{piou_0} \quad (3.23)$$

Ainsi dans la figure 3.11, on voit l’application de cette contrainte déterminer le successeur d’une tracklet T entre deux candidats S_1 et S_2 sachant que la dernière détection de T se situe à la trame ($t = 3$) quand les premières de S_1 et S_2 se situent à la trame ($t = 6$). On observe donc dans la seconde image que l’on mesure la vitesse de déplacement de T entre ($t = 2$) et ($t = 3$). Dans la troisième image, on projette des boîtes englobantes en supposant que T conserve exactement la même vitesse. Enfin on mesure l’IOU entre la boîte

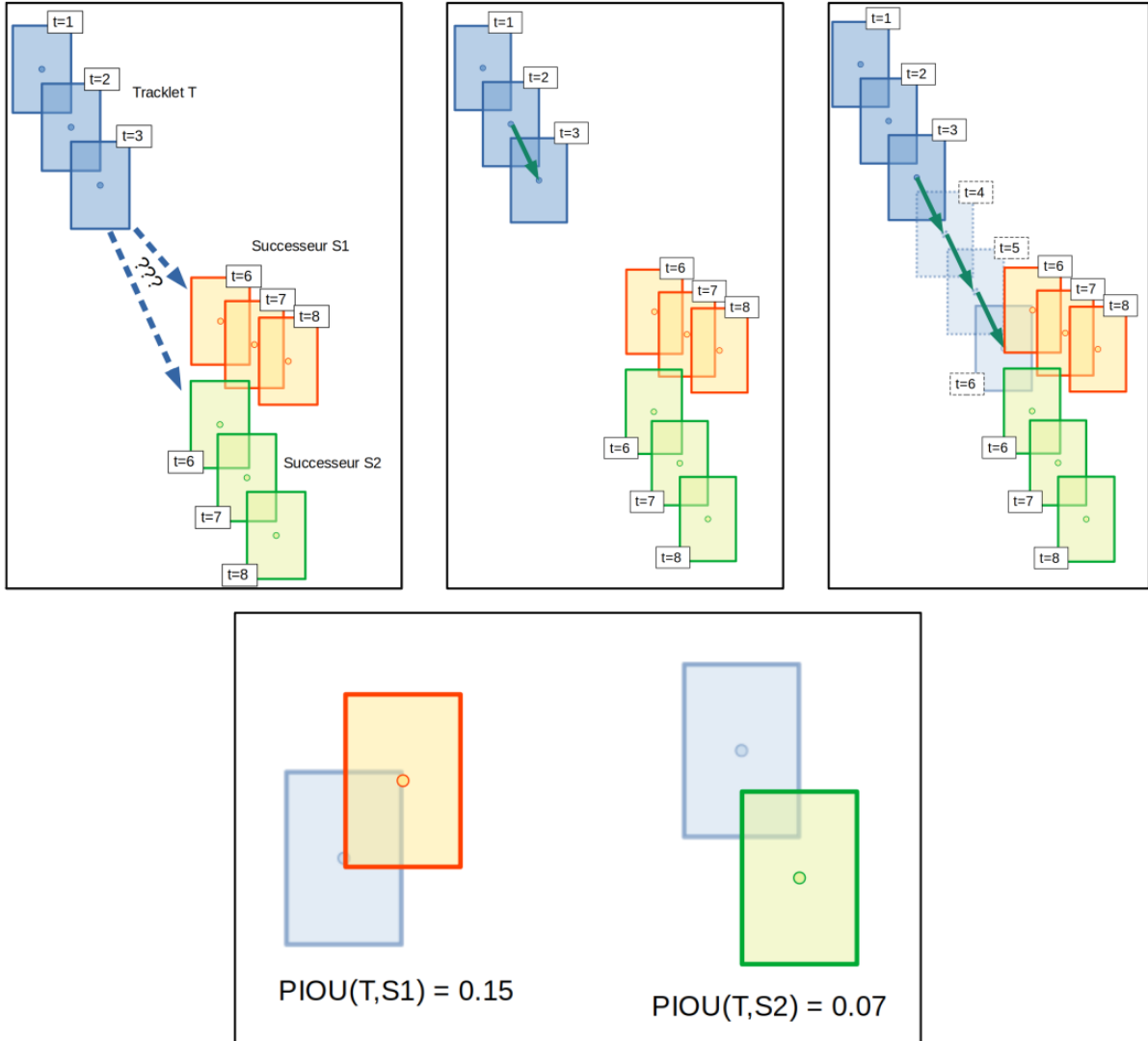


Figure 3.11 Exemple d'application de la contrainte PredictedIOU : Visualisation de la projection de la dernière boîte englobante d'une tracklet sur la trame de début de ses successeurs envisagés

englobante projetée à ($t = 6$) et celles réelles des deux candidats. Ici le score sera plus élevé pour S_1 que pour S_2 pour cette contrainte.

- **PredictedCenterDistance (PCD)** : Cette contrainte suit le même principe que PredictedIOU à ceci près qu'au lieu de projeter l'ensemble de la boîte englobante, on a décidé de ne projeter que la position du centre de l'objet suivi (celui de la boîte englobante en réalité) et la métrique mesurée est donc la distance entre le centre projeté et le centre réel de la première détection du successeur. Cette contrainte s'inspire du modèle CenterTrack proposé par [11],

qui s'appuie justement sur le centre des objets d'intérêt et pas sur les boîtes englobantes, pour limiter l'impact des changements de taille et de forme des détections : Dans le cas d'occlusions progressives, le centre de la détection est en effet moins impacté que la forme de la boîte englobante dont certaines dimensions peuvent très largement varier, comme on le montre dans la figure 2.4. Toutefois, si dans leur cas, le centre est récupéré dès la phase de détection, ce qui leur permet de transporter moins d'informations pour chaque détection, nous construisons les centres a posteriori en prenant le centre des boîtes englobantes. Les paramètres d'entrée sont alors les suivants :

- le facteur pcd_{50} permet de calculer le seuil T_{50}^{PCD} de distance entre les centres des boîtes englobantes à partir duquel la contrainte affectera à l'association un score inférieur à 50%. On aura :

$$\mathbf{T}_{50}^{PCD} = \mathbf{pcd}_{50} \times \mathbf{Diag}(\mathbf{video}) \quad (3.24)$$

On indexe encore une fois ce paramètre sur la diagonale de la vidéo étudiée

- le facteur pcd_{fin} permet de calculer le seuil T_{fin}^{PCD} à partir de T_{50}^{PCD} . On aura :

$$\mathbf{T}_{fin}^{PCD} = \mathbf{pcd}_{fin} * \mathbf{T}_{50}^{PCD} \quad (3.25)$$

- le facteur pcd_0 permet de calculer le seuil T_0^{PCD} de suppression de successeur du domaine à partir de T_{50}^{PCD} . On aura :

$$\mathbf{T}_0^{PCD} = \mathbf{pcd}_0 * \mathbf{T}_{50}^{PCD} \quad (3.26)$$

Sur le même modèle que la figure 3.11, dans la figure 3.12 on cherche à déterminer qui de S_1 ou S_2 mérite le plus d'être associé à T . Ici comme prévu on projette les centres et encore une fois on préfère S_1 à S_2 .

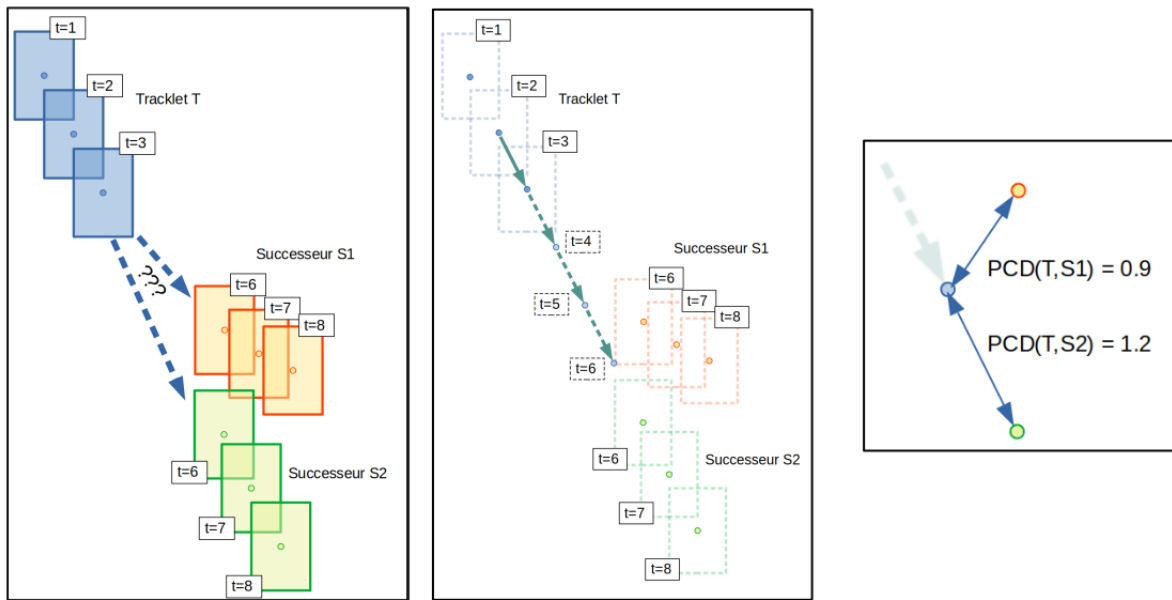


Figure 3.12 Exemple d'application de la contrainte PredictedCenterDistance : Visualisation de la projection du centre de la dernière détection d'une trajectoire sur la trame de début de ses successeurs envisagés

3.3.3 Reconstruction des trajectoires à partir de la liste des successeurs

Par la suite on reconstruit les trajectoires en parcourant la liste fournie grâce à l'algorithme 1.

Algorithme 1 : Des successeurs aux trajectoires

Entrées :

- $T = \{t_0, \dots, t_n\}$ l'ensemble des tracklets, ordonné par ordre chronologique des détections de début des tracklets
- $S : T \rightarrow S(T)$ tel que $\forall t \in T, S(t) \in T \setminus \{t\} \cup \{Stop(t)\}$ la fonction qui à chaque tracklet associe sa variable successeur

Sorties :

- $T_F = \{\{t_a, \dots, t_b\}, \dots, \{t_x, \dots, t_y\}\}$ l'ensemble des trajectoires (suites de tracklets ordonnées temporellement appartenant aux mêmes trajectoires)

début

```

// Initialisation des variables :  $T_F$  les trajectoires
// finales,  $T_C$  la trajectoire en cours,  $T_D$  les trajectoires
// disponibles
 $T_F \leftarrow \emptyset$ 
 $T_C \leftarrow \emptyset$ 
 $T_D \leftarrow T$ 
// Initialisation à la première tracklet
 $t \leftarrow T_D[0]$ 
tant que  $T_D$  non vide faire
    // Ajout de t à la trajectoire en cours
     $T_C \leftarrow T_C \cup \{t\}$ 
    // Retrait de t des tracklets disponibles
     $T_D \leftarrow T_D \setminus \{t\}$ 
    si  $S(t) = Stop(t)$  alors
        // Ajout de la trajectoire en cours aux trajectoires
        // finales
         $T_F \leftarrow T_F \cup \{T_C\}$ 
        // Réinitialisation la trajectoire en cours
         $T_C \leftarrow \emptyset$ 
        si  $T_D$  vide alors
            retourner  $T_F$ 
        sinon
            // Reprise à la première tracklet disponible
             $t \leftarrow T_D[0]$ 
    sinon
        // Reprise au successeur de t
         $t \leftarrow S(t)$ 

```

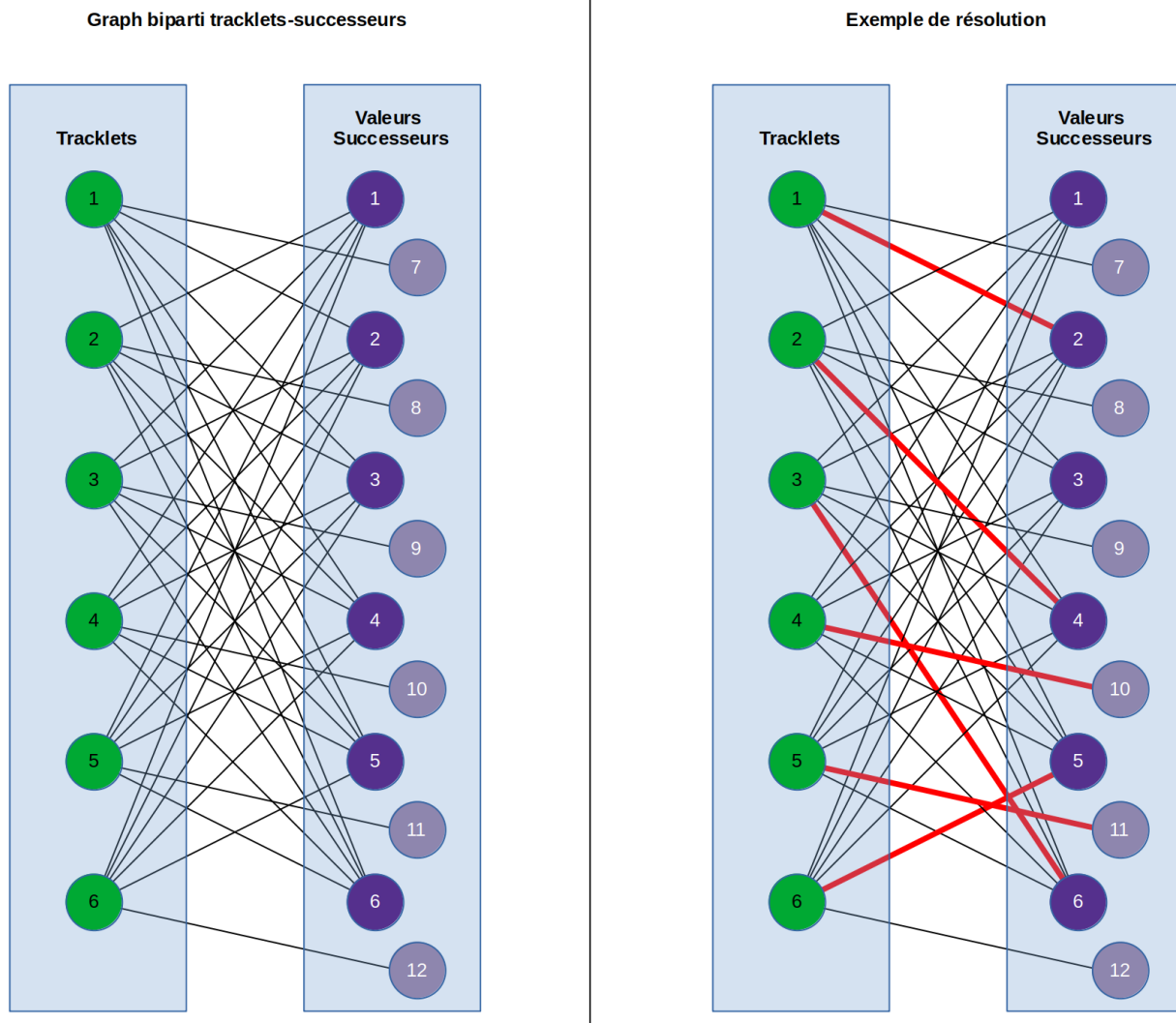


Figure 3.13 Exemple de modélisation du problème d'association par graphe binaire associant les tracklets à leur successeur

Dans le cas de la figure 3.6, on obtient après résolution la figure 3.13 qui nous permet de construire les trajectoires comme suit :

- 1 a pour successeur 2 : on ouvre une trajectoire $T_0 = \{1, 2\}$
- 2 a pour successeur 4 : $T_0 = \{1, 2, 4\}$
- 4 a pour successeur 10 qui est sa variable d'arrêt : T_0 est donc fermée.
- 3 a pour successeur 6 : on ouvre une trajectoire $T_1 = \{3, 6\}$
- 6 a pour successeur 5 : $T_1 = \{3, 6, 5\}$
- 5 a pour successeur 11 qui est sa variable d'arrêt : T_1 est fermée.

On obtient finalement deux trajectoires $T_0 = \{1, 2, 4\}$ et $T_1 = \{3, 6, 5\}$

3.4 Interpolation : combler les trous dans les trajectoires

Enfin une fois nos tracklets associées entre elles, il est probable que les trajectoires résultantes présentent des trous, c'est-à-dire des suites de trames dans lesquelles l'objet disparaît avant de réapparaître. Or cela n'est jamais le cas dans la réalité-terrain, dans laquelle même si l'objet d'intérêt est dissimulé par un obstacle, on aura quand même une boîte englobante de sa position dans l'image. Ainsi, de nombreuses méthodes d'interpolation existent (notamment employées dans *VIOU* [21]) et permettent d'améliorer les résultats de traqueurs déjà performants. C'est pourquoi nous avons décidé d'intégrer un modèle simple d'interpolation dans notre méthode, qui fonctionne comme suit :

- On fixe un unique paramètre : *maxGapSize*, la taille des trous que l'on s'autorise à combler en terme de nombre de trames successives. En effet, si l'objet disparaît dans une seule trame, on peut considérer qu'ajouter une boîte englobante similaire à celles des trames précédente et suivante dans la trame en question ne présente pas un grand risque. En revanche, si le trou à combler s'étend sur 200 trames, si notre association était mauvaise, on introduit énormément de faux positifs.
- On repère les trous dans chaque trajectoire et s'ils sont de taille inférieure au seuil précédemment fixé, *maxGapSize*, on les comble en faisant une interpolation linéaire de la détection précédant le trou à celle lui succédant. Simplement dit, on considère que l'objet s'est déplacé (et a changé de forme ou de dimension) à une vitesse constante de la détection qui précède le trou à celle qui lui succède, et on ajoute toutes les détections manquantes à la liste des détections. Cette méthode est très rudimentaire et pourrait être améliorée en cherchant à minimiser les discontinuités de vitesse, toutefois on cherche à combler des trous suffisamment petits ici pour que la méthode proposée ici permette d'obtenir des résultats proches de ceux obtenus par une méthode plus complète mais plus coûteuse.

