Analysis and data mining of moving object trajectories

Mohamed Khalil El Mahrsi

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2013-ENST-0056

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et réseaux »

présentée et soutenue publiquement par

Mohamed Khalil EL MAHRSI

le 30 septembre 2013

Analyse et fouille de données de trajectoires d'objets mobiles

Directeur de thèse : **Prof. Fabrice ROSSI**

Jury

M. Talel ABDESSALEM, Professeur, INFRES, Télécom ParisTech	Président du jury
Mme Barbara HAMMER, Professeur, CITEC, Bielefeld University	Rapporteur
Mme Karine ZEITOUNI, Professeur, PRISM, Université de Versailles	Rapporteur
M. Pierre BORGNAT, Chargé de Recherche, École normale supérieure de Lyon	Examinateur
M. Etienne CÔME, Chargé de Recherche, IFSTTAR	Examinateur
M. Ludovic DENOYER, Maître de Conférences, LIP6, Université Pierre et Marie Curie	Examinateur
M. Cédric DU MOUZA, Maître de Conférences, ICID, CNAM	Examinateur
M. Fabrice ROSSI, Professeur, SAMM, Université Paris 1 Panthéon-Sorbonne	Directeur de thèse

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

In the name of God,
the Most Gracious, the Most Merciful.

To my dearly beloved parents and sisters,
for their patience, constant support, and unconditional love.

Acknowledgements

All praise be to God. I praise Him, request His help, ask for His forgiveness, and repent to Him. In Him I seek refuge from the evils of myself and from my sinful deeds. It is by virtue of the patience and courage that He has bestowed upon me that I was able to see this work through.

I would like to express my deepest and sincerest gratitude to my Ph.D. advisor, Prof. Fabrice Rossi, for supporting me throughout my thesis with his kindness, patience, involvement, and knowledge. On many levels, I consider Prof. Rossi a true mentor. His genuine passion for research shows up immediately as soon as he starts presenting his work (heck, he even outshines his Ph.D. students when he masterfully presents their work) and this passion ended up rubbing off on me. He is also a great teacher and countless are the times that I attended his courses in order to learn his know-how. But above all, I'm honored to consider Prof. Rossi a true friend. He was always there to comfort and reassure me even in my moments of doubt, and he kept encouraging and believing in me until the very end. Dear Fabrice, for all of the aforementioned reasons, I remain eternally indebted to you.

In my work on co-clustering trajectory data, I collaborated with Romain Guigourès and Marc Boullé from Orange Labs. I really appreciated our collaboration and therefore I would like to thank them for their great help and friendliness. I also wish Romain the best of luck defending his Ph.D. thesis and I thank him for the great moments and the many private jokes we shared together.

I thank my colleagues and friends from Télécom ParisTech for making my stay there very enjoyable: Hoa Dung Ha Duong (for the countless hours we spent playing Ingress and discussing interesting research topics while ruthlessly defending our portals against those nasty resistance agents), Simon Perrault (who kept banning me from participating in his experiments on human-computer interaction under the pretext that I would introduce bias in his sample since I am left-handed), Dông Bách Vo a.k.a. Dongy (see? we both finally made it and became doctors!), Céline Bizart (our extremely helpful and always cheerful assistant in the computer science and networks department), Florence Besnard (the guardian angel of Ph.D. students at Télécom ParisTech), Hayette Soussou, Fabien Cadoret, Damien Munch, Sylvain Frey, and all the kind persons that I had the chance of encountering there.

I am thankful to my colleagues and friends from Université Paris 1 Panthéon-Sorbonne: Bénédicte Le Grand (teaching alongside you is a great honor and I can't thank you enough for constantly lifting up my spirits), Manuele Kirsch Pinheiro (for her confidence, encouragements, and unbeatable sense of humor), Dhaou Lassoued (hang in there, my friend, and best wishes defending your thesis), Bechir Dola (his great and thorough knowledge about an unbelievably large number of topics simply fascinates me), Marie Cottrell, Jean-Marc Bardet, as well as all the Ph.D. students and researchers who welcomed me among

them (and endured my bad temper and lousy sense of humor) in the SAMM laboratory.