Prenons pour exemple les trajectoires contenues dans le tableau 3.1, avec *maxGapSize* = 5. Dans ce tableau, chaque ligne correspond à une détection dont on fournit la trajectoire à laquelle elle est affectée, la trame dans laquelle elle apparaît, un identifiant propre ainsi que les coordonnées de son centre (c_x, c_y) et ses dimensions (w, h).

Dans ce cadre, on observe trois trous qui vont nécessiter des actions différentes :

- Dans la trajectoire 0, on a un trou entre les détections 2 et 3. En effet, la détection 2 se situe dans la trame 2 et la détection 3 se situe dans la trame 6. Le trou est de taille $6 - 2 + 1 = 3 < \text{maxGapSize}$. On effectue donc une interpolation linéaire pour introduire des détections dans les trames 3, 4 et 5. On observe que les dimensions de l'objet d'intérêt ne changent pas donc ici l'interpolation porte exclusivement sur les

Tableau 3.1 Exemple de trajectoires avant interpolation

trajectoire	détection	trame	c_x	c_y	w	h
0	1	1	10	30	20	30
0	2	2	11	32	20	30
0	3	6	15	38	20	30
0	4	7	16	41	20	30
1	5	1	110	10	20	30
1	6	2	120	10	20	30
1	7	4	140	10	30	40
1	8	5	150	10	30	40
2	9	1	220	10	30	40
2	10	2	230	10	30	40
2	11	14	240	10	30	40
2	12	15	250	10	30	40

positions des centres.

- Dans la trajectoire 1, on a un trou de taille 1, qui sera donc interpolé. On observe en revanche un changement dans les dimensions cette fois-ci, qui vont donc demander elles aussi une interpolation linéaire.
- Dans la trajectoire 2, le trou entre les détections 10 et 11 est de taille 11, ce qui dépasse *maxGapSize*. Il n’y aura donc pas ici d’ajout de détections.

Les trajectoires finales après interpolations sont telles que détaillées dans le tableau 3.2.

La figure 3.14 fournit un exemple visuel de ce modèle d’interpolation. On a une trajectoire dont toutes les boîtes englobantes sont de mêmes dimensions, mais qui présente un trou sur deux trames (à $(t = 4)$ et $(t = 5)$). On va donc placer les centres à intervalles réguliers entre ceux aux bords du trou, puis interpoler les boîtes englobantes (qui restent donc de taille constante) et la trajectoire se voit complétée.

Tableau 3.2 Exemple de trajectoires après interpolation avec $maxGapSize = 5$ sur les trajectoires du tableau 3.1

trajectoire	détection	trame	c_x	c_y	w	h
0	1	1	10	30	20	30
0	2	2	11	32	20	30
0	2_1	3	12	33.5	20	20
0	2_2	4	13	35	20	20
0	2_3	5	14	36.5	20	20
0	3	6	15	38	20	30
0	4	7	16	41	20	30
1	5	1	110	10	20	30
1	6	2	120	10	20	30
1	6_1	3	130	10	25	35
1	7	4	140	10	30	40
1	8	5	150	10	30	40
2	9	1	220	10	30	40
2	10	2	230	10	30	40
2	11	14	240	10	30	40
2	12	15	250	10	30	40

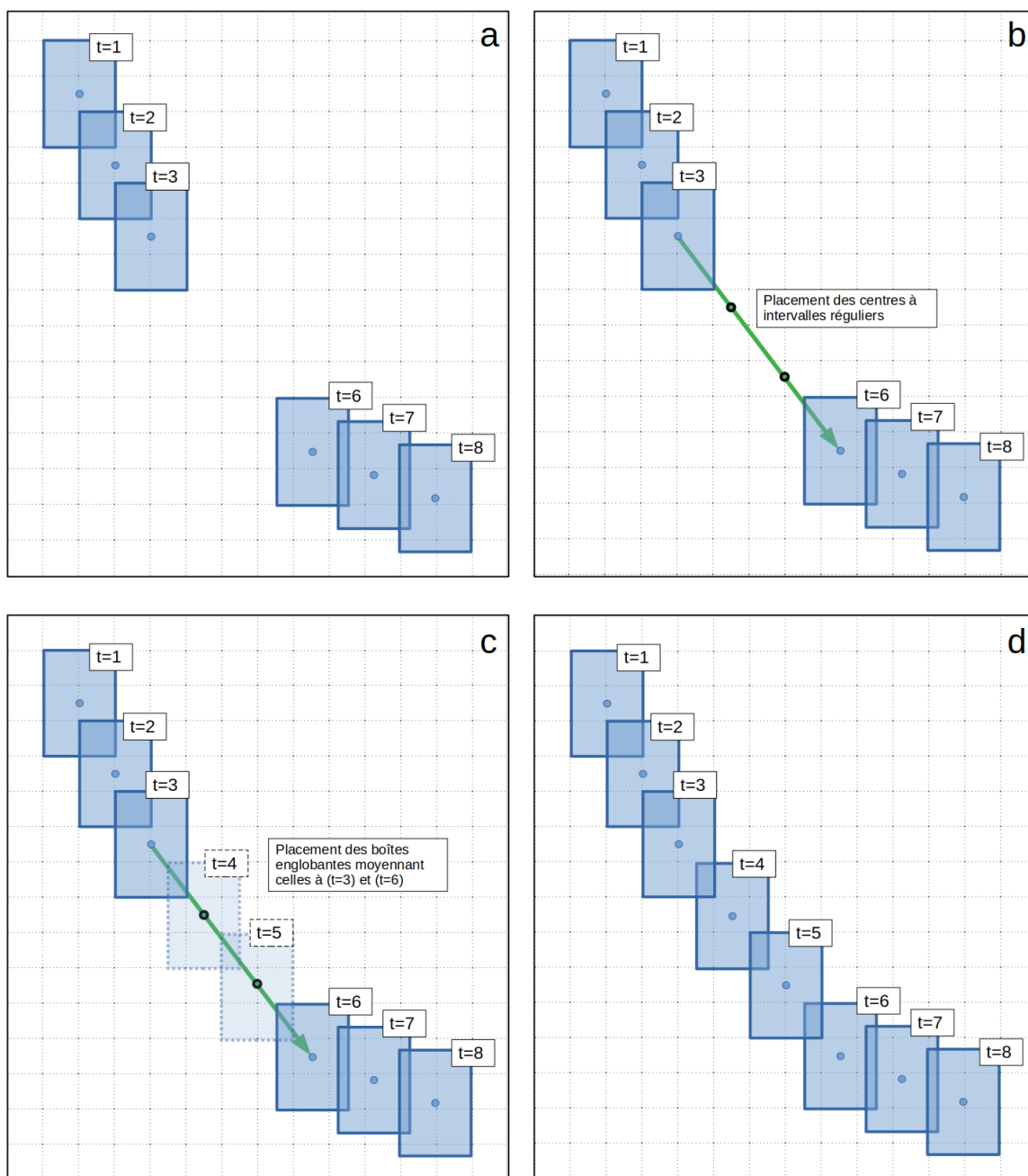


Figure 3.14 Exemple d'application du modèle d'interpolation dans un cas simple

CHAPITRE 4 RÉSULTATS ET DISCUSSIONS

4.1 Métriques et scores choisis

Pour mesurer la performance de notre modèle, et de tout traqueur par ailleurs, il est nécessaire d'établir des métriques pour comparer les résultats obtenus par rapport aux résultats attendus. Pour de nombreux jeux de données (dont ceux sur lesquels nous avons travaillé), les auteurs fournissent en parallèle des détections obtenues par un détecteur (*Faster R-CNN* pour *MOT17* dans notre cas) ainsi que ce que l'on appelle la *réalité-terrain* (ou en anglais "*ground-truth*"). Il s'agit d'une liste exhaustive, relevée à la main, des positions de chaque objet à chaque trame avec leurs identifiants respectifs. Ainsi ces détections sont par construction meilleures que celles qui nous sont fournies par le traqueur et on a accès aux trajectoires complètes. C'est en règle générale en observant la façon dont nos résultats se superposent à ceux de la réalité-terrain que l'on peut évaluer la performance d'un modèle.

Nous avons décidé de mesurer cette performance suivant trois scores principaux :

- **MOTA et MOTP** : des métriques qui mesurent principalement la qualité des détections, et sont utilisées très globalement dans la littérature
- **IDF1** : une métrique qui renvoie plus à la qualité de l'association des détections entre elles, elle aussi est très largement employée dans la littérature
- **HOTA** : une métrique plus récente mais qui rend compte à la fois de la performance en terme d'association de détections et de la qualité de ces détections.

4.2 Définition des concepts

Il convient de poser quelques concepts qui permettront de mieux comprendre les scores qui suivent.

4.2.1 Détections

Durant toute la suite de cette partie, nous emploierons les abréviations suivantes pour parler des détections et des identifiants de trajectoires.

- ***gtDets*** (pour *ground-truth detections*) : ensemble des détections fourni par la réalité-terrain.
- ***prDets*** (pour *predictor detections*) : ensemble des détections renvoyé par notre modèle (détecteur et traqueur), qualifié ici de prédicteur.

- **gtID** (pour *ground truth identifier*) : ensemble des identifiants de trajectoires fourni par la réalité-terrain.
- **prID** (pour *predictor identifier*) : ensemble des identifiants de trajectoires fourni par le prédicteur.

4.2.2 Type d'erreurs

Nous différencions en suivi multi-objet trois types d'erreurs :

- une **erreur de détection** est commise quand une *prDet* ne correspond à aucune *gtDet* ou qu'aucune *gtDet* ne correspond à une *prDet*. Cela signifie soit que le prédicteur a inventé une détection, soit qu'il en a manqué une présente dans la réalité-terrain.
- une **erreur de localisation** est commise quand une *prDet* correspond à une *gtDet* mais n'y est pas parfaitement spatialement alignée.
- une **erreur d'association** est commise quand le traqueur assigne le même *prID* à deux détections qui n'ont pas le même *gtID* ou qu'il n'assigne pas le même *prID* à deux détections qui ont le même *gtID*. Cela signifie que le traqueur a placé dans la même trajectoire deux détections qui ne devraient pas être associées ou qu'il a omis d'en associer qui correspondent pourtant au même objet. On s'y réfère généralement en parlant d'*ID Switches* (ou en français "intersion d'identité"). Pour la suite, la nomenclature employée sera **IDSW**.

4.2.3 Similarité $\mathcal{S}$ et seuil α :

Pour déterminer les erreurs de détection et celles de localisation, on mesure $\mathcal{S}$, la similarité entre les *prDets* et les *gtDets*. Celle-ci est en fait égale à l'*IOU* (*intersection over union*, tel que décrit dans la partie 2.2.1) entre les *prDets* et les *gtDets* qui leur sont associées. Cette association est faite en utilisant l'*algorithme hongrois* [41], méthode très efficace d'optimisation combinatoire, pour maximiser la somme de ces similarités. Une fois cette association bijective des détections du prédicteurs à celle de la vérité-terrain faite, on compare ces scores à un seuil α . Et pour toute paire de détections associée par l'algorithme hongrois telle que $\mathcal{S} \leq \alpha$, on supprime les détections de l'association. Ainsi on peut reformuler la définition des erreurs de détection et de localisation :

- Soit ED l'ensemble des erreurs de détection :

$$ED = \{g \in gtDets \mid \forall p \in prDets, \mathcal{S}(g, p) \leq \alpha\} \cup \{p \in prDets \mid \forall g \in gtDets, \mathcal{S}(g, p) \leq \alpha\} \quad (4.1)$$

- Soit EL l'ensemble des erreurs de localisation et HM (pour *Hungarian Matching*)

l'association de détections fournie par l'algorithme hongrois :

$$EL = \{(g, p) \in HM \mid 1 > \mathcal{S}(g, p) > \alpha\} \quad (4.2)$$

Dans la suite, on parlera, pour toute paire de détections réalité terrain - prédiction (g, p) de $\mathcal{S}$ pour référer à :

$$\mathcal{S} = \mathcal{S}(g, p) = IOU(g, p) \quad (4.3)$$

4.2.4 Classification des détections : TP, FP, FN

Après association par l'algorithme hongrois, les détections sont classées en trois catégories :

- **TP** (pour *True Positive* ou *vrai positif*) : *gtDets* associées avec des *prDets* (avec $\mathcal{S} > \alpha$)
- **FP** (pour *False Positive* ou *faux positif*) : *prDets* associées à aucune *gtDet*.
- **FN** (pour *False Negative* ou *faux négatif*) : *gtDets* associées à aucune *prDet*.

On pourra noter que l'ensemble *ED* des erreurs de détection est l'union des ensembles *FN* et *FP*.

4.3 Une mesure d'apparence : MOTA et MOTP

De nombreuses métriques existent mais les plus employées à ce jour en suivi multi-objet sont définitivement celles de ClearMOT : *MOTA* (pour *Multi Object Tracking Accuracy* ou en français *exactitude du suivi multi-objet*) et du *MOTP* (pour *Multi Object Tracking Precision* ou en français *précision du suivi multi-objet*).

4.3.1 Exactitude et précision (Accuracy and precision) :

Si les deux concepts semblent similaires, ils renvoient à des réalités différentes. L'*exactitude* (ou *accuracy*) se réfère ici à la quantité de détections atteignant un certain seuil de proximité (avec $\mathcal{S} > \alpha$ tel que décrit précédemment) tandis que la précision décrit la qualité de cette proximité.

4.3.2 MOTA

Le MOTA prend en compte trois types d'erreurs : les deux types d'erreurs de détection (*FN* et *FP*) ainsi que les erreurs d'association (*IDSW*). On parle d'*exactitude* (ou *accuracy*) tel que défini plus haut parce qu'ici on cherche à mesurer la distance entre le modèle *parfait* de

la réalité-terrain et les trajectoires proposées par le traqueur suivant la définition suivante :

$$MOTA = 1 - \frac{|FN| + |FP| + |IDSW|}{|gtDet|} \quad (4.4)$$

4.3.3 Erreurs de localisation : MOTP

Pour rendre compte du dernier type d'erreur à ne pas être pris en compte, les **erreurs de localisation**, ClearMOT a proposé le MOTP qui va renvoyer la moyenne des similarités de chaque paire (g, p) associées par l'algorithme hongrois. Le MOTP mesure donc la qualité moyenne des vrais positifs.

$$MOTP = \frac{1}{|TP|} \sum_{TP} \mathcal{S} \quad (4.5)$$

Ces deux scores sont assez fortement orientés vers la qualité des détections, plus que celle de l'association qui n'est prise en compte que partiellement dans le MOTA au niveau des *IDSW*. Pour rendre mieux compte de la qualité de l'association de ces détections entre elles (qui nous concerne plus dans ce mémoire), d'autres métriques existent.

4.4 Une mesure d'association : IDF1

La métrique qui suit tente de mesurer plus efficacement la qualité du résultat au niveau des trajectoires. Toutefois la similarité $\mathcal{S}$ mesurée est encore celle présentée plus haut, faite au niveau des détections.

4.4.1 Classification des détections selon la superposition des trajectoires

Pour la décrire nous devons introduire une nouvelle classification des détections :

- **IDTP** (pour *Identity True Positive* ou *vrai positif d'identification*) : associations dans les parties bien superposées (avec $\mathcal{S} > \alpha$) des trajectoires du prédicteur associées aux trajectoires de la réalité-terrain par l'algorithme hongrois.
- **IDFP** (pour *Identity False Positive* ou *faux positif d'identification*) : prDets appartenant :
 - à des parties mal superposées de trajectoires associées par l'algorithme hongrois.
 - à des trajectoires du prédicteur associées à aucune trajectoire de la réalité-terrain.
- **IDFN** (pour *Identity False Negative* ou *faux négatif d'identification*) : gtDets appartenant :
 - à des parties mal superposées de trajectoires associées par l'algorithme hongrois.
 - à des trajectoires de la réalité-terrain associées à aucune trajectoire du prédicteur.

4.4.2 Calcul du *IDF1*

Et le score *IDF1* est calculé comme suit :

$$IDF1 = \frac{|IDTP|}{|IDTP| + 0.5|IDFN| + 0.5|IDFP|} \quad (4.6)$$

L'association des détections entre elles est encore une fois faite par un algorithme hongrois qui maximise ce score. Ici il fonctionne en énumérant pour chaque association de trajectoires le nombre d'*IDTN* et d'*IDFP* et en choisissant les associations qui minimisent la somme des quantités de ces deux ensembles.

4.5 Le meilleur des deux mondes : *HOTA*

Le score *HOTA* quant à lui permet de rendre compte à la fois de la qualité de la détection et de l'association. Pour le décrire, nous devons de nouveau introduire une classification des détections qui se base sur les *TP*, *FN* et *FP* décrites en partie 4.2.4.

4.5.1 Classification selon les associations

On se référera dans ce qui suit à la classification suivante :

- **TPA** (pour *True Positive Association*) : pour tout $c \in TP$, l'ensemble $TPA(c)$ est l'ensemble des *TP* qui ont le même *gtID* et le même *prID* que c . Il s'agit de l'ensemble des détections proposées par le prédicteur et affectées au bon objet.
- **FPA** (pour *False Positive Association*) : pour tout $c \in TP$, l'ensemble $FPA(c)$ est l'ensemble des détections qui ont le même *prID* que c , mais :
 - ont un *gtID* différent (et sont donc des *TP* en terme de détections mais pointent vers des objets différents).
 - n'ont pas de *gtID* et sont donc des *FP* car elles ne correspondent à aucun objet réel.
- **FNA** (pour *False Negative Association*) : pour tout $c \in TP$, l'ensemble $FNA(c)$ est l'ensemble des détections qui ont le même *gtID* que c , mais :
 - ont un *prID* différent (et sont donc des *TP* en terme de détections mais pointent vers des objets différents).
 - n'ont pas de *prID* (et sont donc des *FN* car elles n'ont pas été détectées).

4.5.2 Calcul du HOTA

À partir de ces classifications on introduit alors le score $HOTA_\alpha$ et le score d'association $\mathcal{A}$ tels que :

$$\forall c \in TP, \mathcal{A}(c) = \frac{|TPA(c)|}{|TPA(c)| + |FNA(c)| + |FPA(c)|} \quad (4.7)$$

$$\forall \alpha \in [0, 1], HOTA_\alpha = \sqrt{\frac{|\sum_{c \in TP} \mathcal{A}(c)|}{|TP| + |FN| + |FP|}}$$

où α est le score d'association permettant d'appartenir ou non aux TP. Le score $HOTA$ final est calculé comme l'intégrale sur α des $HOTA_\alpha$ dont on prend l'approximation suivante pour le calcul :

$$HOTA = \int_0^1 HOTA_\alpha d\alpha \approx \frac{1}{19} \sum_{\alpha \in [0.05, 0.1, \dots, 0.9, 0.95]} HOTA_\alpha \quad (4.8)$$

On pourra citer l'article original qui décrit la méthode [42] pour décrire la pertinence de ce score pour capturer l'ensemble des dimensions du suivi multi-objet : *"HOTA measures how well the trajectories of matching detections align, and averages this over all matching detections, while also penalising detections that don't match."* que l'on pourrait traduire en français par : *"HOTA mesure la qualité de l'alignement des trajectoires des détections associées deux-à-deux et moyenne ceci sur l'ensemble des détections tout en pénalisation les détections qui ne sont pas associées."*

4.6 Jeux de données étudiés : MOT Challenge

Les défis Pour tester et évaluer les différentes méthodes de suivi multi-objet, un certain nombre de bases de données de vidéos annotées à la main existent et sont renouvelées régulièrement pour augmenter la difficulté de la tâche fournie aux traqueurs, en sélectionnant au fur et à mesure des vidéos présentant toujours plus d'occlusions, de reflets et autres caractéristiques rendant le suivi difficile. Elles sont le plus souvent développées dans le cadre de challenges tels que le MOT-Challenge [5–7] pour le suivi de piétons et UA-Detrac [43] pour celui de véhicules automobiles. Ces bases de données fournissent un standard auquel peuvent ainsi se mesurer tous les traqueurs actuels, en s'entraînant sur leurs vidéos d'entraînement (*Training set*) avant de s'appliquer à résoudre les vidéos de test (*Test set*). Le niveau de correspondance entre les résultats des traqueurs et la réalité-terrain (ou *Ground Truth*) mesurée à la main est alors mesuré par les challenges qui mettent régulièrement leurs classements à jour.

Format des détections Le format des détections fournies par les challenges est ainsi le suivant : (f, id, x, y, w, h, s) tels que :

- f correspond à l'indice entier de la trame à laquelle apparaît la détection.
- id correspond à l'indice de l'objet ou de la trajectoire à laquelle est affectée la détection. Cet indice est fixé à -1 pour l'ensemble des boîtes englobantes lorsque les détections sont fournies dans la mesure où les détections ne sont encore affectées à aucune trajectoire.
- (x, y) sont les coordonnées du coin haut-gauche de la boîte englobante qui encadre la détection
- (w, h) correspondent respectivement à la largeur (w pour "*width*") et la hauteur (h pour "*height*") de la boîte englobante.
- s correspond au score de confiance affecté par le détecteur à la détection en question. Cette information est en effet utilisée à certains traqueurs qui suppriment les détections les moins sûres par exemple.

L'objectif de la phase d'association est alors d'affecter un identifiant à chacune de ces détections, potentiellement d'en ajouter ou d'en supprimer, de sorte à se rapprocher au maximum de l'idéal d'un identifiant unique par objet d'intérêt tout le long de la vidéo et d'être au plus près de la *réalité-terrain*.

Le choix de MOT17 Nous avons choisi de calibrer notre modèle sur les données d'entraînement du défi MOT17 [5] du MOT-Challenge en se basant sur les détections du meilleur détecteur proposé, FRCNN [10], et sur les résultats du *IOU-Tracker*. Cette base de données représente une référence dans le domaine et présente de nombreuses difficultés auxquelles il est particulièrement intéressant de se confronter. Qu'il s'agisse des taux d'occlusions souvent extrêmement élevés, des turbulences, des caméras en mouvement, des fortes variations d'exposition lumineuse, des points de vue parfois subjectifs et d'autres très surélevés, ou des nombreux reflets présents dans ces vidéos, on a affaire à des situations extrêmement variées qui doivent permettre au modèle de s'adapter à la majorité de ce que peuvent proposer des vidéos en cadre urbain.

4.7 Calibration du modèle

Notre modèle a un nombre de paramètres assez élevé et avant de le tester, il convenait de régler leurs valeurs pour obtenir des résultats intéressants. Les risques auxquels on s'expose sans calibration sont les suivants :

- avec un **modèle d'association** mal calibré par exemple, on risquerait de n'obte-

nir soit aucune association (si le modèle est trop discriminant, on risque de préférer systématiquement arrêter les trajectoires à la première tracklet), soit de donner trop de poids à une contrainte par rapport aux autres et donc d’obtenir des associations erronées.

- avec un **modèle de découpage de tracklets** mal calibré, on risque soit de ne couper nulle part, soit de couper les trajectoires en des zones qui n’étaient pas problématiques mais que notre modèle d’association ne saurait pas bien associer. On empirerait alors les résultats.
- avec un **modèle d’interpolation** mal calibré, on risque soit de ne jamais effectuer d’interpolation, soit de propager des erreurs en remplissant des trous énormes qui se situent entre des tracklets mal associées.

Toutefois, trouver les valeurs optimales de l’ensemble des paramètres du modèle demande beaucoup de temps et il a fallu chercher comment plutôt en obtenir une approximation. En effet, tenter de fouiller avec une granularité suffisamment élevée l’ensemble de l’espace représenté par nos dix-sept paramètres (sachant que le temps de calcul pour l’ensemble des vidéos est de l’ordre de la minute) semblait non seulement sous-optimal mais même impossible.

4.7.1 Procédure de calibration

Nous avons donc fait l’hypothèse que nos trois modules (association, découpage de tracklets et interpolation) sont suffisamment indépendants pour que les valeurs optimales des paramètres du modèle d’association soient les mêmes, qu’il soit utilisé en conjonction ou non avec les modules de découpage et d’interpolation. Nous avons commencé par calibrer le modèle d’association, récupéré les combinaisons de contraintes et les valeurs des paramètres permettant d’obtenir les meilleurs résultats en terme de HOTA, avant d’employer ce modèle calibré pour optimiser les paramètres du *TrackletCutter*, puis de l’interpolation.

4.7.2 Calibration du modèle d’association

Pour calibrer le modèle d’association, nous avons divisé l’exploration en trois étapes que nous présentons ci-dessous et que nous allons en partie illustrer à l’aide de cartes de chaleur.

Affichage des résultats par cartes de chaleur Après nous être rendus compte que l’obtention d’une solution pour l’ensemble des vidéos du défi MOT17 prenait en moyenne moins d’une minute, nous avons décidé de ne pas activer la fonction d’élagage des contraintes (le T_0^C , qui permet pour chacune des contraintes de supprimer un successeur potentiel du

domaine d'une tracklet s'il dépasse un certain seuil pour la caractéristique évaluée par la contrainte) et donc de faire une calibration sur l'effet des deux autres paramètres décrits en partie 3.3.2 pour chaque contrainte C : T_{50}^C et T_{fin}^C . Ainsi dans les tests de calibration du modèle d'association, on a cherché à faire varier ces deux paramètres pour chaque contrainte testée et nos résultats sont présentés sous forme de cartes de chaleur pour chaque vidéo du défi MOT17 ainsi qu'en version *combinée* (notée "COMBINED") qui correspond au score sur l'ensemble des vidéos à la fois. Ces cartes de chaleur représentent le score $HOTA$ pour chaque configuration et sont à comprendre comme suit :

- Les ordonnées représentent le paramètre f_{50} (resp. t_{50} selon la contrainte) et les abscisses le paramètre f_{fin} (resp. p_{fin} selon la contrainte).
- les cases de la carte de chaleur sont rouges si elles représentent des résultats **moins bons** que ceux d'*IOU-Tracker* (le traqueur de base) et **bleues** si elles représentent des résultats **meilleurs**. Les cases tendant vers le gris correspondent à des résultats équivalents.

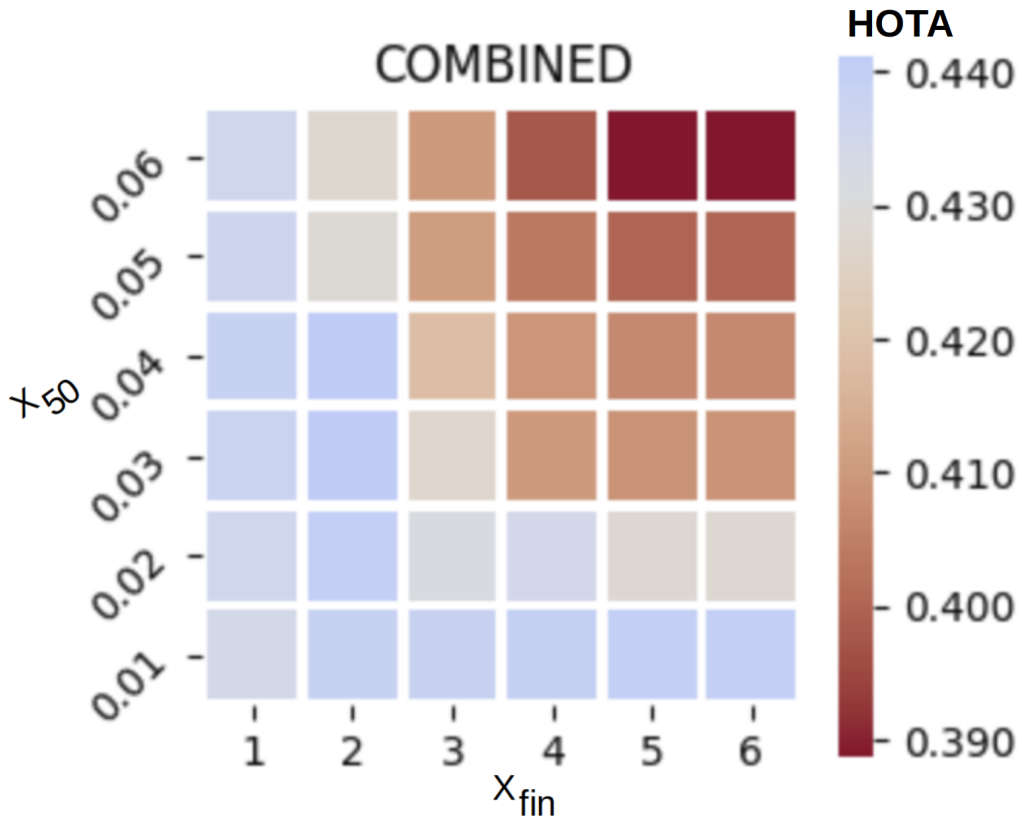


Figure 4.1 Exemple de carte de chaleur des scores HOTA pour une contrainte fictive X

Ainsi si on prend l'exemple de la figure 4.1, on peut voir les résultats d'une variation des paramètres fictifs X_{50} et X_{fin} d'une contrainte sur l'ensemble des vidéos du MOT17 (d'où

l'emploi du terme "COMBINED" qui signifie combiné). On peut constater des résultats moins bons que ceux d'IOU-Tracker dans le coin en haut à droite (pour $X_{fin} \geq 3$ et $X_{50} \geq 0.03$) et les pires résultats pour les valeurs les plus élevées de ces deux paramètres. Pour les valeurs les plus faibles de ces deux paramètres en revanche, on observe des cases bleues claires, ce qui renvoie à des résultats faiblement meilleurs que ceux du traqueur initial.

Test des contraintes une par une Tout d'abord nous avons testé l'effet de chaque contrainte si on l'utilisait seule (mais en conjonction avec *AllDifferent*, sans quoi les résultats n'auraient aucun sens) pour voir si les résultats correspondaient à nos attentes et avoir une première idée des zones à explorer pour chaque paramètre.

Par construction, on pouvait s'attendre à ce que les trois contraintes *AngleDifference*, *SpeedNormDifference* et *TimeDistance* ne fournissent pas de solutions améliorantes lorsqu'elles sont utilisées individuellement, comme nous en reparlerons dans la partie 4.8.2 et qu'il serait nécessaire de les calibrer à la lumière de leur association avec d'autres contraintes.

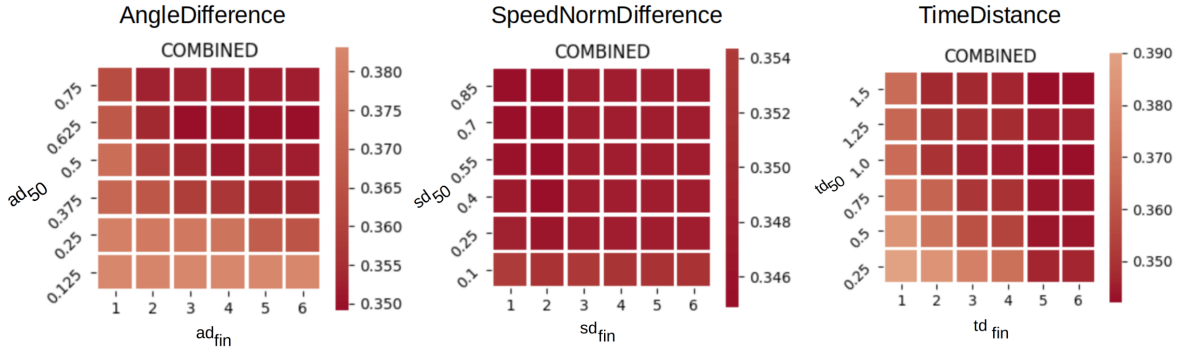


Figure 4.2 Cartes de chaleur sur l'ensemble des vidéos pour les contraintes *AngleDifference*, *TimeDistance* et de *TimeDistance* utilisées seules

Dans la figure 4.2, on observe comme attendu dans le cas de ces 3 contraintes prises individuellement qu'aucun choix de paramètres ne permet d'obtenir de solution améliorante en moyenne. Comme prévu, on a décidé d'attendre d'employer ces contraintes en conjonction avec d'autres pour tirer de réelles conclusions et de pouvoir vraiment les calibrer.

La figure 4.3 montre qu'en revanche, les deux autres contraintes *PredictedIOU* et *PredictedCenterDistance* semblaient, même employées seules, plus prometteuses. En effet celles-ci pénalisent à la fois l'éloignement temporel et spatial et devaient permettre des améliorations au moins dans certaines configurations. Pour *PredictedCenterDistance*, à gauche de la figure 4.3, on observe donc bien des résultats améliorants et la zone qui en compte pour chaque

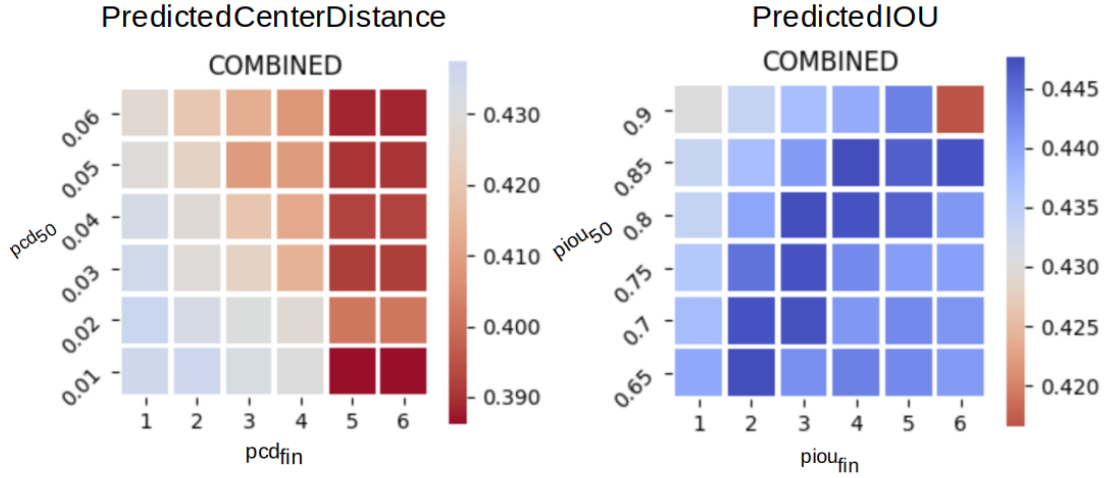


Figure 4.3 Cartes de chaleur sur l'ensemble des vidéos pour les contraintes *PredictedCenterDistance*, *PredictedIOU* utilisées seules

vidéo est la partie de l'espace des paramètres telle que :

$$\begin{cases} pcd_{50} \leq 0.03 \\ pcd_{fin} \leq 3 \end{cases} \quad (4.9)$$

C'est la raison pour laquelle pour calibrer les autres contraintes en conjonction avec *PredictedCenterDistance*, nous fixerons ses paramètres dans cette portion de l'espace à $pcd_{50} = 0.02$ et $pcd_{fin} = 2$.