I extend my heartfelt thanks to all my friends for their moral support and constant motivation. Since I am sure that I am going to miss a few names (causing mayhem and diplomatic crises by doing so), I first chose not to designate anyone in particular. However, I feel compelled to express that I am especially grateful to Hamdi Mlaouhi, Hamdi Najjar, and Mehdi Nheri who stood by my side through thick and thin.

I would also like to thank my dearly beloved family. My hard-working parents made huge sacrifices throughout the years in order to provide for my sisters and myself. They were always there for me with their unconditional love and infinite support, without asking for anything in return. Dear parents, I did my very best to make you proud and I really hope that it made your sacrifices worthwhile (on a side note, please stop worrying, I will get married eventually). My two sisters are a true blessing and they are the best sisters one can wish for (well, except when they are plotting against me when playing video games together). The eldest of them is the one who took charge of proofreading this thesis (we all know how painful this can be). She literally kept harassing and lecturing me about my unforgivable ignorance of punctuation rules every single time she encountered a missing serial comma or a misplaced mark (which makes me fear she has some serious obsessive-compulsive disorder going on). The youngest joined in the "fun" later on and helped iron out some of the mistakes and errors that went unnoticed. Therefore, I thank them both for their involvement and for the multiple all-nighters they had to pull.

Finally, I would like to observe a moment of silence in remembrance of all the fallen comrades I left behind. Two hard drives, one MacBook Pro, one DualShock 3, and one Nexus 4 perished in sad and painful circumstances related to my thesis. May they all rest in peace (in pieces in the case of the poor DualShock 3).

Mohamed K. El Mahrsi
Télécom ParisTech
November, 2013

Abstract

In this thesis, we explore two problems related to managing and mining moving object trajectories.

First, we study the problem of sampling trajectory data streams. Modern location-aware devices are capable of capturing and transmitting their position at very high rates. Storing the entirety of the trajectories provided by such devices can entail severe storage and processing overheads. Therefore, adapted sampling techniques are necessary in order to discard unneeded positions and reduce the size of the trajectories while still preserving their key spatiotemporal features. In streaming environments, this process needs to be conducted "on-the-fly" since the data are transient and arrive continuously. To this end, we introduce a new sampling algorithm called Spatiotemporal Stream Sampling (STSS). This algorithm is computationally-efficient and guarantees an upper bound for the approximation error introduced during the sampling process. Experimental results show that STSS achieves good performances and can compete with more sophisticated and costly approaches.

The second problem we study is clustering trajectory data in road network environments. Most of prior work assumed that moving objects can move freely in an Euclidean space and did not consider the presence of an underlying road network and its influence on evaluating the similarity between trajectories. We present three approaches to clustering such data: the first approach discovers clusters of trajectories that traveled along the same parts of the road network; the second approach is segment-oriented and aims to group together road segments based on trajectories that they have in common; the third approach combines both aspects and simultaneously clusters trajectories and road segments. Through extensive case studies, we show how these approaches can be used to reveal useful knowledge about flow dynamics and characterize traffic in road networks. We also provide experimental results where we evaluate the performances of our propositions.

Keywords: moving objects, trajectories, road network, spatiotemporal, sampling, data streams, similarity, clustering.

Résumé

Dans cette thèse, nous explorons deux problèmes de recherche liés à la gestion et à la fouille de données de trajectoires d'objets mobiles.

Dans un premier temps, nous étudions l'échantillonnage de flux de trajectoires. Les appareils de géo-localisation modernes sont capables d'enregistrer et de transmettre leurs coordonnées géographiques à un taux très élevé. Garder l'intégralité des trajectoires capturées grâce à ces terminaux peut s'avérer coûteux tant en espace de stockage qu'en temps de calcul. L'élaboration de techniques d'échantillonnage adaptées devient alors primordiale afin de réduire la volumétrie des données en supprimant certaines positions (jugées inutiles ou redondantes) tout en veillant à préserver le maximum des caractéristiques spatiotemporelles des trajectoires originales. Dans le contexte de flux de données, ces techniques doivent en plus être exécutées « à la volée » et s'adapter au caractère à la fois continu et éphémère des données. Afin de répondre à ces besoins, nous proposons l'algorithme STSS (*Spatiotemporal Stream Sampling*). STSS bénéficie d'une faible complexité temporelle et garantit une borne supérieure pour les erreurs commises lors de l'échantillonnage. Nous présentons également une étude expérimentale à travers laquelle nous montrons les performances de notre proposition tout en la comparant à d'autres approches proposées dans la littérature.