Pour *PredictedIOU*, à droite de la figure 4.3, on observe aussi comme on s'y attendait des résultats améliorants. De plus on peut noter une similarité dans les résultats pour un certain rapport de proportions entre $piou_{50}$ et $piou_{fin}$, c'est-à-dire qu'on observe des diagonales allant du coin bas-gauche vers le coin haut-droit des cartes de chaleur se dessiner. On observe tout de même qu'en général, les meilleurs résultats sont obtenus pour :

$$\begin{cases} piou_{50} \leq 0.85 \\ piou_{fin} \geq 2 \end{cases} \quad (4.10)$$

et particulièrement sur la diagonale allant de bas en haut de gauche à droite. Ainsi on fixera les valeurs des paramètres de cette contrainte à $piou_{50} = 0.80$ et $piou_{fin} = 4$ pour aider au calibration des autres dans ce balayage.

Sélection de plages et de contraintes Une fois ces deux versions par défaut de *PredictedCenterDistance* et *PredictedIOU* fixées, nous avons pu les utiliser pour calibrer un peu plus efficacement *TimeDistance* en faisant varier ses paramètres en conjonction avec chacune de ces deux contraintes, l'une après l'autre. Comme précédemment on extrait une plage efficace de valeurs :

$$\begin{cases} td_{50} \in [0.5, 1.25] \\ td_{fin} \geq 4 \end{cases} \quad (4.11)$$

On a alors pu fixer sur le même modèle des valeurs par défaut à *TimeDistance* et l'employer pour obtenir trois calibrations de contraintes par défaut qui nous servent à calibrer les autres :

Tableau 4.1 Valeurs par défaut des paramètres des contraintes *TimeDistance*, *PredictedCenterDistance* et *PredictedIOU*

Contrainte	Paramètres par défaut	
<i>TimeDistance</i>	$td_{50} = 0.75$	$td_{fin} = 2$
<i>PredictedCenterDistance</i>	$pcd_{50} = 0.02$	$pcd_{fin} = 2$
<i>PredictedIOU</i>	$piou_{50} = 0.80$	$piou_{fin} = 4$

Partant de ces valeurs par défaut, on a alors cherché à optimiser les valeurs des paramètres de chaque contrainte en l'associant avec des combinaisons des trois précédemment citées, pour chercher à extraire les plages de paramètres les plus efficaces pour chaque contrainte lorsqu'elle est utilisée dans diverses situations. Les résultats de ce criblage nous en partie permis de mener notre étude sur la sensibilité du modèle aux variations de paramètres détaillée dans la partie 4.8.2. Nous avons par ailleurs essayé de faire varier le nombre de solutions valides à explorer avant de s'arrêter dans notre recherche et testé les deux types de stratégies de branchement *LDS* et *DFS* décrites dans la partie 3.3.1 au paragraphe "Stratégies d'exploration".

Un second balayage pour régler les interactions entre contraintes

On a essayé d'associer directement les meilleures configurations de chaque contrainte ensemble mais il s'est avéré que cela ne permettait pas une amélioration des résultats car on effectuait alors trop d'associations, c'est-à-dire que l'on construisait des trajectoires trop longues. On aurait pu s'y attendre dans la mesure où, comme nos scores sont le produit des scores renvoyés par chaque contrainte, le seuil (T_{fin}^C) auquel placer l'arrêt d'une trajectoire pour une contrainte C efficace lorsque la contrainte est seule est à réduire lorsque l'on emploie de multiples contraintes, sans quoi il sera dur d'atteindre un score suffisamment mauvais sur chaque contrainte pour préférer l'arrêt de la trajectoire à son prolongement.

En bref, on a décidé de repartir des meilleures zones de recherche et de balayer l'espace de

recherche dans ces zones déjà restreintes. Nous avons de plus décidé de n’effectuer cette calibration que sur les 3 contraintes *TimeDistance*, *PredictedCenterDistance* et *PredictedIOU* dans la mesure où ce sont les trois contraintes qui permettent le plus d’améliorations et où cela nous permettrait d’optimiser leurs paramètres plus efficacement. D’autre part, les tests menés précédemment nous ont poussé à fixer l’arrêt de l’exploration des solutions à la première solution (car on n’observait pas d’amélioration en poussant plus loin) et de poursuivre avec la stratégie d’exploration *DFS*. Cela nous a mené à chercher dans l’espace des paramètres suivants qui correspond à l’union des plages qui donnaient les meilleurs résultats jusqu’alors :

- pour *TimeDistance* :
 - utilisée ou non
 - si utilisée : $td_{50} \in \{0.5, 0.75, 1, 1.25\}$ et $td_{fin} \in \{1, 2, 3, 4\}$
- pour *PredictedCenterDistance* :
 - utilisée ou non
 - si utilisée : $pcd_{50} \in \{0.01, 0.02, 0.03, 0.04\}$ et $pcd_{fin} \in \{2, 3, 4, 5\}$
- pour *PredictedIOU* :
 - utilisée ou non
 - si utilisée : $piou_{50} \in \{0.65, 0.7, 0.75, 0.8\}$ et $piou_{fin} \in \{2, 3, 4, 5\}$

La meilleure configuration que nous avons trouvée lors de cette recherche est visible dans le tableau 4.2.

Tableau 4.2 Valeurs par défaut des paramètres des contraintes *TimeDistance*, *PredictedCenterDistance* et *PredictedIOU*

Contrainte	Paramètres de HOTA maximal	
<i>TimeDistance</i>	$td_{50} = 1$	$td_{fin} = 3$
<i>PredictedCenterDistance</i>	$pcd_{50} = 0.02$	$pcd_{fin} = 2$
<i>PredictedIOU</i>	$piou_{50} = 0.75$	$piou_{fin} = 2$

4.7.3 Calibrations du *TrackletCutter* et du module d’interpolation

Une fois cette calibration du modèle d’association finie, il nous restait à calibrer nos deux modules supplémentaires, celui du découpage de tracklets (le *TrackletCutter*) et celui de l’interpolation.

Calibration du *TrackletCutter* Pour le *TrackletCutter*, tel que décrit dans la section 3.1.2, nous avons seulement un paramètre à calibrer : T_{TC} , l’IOU des détections appartenant à la même trame à partir duquel on coupe les trajectoires qui les contiennent car on suppose qu’une occlusion est probable. Pour le calibrer, nous avons construit 20 jeux de détections

en appliquant 20 seuils (de 0.05 à 1 par incréments de 0.05) de base aux détections fournies par le défi *MOT17* pour le détecteur *FRCNN*. Nous avons ensuite appliqué à ces 20 jeux de données nos 100 meilleurs modèles (déterminés lors de la phase de calibration précédente). Les meilleurs résultats ont été obtenus en moyenne pour $T_{TC} = 0.5$ et c'est donc la valeur que l'on a conservée lorsque l'on a appliqué le *TrackletCutter* pour les tests suivants.

Calibration du modèle d'interpolation Pour la calibration du modèle d'interpolation, nous avons repris nos meilleurs résultats, avec et sans découpage de tracklets, pour tester différentes valeurs pour le paramètre du modèle décrit dans la section 3.4 : *maxGapSize*, qui représente la taille maximale (en nombre de trames successives) des trous présents dans des trajectoires que l'on souhaite combler par interpolation linéaire. Comme pour le *TrackletCutter* on appliqué le modèle à différents seuils (de *maxGapSize* = 3 à *maxGapSize* = 150 en allant par incréments de 3) sur les meilleurs résultats obtenus aux étapes précédentes et conservé les meilleurs valeurs de seuil. Comme on le verra dans la partie 4.8.2, on n'a ici pas conservé le seuil permettant d'obtenir le meilleur *HOTA* en moyenne car il représentait une trop grosse perte pour certaines vidéos (et donc un risque accru de ne pas se généraliser à d'autres traqueurs ou d'autres vidéos) mais nous sommes arrêtés à un seuil un peu plus faible qui lui présentait moins de dégâts pour des résultats presque aussi bons. On a donc fixé *maxGapSize* = 42, seuil auquel on obtient la majorité de nos améliorations sans être trop pénalisés sur les vidéos pour lesquelles le modèle semble moins efficace.

4.8 Résultats et discussion

4.8.1 Résultats en ablation sur *IOU-Tracker*, *SORT* et *Tracktor* et discussion sur la contribution des composantes du modèle

Nous avons testé notre modèle final en ablation pour trois traqueurs sur le défi MOT17 pour vérifier la généralisation de nos résultats à plusieurs traqueurs. Les résultats sont fournis au tableau 4.3. Et les configurations employées pour ces tests sont accessibles dans le tableau 4.4.

On observe donc tout d'abord que quel que soit le traqueur auquel on l'applique, notre modèle permet d'obtenir des améliorations de plusieurs pourcent sur les trois scores qui nous intéressent : *HOTA*, *MOTA* et *IDF1*. Concernant le *HOTA*, métrique principale de notre évaluation, on arrive à une amélioration de 3.14% pour *IOU-Tracker*, 3.55% pour *SORT* et 1.8% pour *Tracktor* qui avait pourtant déjà un score plutôt élevé (13% de plus que les deux autres au départ) ce qui montre que notre module est efficace aussi sur tout type de traqueur

Tableau 4.3 Résultats des différentes composantes du modèle en ablation sur trois traqueurs différents appliqués aux vidéos du MOT17

Défi	Méthode	HOTA	MOTA	IDF1
MOT17	IOU-Tracker	43.04%	49.74%	50.27%
	IOU-Tracker + CP	45.34%	49.91%	53.78%
	IOU-Tracker + TC + CP	45.47%	49.90%	54.11%
	IOU-Tracker + CP + Int	46.04%	50.62%	54.38%
	IOU-Tracker + TC + CP + Int	46.18%	50.49%	54.74%
	SORT	42.80%	48.54%	50.63%
	SORT + CP	45.40%	48.71%	54.34%
	SORT + TC + CP	45.14%	48.70%	53.96%
	SORT + CP + Int	46.35%	49.46%	54.92%
	SORT + TC + CP + Int	46.06%	49.39%	54.48%
	Tracktor	55.18%	61.81%	65.06%
	Tracktor + CP	56.39%	61.87%	67.40%
	Tracktor + TC + CP	55.89%	61.85%	66.77%
	Tracktor + CP + Int	56.98%	62.99%	67.88%
	Tracktor + TC + CP + Int	56.49%	62.96%	67.29%

indépendamment de leur niveau de performance initial.

On observe que le modèle d’association est celui qui permet le plus d’améliorations en terme de *HOTA* et d’*IDF1* (70% des améliorations en moyenne ici) aux résultats des trois traqueurs sans influencer ou presque sur le *MOTA*, ce à quoi on aurait pu s’attendre dans la mesure où il s’agit des deux métriques les plus sensibles à la qualité des associations. Le reste des améliorations est majoritairement apporté par le modèle d’interpolation qui permet d’améliorer le *MOTA* de plus d’un point pour *Tracktor* notamment.

IOU-Tracker est toutefois le seul traqueur pour lequel le *TrackletCutter* permet réellement une amélioration des résultats. Peut-être est-ce dû au fait que ce traqueur présente plus d’erreurs dues aux occlusions repérées par notre méthode. Toujours est-il qu’il semblerait que notre modèle soit plus performant sans le découpage de tracklets. Il s’avère qu’il s’agit de la partie du modèle qui requière le plus grand temps de calcul, pour peu, voire pas, d’améliorations. Avant de lui avoir apporté quelques modifications, nous conseillerions donc de ne pas l’employer pour d’autres traqueurs qu’*IOU-Tracker*.

Concernant l’exploration des solutions, il s’est avéré qu’en poussant celle-ci même jusqu’à la 10 000ème solution explorée pour de multiples configurations de paramètres et de contraintes, on n’obtenait pas de meilleurs résultats qu’en s’arrêtant à la première trouvée par le modèle en choisissant les marginales maximales (sauf pour les modèles très mauvais, qui obtenaient dans tous les cas des solutions moins performantes que le traqueur *IOU-Tracker* initial),

Tableau 4.4 Configurations des différents modules du modèle final employés dans les tests en ablation

	Module	Configuration du modèle final
Association (CP)	TimeDistance	$td_{50} = 1$
		$td_{fin} = 3$
	PredictedCenterDistance	$pcd_{50} = 0.02$
		$pcd_{fin} = 2$
	PredictedIOU	$piou_{50} = 0.75$
		$piou_{fin} = 2$
	<i>AngleDifference</i>	désactivée
	<i>SpeedNormDifference</i>	désactivée
	Stratégie d’exploration	DFS
	Solution arrêtant l’exploration	1
Découpage de tracklets : <i>TrackletCutter</i>		$T_{TC} = 0.5$
Interpolation		$maxGapSize = 42$

donc on a décidé de s’arrêter systématiquement à la première solution valide dans notre exploration. On n’a en outre pas observé de différence notable en terme de scores entre les deux stratégies d’exploration étudiées, ce à quoi on pouvait s’attendre : la première solution est la même pour les deux stratégies et dans notre cas, c’est systématiquement la meilleure. On a donc poursuivi nos tests avec *DFS*.

4.8.2 Discussion sur la sensibilité des résultats aux paramètres des contraintes

Lors de la calibration de notre modèle, nous avons étudié les effets des différents paramètres des contraintes sur les scores et cherché les plages de valeurs qui permettaient d’optimiser ces scores. Dans cette section, nous étudions donc la sensibilité de notre méthode aux variations des paramètres et contraintes sur l’ensemble de test de MOT17. Cette sensibilité peut être présentée sous la forme des cartes de chaleur présentés dans la section 4.7. Pour évaluer la sensibilité des résultats à chaque contrainte et à ses paramètres, nous avons récupéré les résultats des tests faits lors de la calibration, qui évaluaient l’efficacité des contraintes lorsqu’on les utilise individuellement en conjonction avec *AllDifferent* puis en conjonction les unes avec les autres en fixant les paramètres des autres à des valeurs dans leurs plages d’efficacité respectives. Les paramètres par défaut affectés aux contraintes *TimeDistance*, *PredictedCenterDistance* et *PredictedIOU* lorsqu’elles sont employées pour aider à tester d’autres contraintes sont compilées dans le tableau 4.5.

Tableau 4.5 Valeurs par défaut des paramètres des contraintes *TimeDistance*, *PredictedCenterDistance*, *PredictedIOU* utilisées pour les tests de sensibilité du modèle aux contraintes

Contrainte	Valeurs des paramètres par défaut	
<i>TimeDistance</i>	$td_{50} = 0.75$	$td_{fin} = 2$
<i>PredictedCenterDistance</i>	$pcd_{50} = 0.02$	$pcd_{fin} = 2$
<i>PredictedIOU</i>	$piou_{50} = 0.80$	$piou_{fin} = 4$

Sensibilité aux paramètres de *PredictedCenterDistance* La figure 4.4 qui représente les résultats obtenus lorsque la contrainte est utilisée seule permet de noter que même utilisée seule, la contrainte *PredictedCenterDistance* a une plage de paramètres qui permet d'obtenir presque partout des résultats améliorants par rapport au traqueur original. Il s'agit de la zone suivante :

$$\begin{cases} pcd_{50} \leq 0.03 \\ pcd_{fin} \leq 3 \end{cases} \quad (4.12)$$

On note toutefois que cela va dans le sens du constat qui dit que lorsque l'on utilise une contrainte seule, avoir un T_{50}^C et un T_{fin}^C faibles permet d'obtenir de meilleurs résultats (pas forcément améliorants cela dit) dans la mesure où cela cantonne les associations aux cas où la similarité entre les tracklets associées est extrêmement élevée selon la caractéristique évaluée. Toutefois on ne peut en général pas envisager de résultats très améliorants, dans la mesure où cela ne permet que peu d'associations.

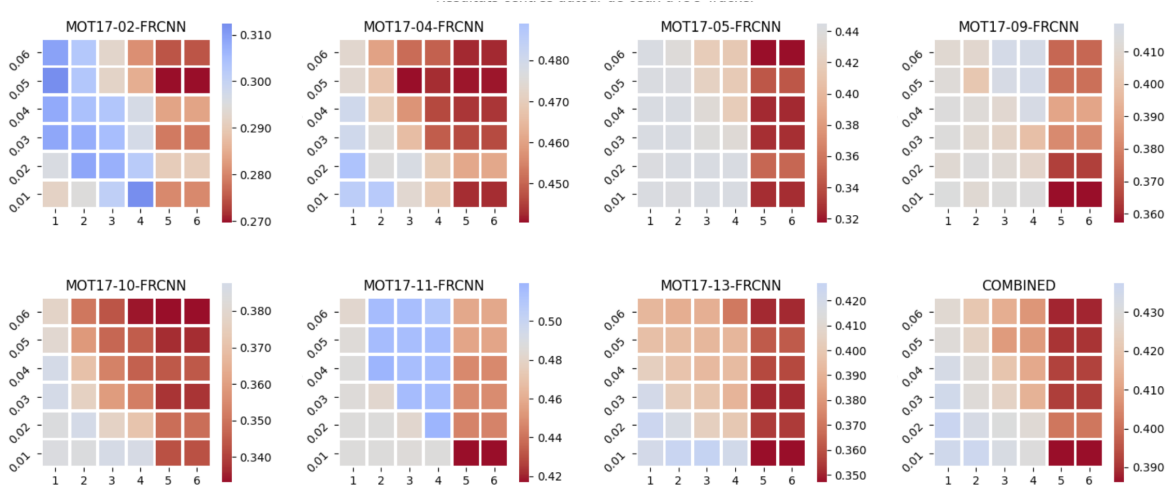


Figure 4.4 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *PredictedCenterDistance* utilisée seule

On a donc voulu tester cette contrainte en conjonction avec les versions par défaut de *PredictedIOU* et *TimeDistance*, et les résultats de ces tests sont visibles dans les figures 4.5 et

4.6. Lors de l'interaction avec *PredictedIOU* dans la figure 4.5, on observe des effets assez

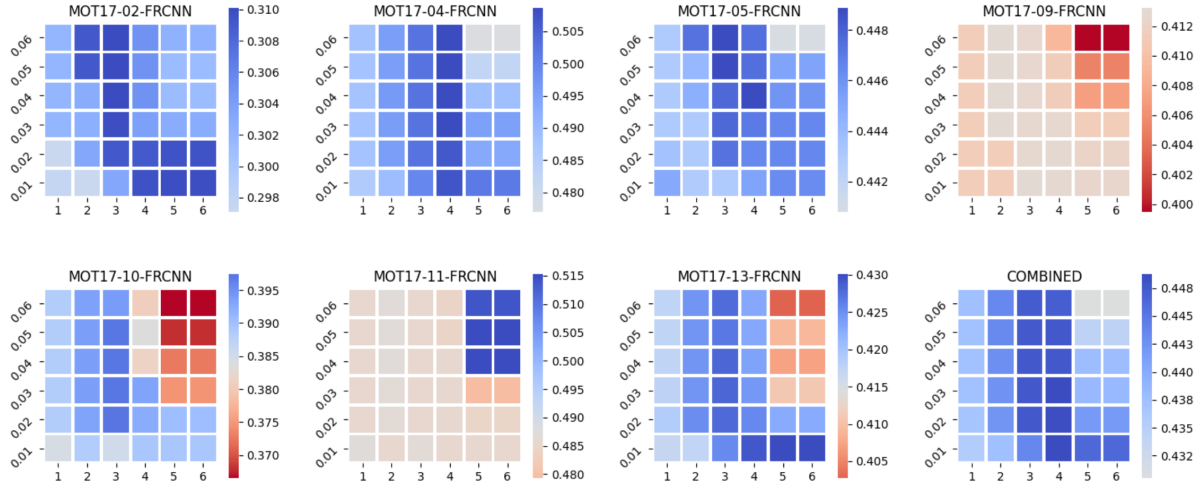


Figure 4.5 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *PredictedCenterDistance* utilisée en conjonction avec *PredictedIOU*

précis qui permettent d'obtenir des résultats très améliorants :

- on peut noter une sensibilité à pcd_{fin} donc au rapport entre la distance permettant d'obtenir un score de 50% sur la contrainte et celle qui détermine qu'il vaut mieux stopper la trajectoire, de sorte à ce que les résultats sont en général améliorés lorsque ce paramètre augmente jusqu'à 4 puis connaissent une chute brusque. Les valeurs de pcd_{fin} à favoriser sont donc comprises entre 2 et 4.
- en parallèle, on observe de meilleurs résultats pour pcd_{50} inférieur à 0.02.

Les résultats de l'association avec *TimeDistance* nous informent moins dans la mesure où ils sont plus faibles et plus variables d'une vidéo à l'autre. On conserve toutefois une préférence pour les valeurs les plus faibles de pcd_{50} (0.01 ou 0.02).

Notre modèle est donc assez sensible aux paramètres de cette contrainte, qu'il convient de maintenir dans $[2, 4]$ pour pcd_{end} et $[0.01, 0.02]$ pour pcd_{50} , et cette contrainte semble fonctionner au maximum de sa performance en conjonction avec *PredictedIOU*. En effet, les scores maximaux alors obtenus dépassent en moyenne 0.448 en conjonction avec *PredictedIOU* quand ils ne dépassaient pas 0.441 sans elle (comme on le voit dans la carte de chaleur COMBINED des figures 4.6 et 4.4).

Sensibilité aux paramètres de *PredictedIOU* La figure 4.7 qui représente les résultats obtenus lorsque la contrainte est utilisée seule permet de visualiser la large plage de paramètres qui permet d'obtenir presque partout des résultats améliorants (déjà très amélio-

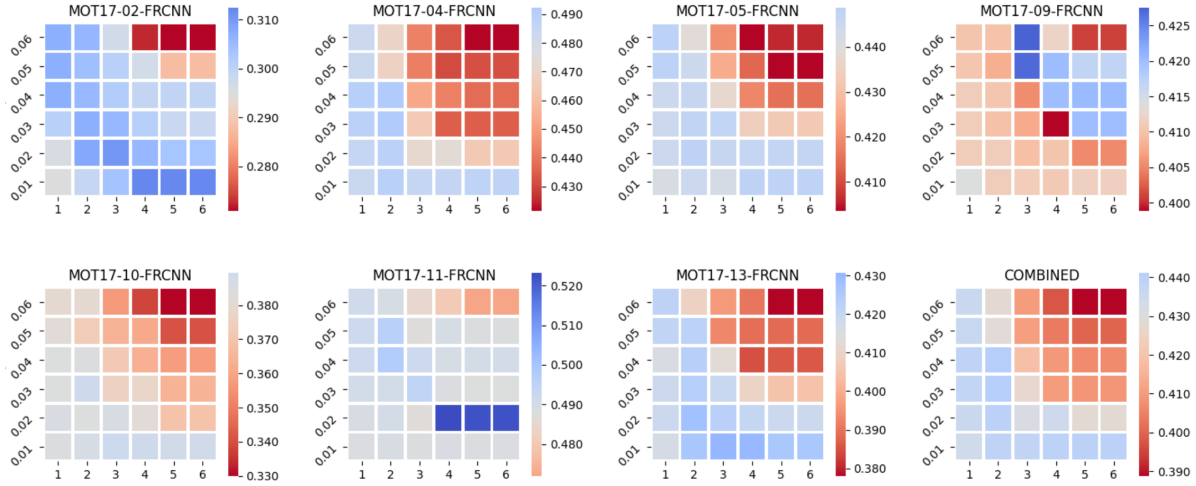


Figure 4.6 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *PredictedCenterDistance* utilisée en conjonction avec *TimeDistance*

rants en moyenne) par rapport au traqueur original. On voit se dégager des lignes de scores très proches sur les diagonales allant du coin inférieur-gauche au coin supérieur-droit des cartes de chaleur. On note donc déjà que les meilleurs résultats sont obtenus (à l'exception des vidéos MOT17-09 et MOT17-11, qui semblent réagir différemment des autres pour cette contrainte, peut-être à cause d'une caractéristique spécifique) pour πou_{fin} compris entre 2 et 5 et $\piou_{50} \leq 0.85$ et sur la diagonale allant de (2, 0.65) à (5, 0.85). Des résultats améliorants sont toutefois disponibles sur plus des deux tiers de la surface des cartes de chaleur pour cinq vidéos sur sept. En étudiant les résultats des variations des paramètres de cette contrainte en conjonction avec les versions par défaut de *PredictedCenterDistance* et *TimeDistance*, nos impressions sont renforcées, comme on peut le voir dans les figures 4.8 et 4.9.

En effet, la diagonale décrite plus haut est encore visible quoiqu'elle s'élargisse dans la figure 4.8 pour de nombreuses vidéos tant les résultats deviennent améliorants de façon similaire sur une large plage. Cette contrainte est celle qui permet les améliorations de scores les plus notables par rapport au traqueur initial, surtout lorsqu'elle est associée à *PredictedCenterDistance*. On préférera des paramètres tels que $\piou_{fin} \in [2, 5]$ et $\piou_{50} \leq 0.85$ ce qui est une plage assez large mais cette contrainte laisse une assez grande liberté de choix de valeurs. Le modèle n'y est donc pas si sensible.

Sensibilité aux paramètres de *TimeDistance* Pour cette contrainte, les résultats visibles dans la figure 4.10 obtenus lorsqu'elle est employée seule ne sont que très peu informatifs (comme pour *AngleDifference* et *SpeedNormDifference*) dans la mesure où ces contraintes ne

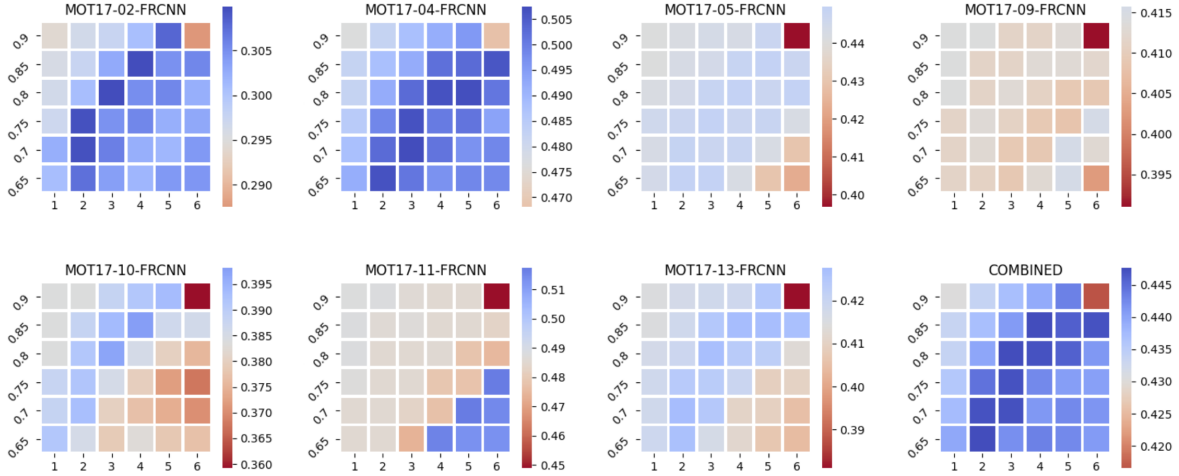


Figure 4.7 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *PredictedIOU* utilisée seule

restreignent pas assez l'espace de recherche : savoir que deux tracklets ne sont pas très espacés dans le temps n'est pas un critère suffisant pour les associer. En revanche, on a des résultats un peu moins mauvais (non améliorants) pour :

- td_{50} faible : si deux tracklets sont extrêmement proches dans le temps l'une de l'autre, il est plus probable qu'elles se suivent directement que si elles sont très éloignées l'une de l'autre. L'association sera juste par moments.
- td_{fin} faible : si on rapproche le score qui déclenche l'arrêt d'une trajectoire de ceux obtenus par l'ensemble des successeurs, la quantité d'association va diminuer et la dégradation des résultats aussi. En effet, si on n'effectue aucune association, on obtient les résultats du tracker initial.

Lorsque l'on observe les résultats présentés dans les figures 4.11 et 4.12 représentant les résultats lorsque la contrainte est employée en conjonction avec *PredictedCenterDistance* et *PredictedIOU*, des comportements plus intéressants émergent.

On observe en effet une forte sensibilité à td_{fin} des résultats sur un point précis, la bascule du score quand le paramètre atteint la valeur de 4 :

- en conjonction avec *PredictedCenterDistance*, les résultats sont améliorants de façon assez homogène pour le facteur $td_{fin} \leq 3$, et deviennent aggravants dès que le paramètre atteint 4.
- en conjonction avec *PredictedIOU*, les résultats sont améliorants et les scores augmentent avec le facteur td_{fin} de 1 à 4 et deviennent brusquement mauvais à 5.

Ainsi, on notera qu'on a une faible sensibilité aux paramètres de cette contrainte à l'exception

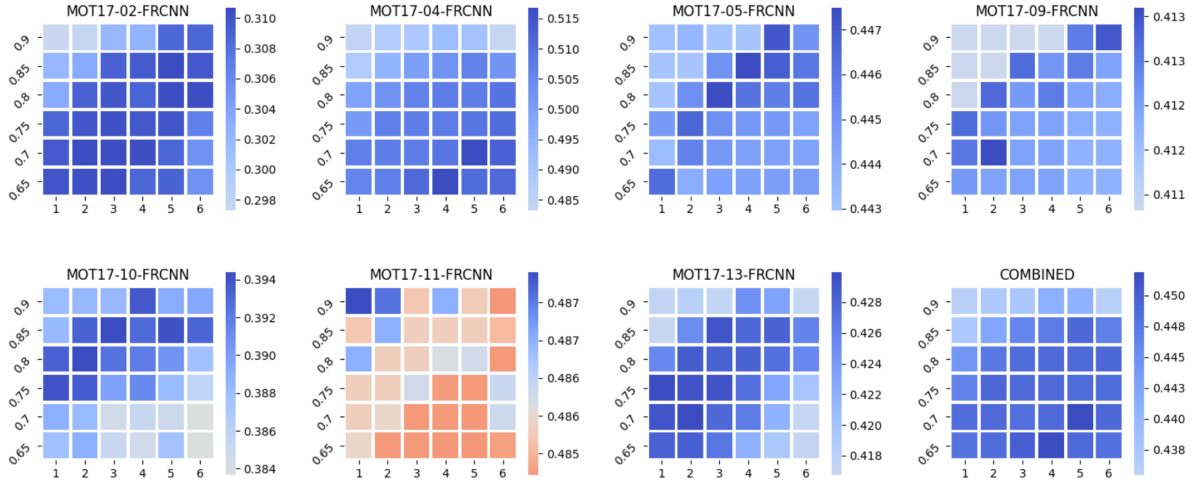


Figure 4.8 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *PredictedIOU* utilisée en conjonction avec *PredictedCenterDistance*

du fait qu'il faut maintenir absolument le facteur td_{fin} sous la valeur 4. Cette valeur semble en effet être une charnière au-delà de laquelle on n'arrive plus à arrêter d'allonger les trajectoires.

Sensibilité aux paramètres de *AngleDifference* La figure 4.13 représente les résultats de cette contrainte lorsqu'elle est employée seule et, comme dit précédemment, on n'obtient aucun résultat améliorant donc on ne s'y étendra pas.

Lorsque cette contrainte est employée en conjonction d'une part avec *TimeDistance* et *PredictedCenterDistance* (visible dans la figure 4.14) et d'autre part avec *TimeDistance* et *PredictedIOU* (visible dans la figure 4.15), on peut tirer quelques conclusions :

- Les vidéos MOT17-09, MOT17-10 et MOT17-11 ne semblent pas bien répondre à l'emploi de cette contrainte, dans la mesure où on n'obtient des résultats que très peu améliorants (quand on en obtient)
- Quelle que soit la configuration des autres contraintes et la vidéo étudiée, le pire résultat est obtenu pour $ad_{50} = 0.125$ et $ad_{fin} = 6$
- En conjonction avec *PredictedCenterDistance*, les résultats s'améliorent lorsque ad_{50} augmente, sauf pour MOT17-09.
- En conjonction avec *PredictedIOU*, les résultats s'améliorent lorsque ad_{50} diminue et que ad_{fin} augmente sauf pour MOT17-09 et MOT17-10.

Ainsi, il est difficile de conclure pour cette contrainte ou de guider la calibration, ce qui est la raison pour laquelle nous l'avons laissé de côté pour cette étude. En effet on obtient par endroits des résultats améliorants mais plusieurs vidéos sont mal gérées. Il semble toutefois

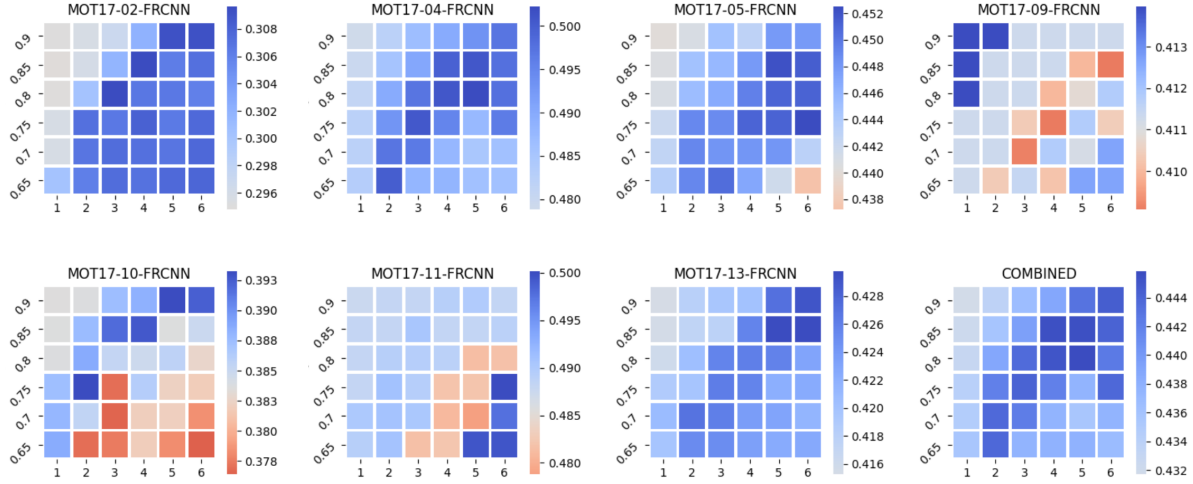


Figure 4.9 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *PredictedIOU* utilisée en conjonction avec *TimeDistance*

qu'il y ait des interactions particulières entre cette contrainte et *PredictedCenterDistance* et *PredictedIOU* qu'il serait intéressant d'explorer à l'avenir dans la mesure où les variations de ses paramètres ont des effets différents selon la contrainte à laquelle on l'associe.