La deuxième problématique étudiée dans le cadre de ce travail est celle de la classification non supervisée (ou *clustering*) de trajectoires contraintes par un réseau routier. La majorité des travaux traitant du clustering de trajectoires se sont intéressés au cas où ces dernières évoluent librement dans un espace Euclidien. Ces travaux n'ont donc pas pris en considération l'éventuelle présence d'un réseau sous-jacent au mouvement, dont les contraintes jouent un rôle primordial dans l'évaluation de la similarité entre trajectoires. Nous proposons trois approches pour traiter ce cas. La première approche se focalise sur la découverte de groupes de trajectoires ayant parcouru les mêmes parties du réseau routier. La deuxième approche vise à grouper des segments routiers visités très fréquemment par les mêmes trajectoires. Quant à la troisième approche, elle combine les deux aspects afin d'effectuer un co-clustering simultané des trajectoires et des segments routiers. Nous illustrons nos approches à travers divers cas d'étude afin de démontrer comment elles peuvent servir à caractériser le trafic routier et les dynamiques de mouvement dans le réseau routier. Nous réalisons des études expérimentales afin d'évaluer les performances de nos propositions.

Mots-clés : objets mobiles, trajectoires, réseau routier, échantillonnage, spatiotemporel, flux de données, similarité, classification non supervisée.

Contents

List of Figures

List of Tables

Résumé de la thèse

Grâce aux avancées technologiques réalisées dans les domaines de la télématique et de la géo-localisation, il est désormais possible de suivre les déplacements de divers objets mobiles (voitures équipées de GPS, piétons utilisant leurs smartphones ou PDAs, animaux identifiés grâce à des capteurs RFID, etc.). Les différentes positions de ces objets mobiles engendrent des trajectoires qui peuvent être collectées aisément et sauvegardées en vue d'effectuer des traitements et des tâches d'analyse et de fouille de données par la suite. Les connaissances extraites grâce à ces traitements et analyses peuvent apporter une valeur ajoutée considérable dans divers domaines applicatifs tels que : (i) l'étude de la migration animale, où la correlation des données récoltées avec d'autres données géographiques peut révéler les tendances migratoires de certaines espèces et aider à étudier l'impact de certains phénomènes (déforestation, construction de nouvelles routes, etc.) sur ces tendances ; (ii) la météorologie, où l'étude des chemins empruntés par certains phénomènes récurrents (cyclones, tornades, etc.) permet d'améliorer la prédiction de leurs zones d'impact ; (iii) la gestion du trafic routier, où l'analyse des trajectoires des automobilistes peut contribuer à l'évaluation de l'adéquation du réseau routier à l'usage qui en est fait ; etc.

À cause de la nature spatiotemporelle particulière et complexe des données de mobilité, les systèmes de gestion de bases de données (SGBD) classiques ne sont pas adéquats pour les manipuler. Des efforts considérables ont donc été entrepris afin de concevoir et développer des systèmes alternatifs plus adaptés, offrant la possibilité de modéliser, stocker et interroger des bases de données d'objets mobiles (*Moving Object Databases*) [4]. Les données de mobilité ont également motivé un grand nombre de travaux de recherche couvrant divers aspects de leur traitement, tels que l'indexation, la compression, la détection de schémas, la classification supervisée et non supervisée, etc.

La présente thèse traite deux problèmes de recherche reliés à la thématique de la mobilité :

– l'échantillonnage de flux de données de trajectoires, qui vise à réduire le volume des données « à la volée » (au fur et à mesure qu'elles sont reçues) tout en essayant de conserver leurs caractéristiques spatiotemporelles ;

– la classification non supervisée des données de trajectoires dans le contexte des réseaux routiers, dont le but est de découvrir des groupes d'obser-

vations similaires telles que des trajectoires qui empruntent les mêmes routes dans le réseau ou encore des portions routières souvent visitées par les mêmes véhicules.