Sensibilité aux paramètres de *SpeedNormDifference* Comme pour les deux contraintes précédentes, on ne trouve pas de résultats améliorants pour celle-ci lorsqu'elle est employée seule, comme on peut le voir dans la figure 4.16.

Lorsqu'elle est employée en conjonction d'une part avec *TimeDistance* et *PredictedCenterDistance* (visible dans la figure 4.17) et d'autre part avec *TimeDistance* et *PredictedIOU* (visible dans la figure 4.18), on note toutefois une forte sensibilité des résultats au paramètre sd_{fin} . En effet on observe dans les deux cas une chute très importante du score *HOTA* lorsque ce paramètre dépasse la valeur 4 (on passe de résultats améliorants pour toutes les vidéos à des résultats très mauvais). On peut noter toutefois une différence entre les deux tests :

- dans la figure 4.17, en conjonction avec *PredictedCenterDistance*, on observe généralement que les scores **baissent** quand on augmente sd_{fin} , et connaissent une chute marquée au-delà de 4.
- dans la figure 4.18, en conjonction avec *PredictedIOU*, on observe que les scores **augmentent** quand on augmente sd_{fin} jusqu'à 4 puis chutent.

Ainsi on retiendra que si l'on souhaite employer cette contrainte, sd_{fin} doit être inférieure ou égale à 4 mais qu'il conviendra d'étudier les interactions avec les deux contraintes *PredictedCenterDistance* et *PredictedIOU*.

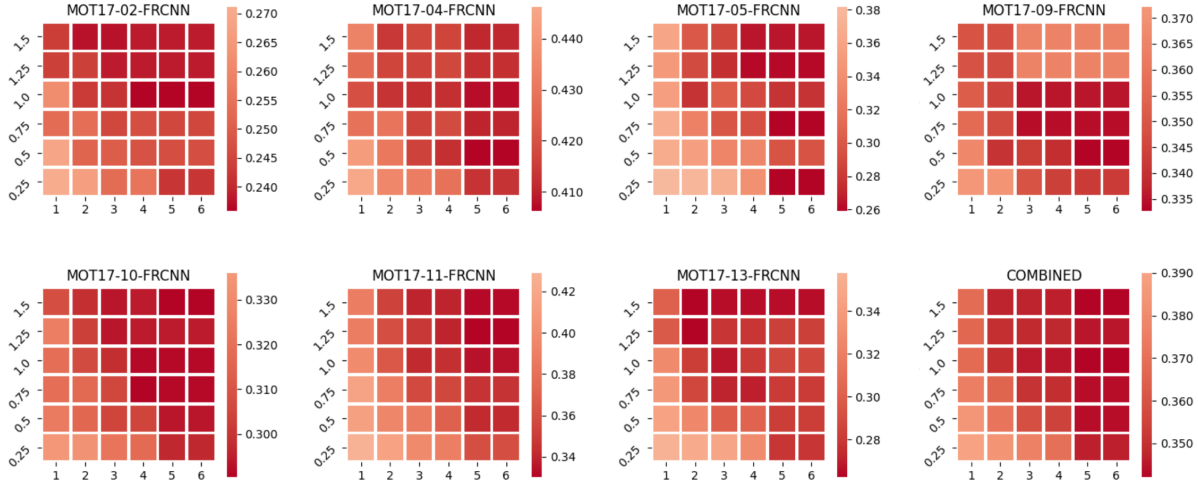


Figure 4.10 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *TimeDistance* utilisée seule

4.8.3 Discussion sur la sensibilité du *TrackletCutter*

Pour le *TrackletCutter*, tel que décrit dans la section 3.1.2, nous avons seulement un paramètre à optimiser : T_{TC} , l'IOU des détections appartenant à la même trame à partir duquel on coupe les tracklets qui les contiennent car on suppose qu'une occlusion est probable. On peut s'attendre au comportement suivant pour ce modèle :

- en dessous d'un certain seuil, le *TrackletCutter* risque d'effectuer de très nombreuses coupes, dont certaines à des endroits qui ne posaient pas problème et notre modèle d'association risque de ne pas les associer toutes ensemble. Ainsi en-dessous d'une certaine valeur de T_{TC} , nous risquons d'obtenir de moins bons résultats que sans rien couper.
- au-dessus d'un certain seuil, le *TrackletCutter* risque de n'effectuer plus aucune coupe ou presque et donc de ne rien changer aux résultats.

Ainsi on s'attend à obtenir en moyenne des résultats moins bons pour les valeurs faibles de T_{TC} et des résultats proches puis égaux à ceux sans découpage pour les valeurs se rapprochant de 1. Et entre les deux, on espérait obtenir un pic plus performant qu'en l'absence de découpage. La présence d'un tel pic signifierait alors que d'une part le découpage s'est fait dans des zones pertinentes **et** que le modèle d'association a associé correctement les tracklets ainsi découpés.

C'est effectivement ce que l'on observe dans la figure 4.19. Celle-ci représente le *HOTA* obtenu lorsque l'on applique les modèles d'association CP qui permettaient d'obtenir les meilleurs scores (trouvés dans la seconde phase de calibration décrite dans la partie 4.7.2) aux trajectoires obtenues pour différentes valeurs de seuil du *TrackletCutter*. La ligne noire

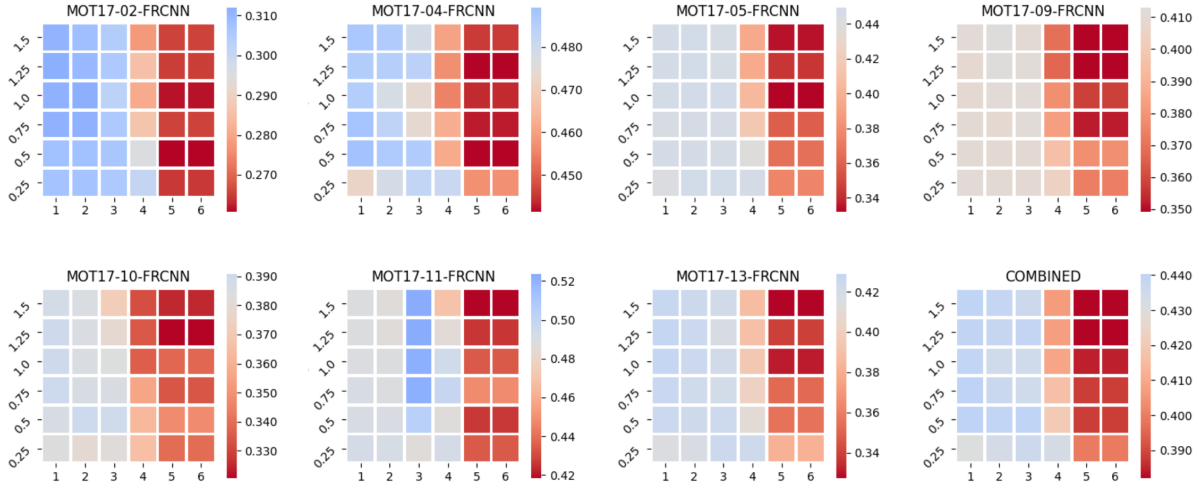


Figure 4.11 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *TimeDistance* utilisée en conjonction avec *PredictedCenterDistance*

représente le score *HOTA* moyen, la zone grise qui l'encadre représente les valeurs comprises dans la moyenne à plus ou moins un écart type et les pointillés rouges et verts représentent les scores maximaux et minimaux obtenus. Les résultats obtenus pour un seuil $T_{TC} = 1$ sont donc ceux qu'on obtient en l'absence du *TrackletCutter* (car il n'arrive jamais qu'on obtienne un IOU de 1 pour deux détections de la même trame). On observe donc bien comme attendu des résultats plus faibles que ceux-ci pour les valeurs faibles de T_{TC} et on peut conclure que jusqu'à $T_{TC} = 0.4$, le modèle d'association n'arrive pas à réassocier efficacement les trop nombreux découpages qu'il fait. En revanche, on observe que pour les valeurs de ce seuil comprises entre 0.5 et 0.6, les scores *HOTA* renvoyés sont meilleurs en moyenne que sans découpage.

Les meilleurs résultats obtenus sont visibles dans le tableau 4.6 :

Tableau 4.6 Scores obtenus pour la meilleure configuration du *TrackletCutter*

TrackletCutter	Scores		
T_{TC}	HOTA	MOTA	IDF1
0.5	45.47%	49.90%	54.11%

Notre *TrackletCutter* est donc utilisé à **un seuil de 0.5** dans les tests que nous avons faits, toutefois on a vu que les améliorations obtenues sont minimales et ne s'appliquent qu'à *IOU-Tracker* : on a un gain de 0.14 points de *HOTA* sur les 3.14 points qu'apporte le modèle complet.

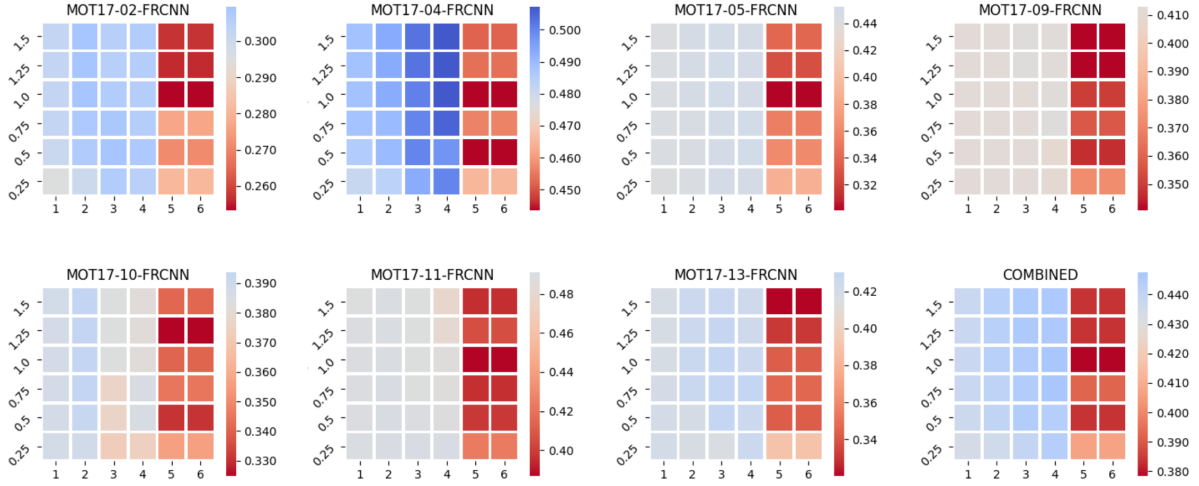


Figure 4.12 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *TimeDistance* utilisée en conjonction avec *PredictedIOU*

4.8.4 Discussion sur la sensibilité du modèle d'interpolation

Dans le test qui suit, nous avons repris les configurations du modèle (avec et sans *TrackletCutter*) qui permettaient d'obtenir les meilleurs scores et y avons appliqué différentes valeurs pour le paramètre décrit dans la section 3.4 : *maxGapSize*, qui représente la taille maximale (en nombre de trames successives) des trous présents dans des trajectoires que l'on souhaite combler par interpolation linéaire. Nos attentes ici étaient les suivantes :

- Dans le cas de bonnes associations, toute taille de trou interpolé devrait permettre d'améliorer les scores, mais de moins en moins à mesure que la taille des trous augmente. En effet, pour des trous qui s'étendent sur plusieurs dizaines de trames, la probabilité que la trajectoire réelle de l'objet ait réellement consisté en un espacement régulier entre les deux extrémités du trou diminue.
- Dans le cas d'associations erronées, plus la taille des trous augmente et plus les résultats risquent d'en pâtir dans la mesure où l'on ajoute des détections qui relient deux objets différents, et donc la probabilité que ces détections correspondent bien à un objet d'intérêt et que celui-ci appartiennent à l'une des deux trajectoires reliés (et donc n'augmente pas le nombre d'IDSW) diminue.
- Plus la taille des trous augmente et plus la probabilité que l'association soit erronée augmente aussi.
- A partir d'une certaine taille de trous, on n'observera plus d'interpolation : tous les trous des trajectoires seront remplis.

Finalement, si une amélioration de la performance intervient (ce que l'on sait maintenant),

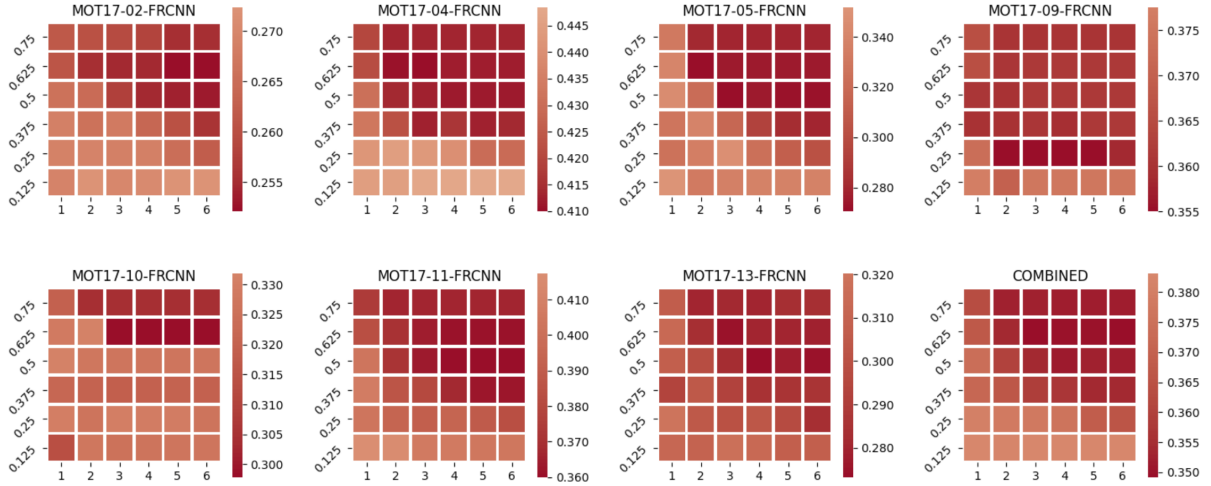


Figure 4.13 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *AngleDifference* utilisée seule

on peut donc s'attendre à ce que jusqu'à un certain seuil pour *maxGapSize*, l'interpolation améliore les résultats en moyenne en comblant les trous présents sur peu de trames. Au-delà de ce seuil, on peut s'attendre à ce que la performance baisse dans la mesure où on va amplifier l'effet d'erreurs d'association. Puis le score devrait se stabiliser après avoir comblé l'ensemble des trous.

C'est effectivement ce que l'on observe en moyenne dans la figure 4.20 qui représente les scores *HOTA* des 20 meilleures configurations (10 avec le TrackletCutter et 10 sans) pour différentes tailles maximales de trous interpolés. Comme attendu on obtient une amélioration, forte jusqu'aux trous de 42 (on passe d'un *HOTA* de 45.2% en moyenne à un *HOTA* de 45.8% avec un pic à 46%) puis cette amélioration ralentit et on observe un faible déclin à partir des trous de 90 trames (qui représentent le score maximal obtenu). En revanche si on observe vidéo par vidéo, comme le représente la figure 4.21, on se rend compte qu'il y a des vidéos pour lesquelles toute interpolation empire les résultats (comme c'est le cas pour la vidéo MOT17-10-FRCNN et qu'en s'arrêtant à des trous de taille 40 (42 précisément), on arrive à éviter les plus grosses pertes tout en ayant obtenu la majorité des améliorations dans les autres vidéos. On a donc fixé notre modèle final *maxGapSize* à **42 trames**. Ce choix semble plus sécuritaire dans le cadre où plus de vidéos réagiraient comme MOT17-05, MOT17-10 et MOT17-13. Si on arrive à l'avenir à extraire la caractéristique de ces vidéos qui les pousse à mal réagir à notre modèle d'interpolation peut-être pourra-t-on ajouter un paramètre supplémentaire qui permet de décider s'il est intéressant de l'utiliser et de la calibrer pour une valeur conservatrice comme ici ou à un seuil plus élevé.

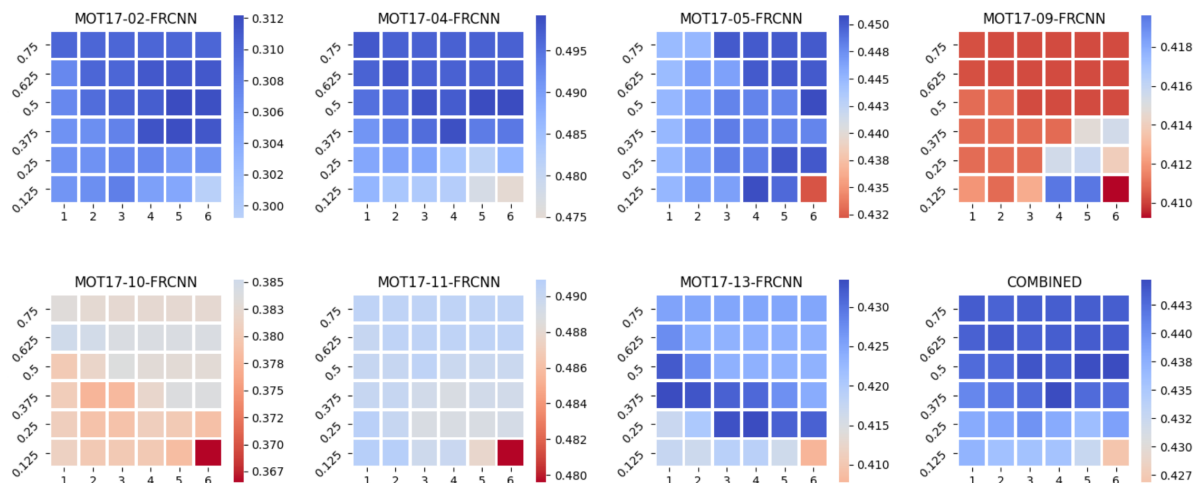


Figure 4.14 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *AngleDifference* utilisée en conjonction avec *TimeDistance* et *PredictedCenterDistance*

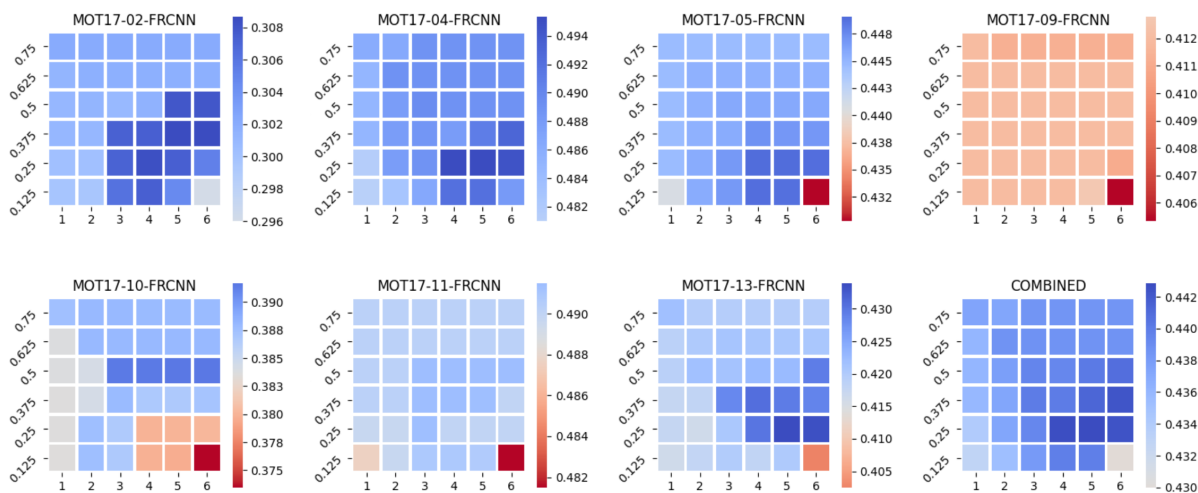


Figure 4.15 Cartes de chaleur des résultats *HOTA* de l'ensemble des vidéos de MOT17 pour *AngleDifference* utilisée en conjonction avec *TimeDistance* et *PredictedIOU*

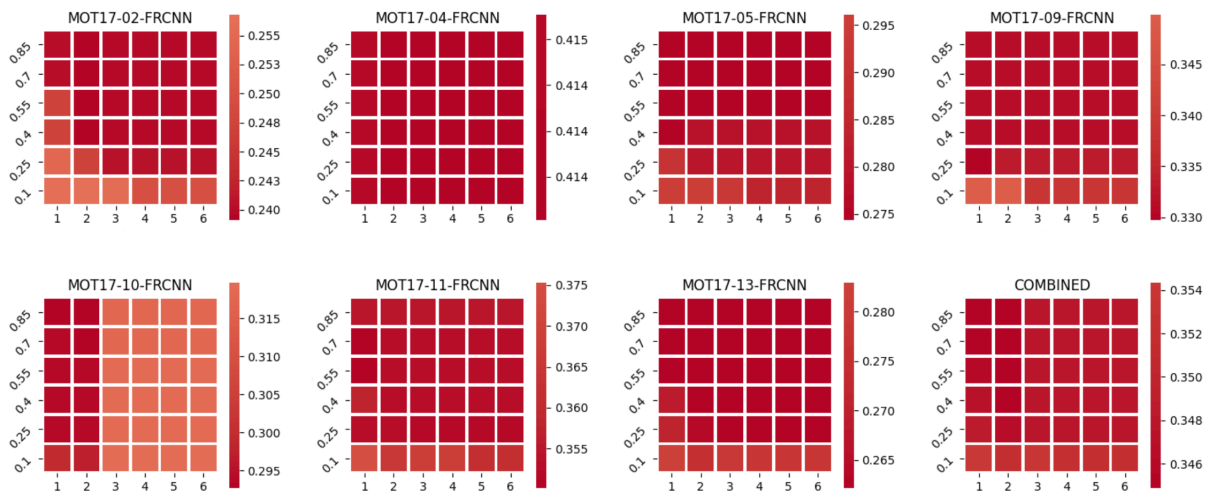


Figure 4.16 Cartes de chaleur des résultats $HOTA$ de l'ensemble des vidéos de MOT17 pour $SpeedNormDifference$ utilisée seule

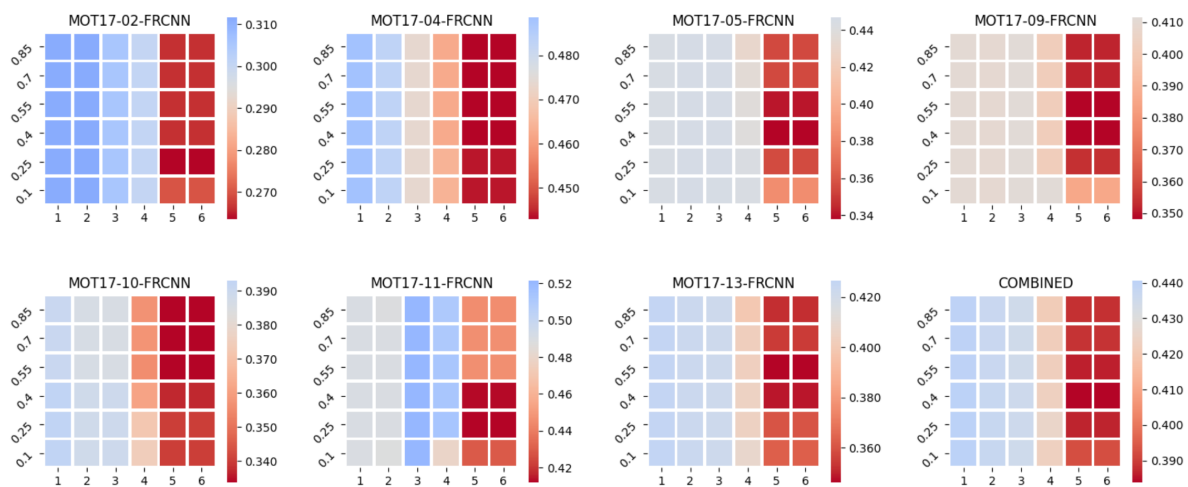


Figure 4.17 Cartes de chaleur des résultats $HOTA$ de l'ensemble des vidéos de MOT17 pour $SpeedNormDifference$ utilisée en conjonction avec $TimeDistance$ et $PredictedCenterDistance$

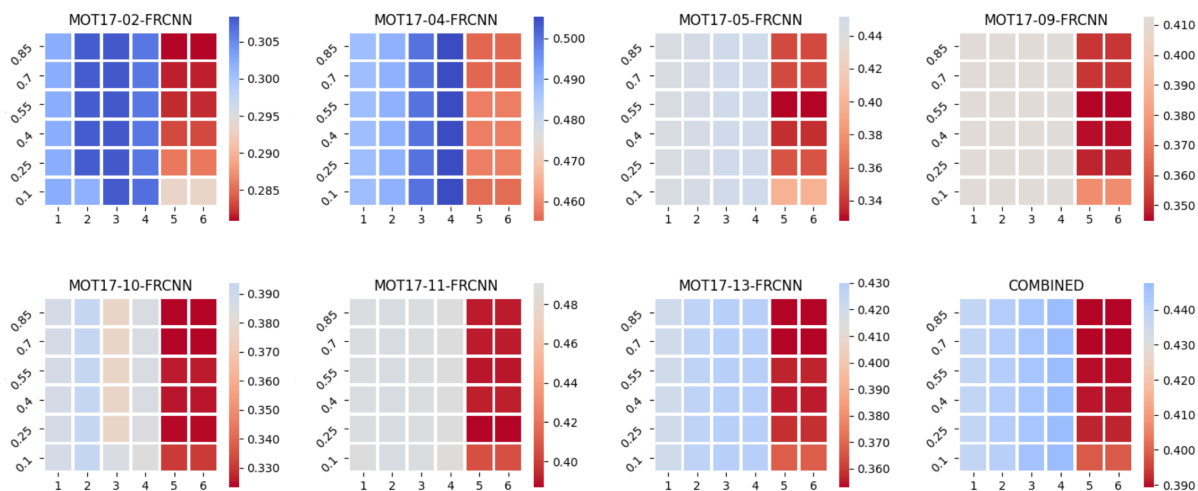


Figure 4.18 Cartes de chaleur des résultats $HOTA$ de l'ensemble des vidéos de MOT17 pour $SpeedNormDifference$ utilisée en conjonction avec $TimeDistance$ et $PredictedIOU$

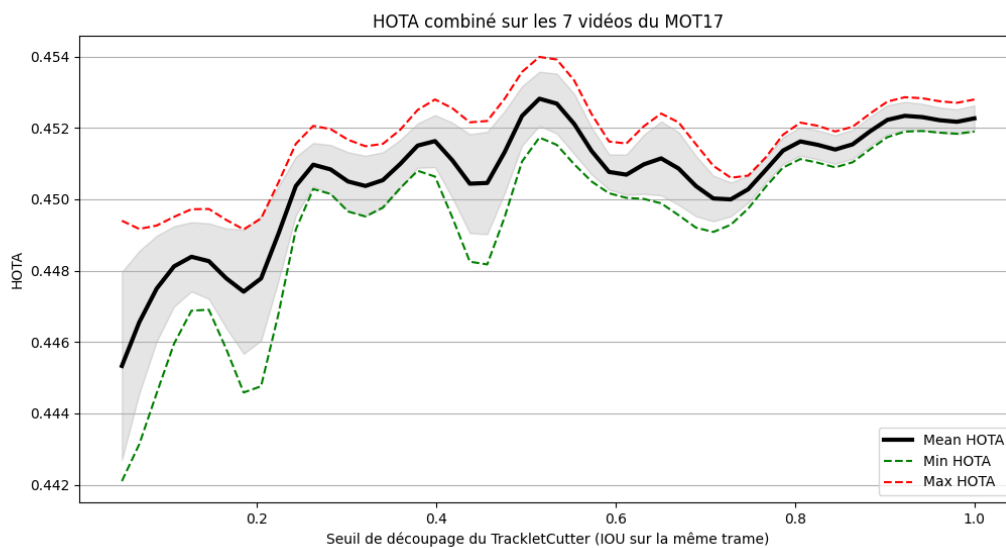


Figure 4.19 Score $HOTA$ sur l'ensemble des vidéos du MOT17 en faisant varier les valeurs du seuil du TrackletCutter

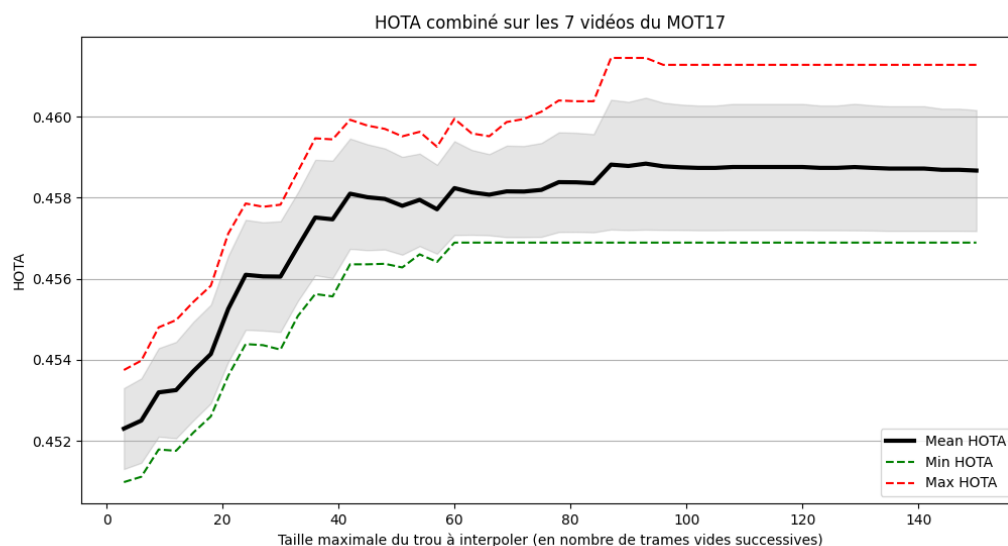


Figure 4.20 Score $HOTA$ sur l'ensemble des vidéos du MOT17 en faisant varier la taille maximale des trous comblés par l'interpolation

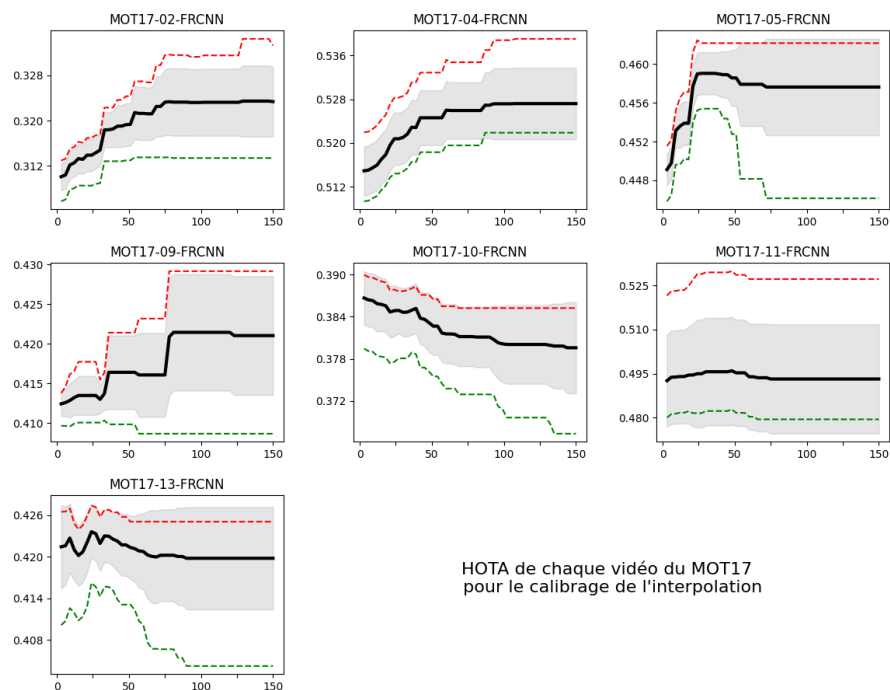


Figure 4.21 Score $HOTA$ de chacune des vidéos du MOT17 en faisant varier la taille maximale des trous comblés par l'interpolation

CHAPITRE 5 CONCLUSION

Ce travail de recherche présente un module que l'on peut greffer à tout type de traqueur de l'état de l'art du suivi multi-objet pour en améliorer la performance d'association comme nous avons pu le montrer en le testant sur les traqueurs *IOU-Tracker*, *SORT* et *Tracktor* sur la base de données MOT17 du MOT-Challenge, ce qui démontre sa compétitivité dans le domaine du suivi de piétons.

5.1 Synthèse des travaux

Nous faisons ci-dessous un résumé des apports de ce travail :

1. Nous proposons un module complet qui améliore les résultats de plusieurs traqueurs de l'état de l'art pour le suivi de piétons.
2. Nous proposons un modèle de programmation par contraintes avec propagation de croyances pour l'association de tracklets, ce qui est une première. Ce modèle se base sur la position et la dynamique des tracklets pour construire des scores d'association. La construction de ces marginales nous permet de nous appuyer sur une heuristique de choix de variables et de valeurs particulièrement efficace, *maxMarginal*, qui nous permet d'explorer efficacement nos solutions.
3. Nous proposons un modèle simple de découpage de tracklets, basé sur la détection de croisements de tracklets et qui permet d'améliorer les scores HOTA et IDF1 pour *IOU-Tracker*.
4. Nous proposons un modèle d'interpolation qui permet de combler les nombreux trous présents dans les trajectoires reconstruites par notre modèle d'association.

5.2 Limitations de la solution proposée

TrackletCutter Dans notre modèle *TrackletCutter*, le découpage des tracklets se fait à chaque fois que l'on observe deux trajectoires qui se coupent, ce qui nous laisse vulnérables à deux types d'erreur :

- le cas, peu grave mais inutile, dans lequel on coupe une trajectoire pourtant correspondant à la réalité-terrain à cet endroit. On s'expose alors à ne pas réussir à la reconstruire correctement.
- le cas, plus gênant, selon lequel on conserve une portion de la trajectoire construite par le tracker alors qu'elle suit différents objets.

La version actuelle fait de ce module le moins efficace des trois que nous proposons (découpage, association et interpolation) en temps de calcul et en performance : Il n’améliore le HOTA que pour *IOU-Tracker* et de très peu (5% de l’amélioration totale que l’on apporte avec notre modèle complet). Il gagnerait donc à être remplacé par un autre plus performant à l’avenir.

Calibration sous optimale Il a été assez fastidieux de calibrer l’ensemble des paramètres du modèle et cela s’est fait sans garantie d’avoir atteint une configuration optimale. En effet ne serait-ce que sur un jeu de données (ici MOT17) et pour un tracker de paramètres fixés (*IOU-Tracker*), trouver une combinaison optimale des paramètres du *TrackletCutter*, de l’interpolation et surtout du modèle d’association CP (jusqu’à 3 paramètres par contrainte sur 5 contraintes) est assez coûteux en temps, sachant que les meilleures valeurs de paramètres pour chaque contrainte testée individuellement n’ont aucune garantie d’être celles pour les contraintes associées ! Cela nous a poussé notamment à abandonner l’emploi des contraintes *AngleDifference* et *SpeedNormDifference* lors de ce travail pour réduire l’espace de recherche et optimiser les autres qui semblaient plus prometteuses, ce qui nous a néanmoins même permis d’atteindre des scores intéressants sur les deux autres traqueurs testés : *SORT* et *Tracktor*.

Variabilité inter-vidéos On a observé lors des tests de sensibilité et de la calibration du modèle une variabilité parfois marquée dans les résultats entre les vidéos avec parfois une ou deux vidéos qui réagissaient différemment des autres à la variation de tel ou tel paramètre. Nous avons essayé d’adapter un peu les valeurs de nos paramètres aux caractéristiques fournies par le défi pour chaque séquence vidéo (hauteur, largeur, FPS par exemple) mais une adaptation plus forte aux vidéos pourrait être intéressante.

Absence de mise à jour de l’affectation à une trajectoire Notre modèle, par construction, gère l’ajout d’un successeur à une trajectoire comme l’association du successeur à la dernière tracklet de la trajectoire. Or la dernière tracklet d’une trajectoire peut être une mauvaise association ou être particulièrement courte et donc mal représenter la trajectoire en question. D’autres modèles proposent de conserver une mémoire des caractéristiques de la trajectoire ou de modifier les critères d’affectation à une trajectoire à mesure qu’elle s’allonge. Cela pourrait être bénéfique à notre modèle.

5.3 Améliorations futures

Ajout d'un modèle d'apparence Notre objectif ici était de construire un premier modèle d'association de tracklets pour le suivi multi-objet et nous nous en sommes tenus à des modèles de mouvement pour guider l'association des tracklets. Toutefois on sait que les modèles d'apparence sont très souvent intégrés dans l'état de l'art et permettent souvent des gains de performance. Il pourrait donc être intéressant de chercher à en intégrer un, et d'ajouter une contrainte qui base son score sur une mesure de la distance entre une description d'apparence du début d'un successeur et la fin de la trajectoire à laquelle on cherche à l'associer.

Améliorations du modèle de découpage de tracklets Il serait possible de repérer de façon plus fine les zones les plus probablement erronées des tracklets et couper à ces endroits précisément. L'objectif serait de repérer des discontinuités dans la dynamique de la trajectoire par exemple (si la trajectoire semble changer brusquement de direction ou de vitesse) ou dans un modèle d'apparence pour couper en ces endroits. On pourrait par exemple calculer une fonction d'énergie similaire à celle proposée par [31] pour couper en les zones qui font augmenter cette fonction. Concernant la vitesse de calcul de cet algorithme, on pourrait essayer aussi de fournir moins de détection en entrée en supprimant plus de détections au préalable (notamment celles auxquelles le détecteur donne un score faible, ce qui est fait par de nombreux traqueurs). Si notre modèle d'interpolation est par la suite suffisamment performant, cela permettra de rattraper cette perte, tant en améliorant la performance du *TrackletCutter* que celle de l'association.

Améliorations du modèle d'interpolation Même si les performances de notre modèle d'interpolation sont allées au-delà de nos attentes (jusqu'à 30% des gains en HOTA fournis par notre module), il est simpliste et gagnerait à être amélioré. On pourrait par exemple se baser sur la vitesse de fin avant le trou et chercher à l'interpoler de sorte à ce qu'il n'y ait pas de discontinuité à la jonction après le trou. D'autre part, des méthodes à base de *Single Object Tracker* existent déjà à cet effet, notamment utilisées pour améliorer *IOU-Tracker* dans [21] et pourraient être implémentées ici pour remplir cette fonction.

Améliorer la calibration du modèle Pour aider à la calibration des différentes valeurs du modèle, plusieurs solutions sont envisageables et peuvent être employées en parallèle. On pourrait d'une part chercher à poser cette calibration comme un problème d'optimisation dont l'objectif serait de maximiser le HOTA d'un ensemble varié de vidéos pour plusieurs

traqueurs de l'état de l'art. Il pourrait aussi être intéressant de tester les interactions entre les trois modules et entre les contraintes elles-mêmes plus finement, là où par manque de temps nous avons optimisé nos modules séparément. On pourrait par ailleurs récupérer des statistiques sur les caractéristiques qui nous intéressent dans la réalité-terrain de divers défis de suivi multi-objet pour avoir une idée plus claire de leurs valeurs et pouvoir répondre à des questions telles que "Quelle variations d'angle ou de vitesse peut-on attendre en moyenne ou au maximum dans une trajectoire réelle ?". De plus, il pourrait être utile de construire un modèle qui s'appuie encore plus sur les caractéristiques de chaque vidéo. Nous avons appuyé nos contraintes sur les informations fournies par le défi mais d'autres informations sont disponibles dans les vidéos, telles que la taille des boîtes englobantes qu'elle contient. En effet, nous nous sommes appuyés sur la taille de l'image pour nos seuils de vitesse et de distance mais il pourrait être plus intéressant de s'appuyer en parallèle sur la taille des détections. On pourrait en apprendre sur le type de point de vue de la vidéo : Si les boîtes englobantes sont toutes assez petites, on peut conclure à un point de vue aérien tandis que s'il y a une grande variabilité parmi leurs tailles, on peut deviner que des individus sont à des distances différentes de la caméra et peuvent s'en approcher ou s'en éloigner.

Travailler sur la mise à jour des informations des trajectoires Un des avantages de travailler sur les caractéristiques des tracklets plutôt que sur celles des détections est de permettre de gagner en richesse sur certains points (notamment grâce à l'apparition de la notion de vitesse) mais surtout en robustesse de ces caractéristiques dans la mesure où elles ont le temps d'être éprouvées et sont moins sensibles au bruit du détecteur. Ainsi une vitesse calculée entre deux détections successives est en moyenne moins représentative de la réalité que celle que l'on peut obtenir en moyennant sur les dix dernières détections de la trajectoire. De même, on pourrait envisager un maintien de caractéristiques de la trajectoire (et non plus seulement des tracklets individuelles) qui gagnerait en crédit à mesure que la trajectoire s'allonge et potentiellement en fonction de la qualité de ses détections et tracklets. Ainsi concernant l'apparence par exemple, on pourrait conserver des caractéristiques propres non pas à la dernière tracklet ajoutée mais à l'ensemble de la trajectoire, ce qui permettrait de mieux gérer des changements de pose ou de couleur s'ils sont déjà intervenus par le passé.

Notre force : la modularité La majorité des améliorations décrites plus haut seront d'autant plus faciles à implémenter dans notre méthode qu'elle est particulièrement modulaire : les parties de découpage de tracklets, d'association et d'interpolation sont complètement indépendantes et donc modifiables indépendamment. À l'intérieur même du modèle d'association, on travaille avec des contraintes complètement séparées et donc la modification de

l'une d'entre elles ou l'ajout d'une autre sont très facilement envisageables.

RÉFÉRENCES

- [1] G. Pesant, “From Support Propagation to Belief Propagation in Constraint Programming (Extended Abstract),” dans *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Yokohama, Japan : International Joint Conferences on Artificial Intelligence Organization, juill. 2020, p. 5100–5104. [En ligne]. Disponible : <https://www.ijcai.org/proceedings/2020/715>
- [2] E. Bochinski, V. Eiselein et T. Sikora, “High-Speed Tracking-by-Detection Without Using Image Information [Challenge winner IWOT4S],” août 2017.
- [3] A. Bewley, Z. Ge, L. Ott, F. Ramos et B. Upcroft, “Simple Online and Realtime Tracking,” *2016 IEEE International Conference on Image Processing (ICIP)*, p. 3464–3468, sept. 2016, arXiv : 1602.00763. [En ligne]. Disponible : <http://arxiv.org/abs/1602.00763>
- [4] P. Bergmann, T. Meinhardt et L. Leal-Taixe, “Tracking without bells and whistles,” *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, p. 941–951, oct. 2019, arXiv : 1903.05625. [En ligne]. Disponible : <http://arxiv.org/abs/1903.05625>
- [5] A. Milan, L. Leal-Taixe, I. Reid, S. Roth et K. Schindler, “Mot16 : A benchmark for multi-object tracking,” 2016.
- [6] L. Leal-Taixé, A. Milan, I. Reid, S. Roth et K. Schindler, “MOTChallenge 2015 : Towards a Benchmark for Multi-Target Tracking,” *arXiv :1504.01942 [cs]*, avr. 2015, arXiv : 1504.01942. [En ligne]. Disponible : <http://arxiv.org/abs/1504.01942>
- [7] P. Dendorfer, A. Ošep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth et L. Leal-Taixé, “MOTChallenge : A Benchmark for Single-Camera Multiple Target Tracking,” *arXiv :2010.07548 [cs]*, déc. 2020, arXiv : 2010.07548. [En ligne]. Disponible : <http://arxiv.org/abs/2010.07548>
- [8] R. Girshick, J. Donahue, T. Darrell et J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” 2014.
- [9] K. He, G. Gkioxari, P. Dollár et R. Girshick, “Mask r-cnn,” 2018.
- [10] S. Ren, K. He, R. Girshick et J. Sun, “Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv :1506.01497 [cs]*, janv. 2016, arXiv : 1506.01497. [En ligne]. Disponible : <http://arxiv.org/abs/1506.01497>
- [11] X. Zhou, V. Koltun et P. Krähenbühl, “Tracking Objects as Points,” *arXiv :2004.01177 [cs]*, août 2020, arXiv : 2004.01177. [En ligne]. Disponible : <http://arxiv.org/abs/2004.01177>

- [12] P. Tokmakov, J. Li, W. Burgard et A. Gaidon, “Learning to Track with Object Permanence,” *arXiv :2103.14258 [cs]*, mars 2021, arXiv : 2103.14258. [En ligne]. Disponible : <http://arxiv.org/abs/2103.14258>
- [13] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, n°. 1, p. 35–45, 03 1960. [En ligne]. Disponible : <https://doi.org/10.1115/1.3662552>
- [14] siyuanc2, “siyuanc2/kiout,” juin 2021, original-date : 2018-10-13T21 :50 :23Z. [En ligne]. Disponible : <https://github.com/siyuanc2/kiout>
- [15] C. Dicle, O. I. Camps et M. Sznaiier, “The Way They Move : Tracking Multiple Targets with Similar Appearance,” dans *2013 IEEE International Conference on Computer Vision*. Sydney, Australia : IEEE, déc. 2013, p. 2304–2311. [En ligne]. Disponible : <http://ieeexplore.ieee.org/document/6751397/>
- [16] G. D. Evangelidis et E. Z. Psarakis, “Parametric image alignment using enhanced correlation coefficient maximization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, n°. 10, p. 1858–1865, 2008.
- [17] D. Stadler et J. Beyerer, “Improving Multiple Pedestrian Tracking by Track Management and Occlusion Handling,” 2021, p. 10 958–10 967. [En ligne]. Disponible : https://openaccess.thecvf.com/content/CVPR2021/html/Stadler_Improving_Multiple_Pedestrian_Tracking_by_Track_Management_and_Occlusion_Handling_CVPR_2021_paper.html
- [18] N. Dalal et B. Triggs, “Histograms of oriented gradients for human detection,” *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, p. 886–893, 2005. [En ligne]. Disponible : http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1467360
- [19] C. Novak et S. Shafer, “Anatomy of a color histogram,” dans *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1992, p. 599–605.
- [20] J. F. Henriques, R. Caseiro, P. Martins et J. Batista, “High-speed tracking with kernelized correlation filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, n°. 3, p. 583–596, Mar 2015. [En ligne]. Disponible : <http://dx.doi.org/10.1109/TPAMI.2014.2345390>
- [21] E. Bochinski, T. Senst et T. Sikora, “Extending IOU Based Multi-Object Tracking by Visual Information,” dans *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, nov. 2018, p. 1–6.

- [22] P. Dai, R. Weng, W. Choi, C. Zhang, Z. He et W. Ding, “Learning a proposal classifier for multiple object tracking,” 2021.
- [23] J. Yang, H. Ge, J. Yang, Y. Tong et S. Su, “Online multi-object tracking using multi-function integration and tracking simulation training,” *Applied Intelligence*, mai 2021. [En ligne]. Disponible : <https://doi.org/10.1007/s10489-021-02457-5>
- [24] S. Sun, N. Akhtar, H. Song, A. Mian et M. Shah, “Deep Affinity Network for Multiple Object Tracking,” *arXiv :1810.11780 [cs]*, juill. 2019, arXiv : 1810.11780. [En ligne]. Disponible : <http://arxiv.org/abs/1810.11780>
- [25] P. Chu et H. Ling, “FAMNet : Joint Learning of Feature, Affinity and Multi-dimensional Assignment for Online Multiple Object Tracking,” *arXiv :1904.04989 [cs]*, avr. 2019, arXiv : 1904.04989. [En ligne]. Disponible : <http://arxiv.org/abs/1904.04989>
- [26] G. Brasó et L. Leal-Taixé, “Learning a Neural Solver for Multiple Object Tracking,” *arXiv :1912.07515 [cs]*, avr. 2020, arXiv : 1912.07515. [En ligne]. Disponible : <http://arxiv.org/abs/1912.07515>
- [27] I. Papakis, A. Sarkar et A. Karpatne, “Gcnmatch : Graph convolutional neural networks for multi-object tracking via sinkhorn normalization,” 2021.
- [28] N. Wojke, A. Bewley et D. Paulus, “Simple online and realtime tracking with a deep association metric,” 2017.
- [29] Y. Xu, Y. Ban, G. Delorme, C. Gan, D. Rus et X. Alameda-Pineda, “TransCenter : Transformers with Dense Queries for Multiple-Object Tracking,” *arXiv :2103.15145 [cs]*, mars 2021, arXiv : 2103.15145. [En ligne]. Disponible : <http://arxiv.org/abs/2103.15145>
- [30] A. Girbau, X. G. i Nieto, I. Rius et F. Marqués, “Multiple object tracking with mixture density networks for trajectory estimation,” 2021.
- [31] A. Andriyenko et K. Schindler, “Multi-target tracking by continuous energy minimization,” dans *CVPR 2011*. Colorado Springs, CO, USA : IEEE, juin 2011, p. 1265–1272. [En ligne]. Disponible : <http://ieeexplore.ieee.org/document/5995311/>
- [32] A. Milan, S. Roth et K. Schindler, “Continuous Energy Minimization for Multitarget Tracking,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, n°. 1, p. 58–72, janv. 2014. [En ligne]. Disponible : <http://ieeexplore.ieee.org/document/6520846/>
- [33] S. Karthik, A. Prabhu et V. Gandhi, “Simple Unsupervised Multi-Object Tracking,” *arXiv :2006.02609 [cs]*, juin 2020, arXiv : 2006.02609. [En ligne]. Disponible : <http://arxiv.org/abs/2006.02609>
- [34] K. He, X. Zhang, S. Ren et J. Sun, “Deep residual learning for image recognition,” 2015.

- [35] A. Pineault, G.-A. Bilodeau et G. Pesant, “Tracking Road Users using Constraint Programming,” *arXiv :2003.04468 [cs]*, mars 2020, arXiv : 2003.04468. [En ligne]. Disponible : <http://arxiv.org/abs/2003.04468>
- [36] A. Pineault, Alexandre, “Suivi multiobjet en situation urbaine à l’aide de la programmation par contraintes,” Mémoire, Polytechnique Montréal, affiliée à l’Université de Montréal, Montréal, déc. 2019.
- [37] Z. Chen, G. Qiu, H. Zhang, B. Sheng et P. Li, “FIOU Tracker : An Improved Algorithm of IOU Tracker in Video with a Lot of Background Inferences,” dans *Advances in Computer Graphics*, N. Magnenat-Thalmann, C. Stephanidis, E. Wu, D. Thalmann, B. Sheng, J. Kim, G. Papagiannakis et M. Gavrilova, édit. Cham : Springer International Publishing, 2020, p. 145–156.
- [38] H. Pirsiavash, D. Ramanan et C. C. Fowlkes, “Globally-optimal greedy algorithms for tracking a variable number of objects,” dans *CVPR 2011*. Colorado Springs, CO, USA : IEEE, juin 2011, p. 1201–1208. [En ligne]. Disponible : <http://ieeexplore.ieee.org/document/5995604/>
- [39] L. Wen, W. Li, J. Yan, Z. Lei, D. Yi et S. Z. Li, “Multiple Target Tracking Based on Undirected Hierarchical Relation Hypergraph,” dans *2014 IEEE Conference on Computer Vision and Pattern Recognition*, juin 2014, p. 1282–1289, iSSN : 1063-6919.
- [40] G. Wang, Y. Wang, R. Gu, W. Hu et J.-N. Hwang, “Split and connect : A universal tracklet booster for multi-object tracking,” 2021.
- [41] H. W. Kuhn, “The Hungarian Method for the Assignment Problem,” *Naval Research Logistics Quarterly*, vol. 2, n°. 1–2, p. 83–97, March 1955.
- [42] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé et B. Leibe, “HOTA : A Higher Order Metric for Evaluating Multi-object Tracking,” *International Journal of Computer Vision*, vol. 129, n°. 2, p. 548–578, 2021. [En ligne]. Disponible : <https://doi.org/10.1007/s11263-020-01375-2>
- [43] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang et S. Lyu, “UA-DETRAC : A New Benchmark and Protocol for Multi-Object Detection and Tracking,” *arXiv :1511.04136 [cs]*, janv. 2020, arXiv : 1511.04136. [En ligne]. Disponible : <http://arxiv.org/abs/1511.04136>