Les contributions principales de nos travaux autour de ces deux problèmes sont les suivantes :

- nous proposons l'algorithme STSS (*Spatio Temporal Stream Sampling*), qui est une technique d'échantillonnage spatiotemporel adaptée au contexte de flux de données. STSS bénéficie d'une faible complexité (en temps et en mémoire vive) et fournit une borne supérieure pour les erreurs d'approximation commises lors de l'échantillonnage. Cette borne est configurable par l'utilisateur, ce qui permet de contrôler la dégradation de la qualité résultant de l'application de l'algorithme ;

- nous proposons une approche de classification non supervisée qui permet de retrouver des groupes de trajectoires similaires dans les réseaux routiers. L'approche compare les trajectoires en se basant sur les segments routiers qu'elles partagent et exploite cette information pour construire un graphe de similarité modélisant les interactions entre trajectoires. Une approche de détection de communautés est ensuite utilisée pour extraire une hiérarchie de groupes de trajectoires qui peut être explorée à différents niveaux de détail afin de comprendre les tendances de mobilité dans le réseau routier ;

- nous proposons une approche complémentaire qui effectue la classification de segments routiers en se basant sur les visites qu'ils reçoivent de la part des trajectoires. Le même principe est appliqué pour construire un graphe modélisant les relations de similarité entretenues par les segments et la même méthode de détection de communautés est utilisée. Les groupes de segments découverts par cette deuxième approche peuvent servir à comprendre les situations de congestion routière et à en prédire la propagation ;

- enfin, nous proposons une approche qui combine les deux aspects susmentionnés et qui effectue une classification simultanée de trajectoires et de segments de routes en se basant sur l'homogénéité de la densité des visites que les segments reçoivent de la part des trajectoires. L'approche peut être utilisée pour révéler des structures intéressantes dans le réseau routier telles que la présence de « hubs routiers » empruntés par plusieurs groupes de trajectoires qui ont des destinations et des zones de départ différentes. Cette dernière contribution est le fruit d'une collaboration avec Romain Guigourès et Marc Boullé (Orange Labs).

Ce résumé vise à présenter une vue d'ensemble des travaux réalisés au cours de cette thèse et qui seront décrits plus en détail à travers le reste du manuscrit.

Échantillonnage de trajectoires d'objets mobiles

Les terminaux de géo-localisation modernes sont capables de déterminer et de communiquer leur position avec une très fine granularité, pouvant aller jusqu'à une position par seconde (voire plus). Par conséquent, conserver l'intégralité des données récoltées à partir d'un nombre important d'objets mobiles risque d'être incommode et très coûteux, tant au niveau de l'espace de stockage qui doit être prévu à cet effet que du temps de calcul que les tâches d'analyse et de visualisation, effectuées par la suite, vont requérir. Des techniques de compression de données doivent donc être appliquées afin de réduire la taille des données. L'échantillonnage est une technique intuitive et simple à mettre en œuvre, qui peut être utilisée à cet effet : en supprimant, de façon intelligente, certaines positions de la trajectoire d'origine (ex. les points redondants résultant d'un véhicule qui communique plusieurs fois la même position alors qu'il est à l'arrêt au feu rouge), l'échantillonnage va permettre de réduire la taille des données tout en essayant de préserver au mieux leurs caractéristiques spatiotemporelles d'origine.

Nous nous sommes intéressés au problème d'échantillonnage de trajectoires d'objets mobiles dans le contexte particulier des systèmes de gestion de flux de données [24]. Dans de tels systèmes, les données arrivent en continu sous forme d'une séquence ordonnée et potentiellement infinie d'éléments (un élément dans le cas étudié ici correspond à une position transmise par un objet mobile donné). Un paradigme *data-push* est donc adopté au lieu du paradigme *data-pull* des systèmes de gestion de bases de données classiques : les requêtes (appelées « requêtes continues » [30]) sont enregistrées au préalable et s'exécutent en permanence pendant une durée déterminée. Chaque nouvel élément est évalué par rapport à ces requêtes dès qu'il se présente, puis il est effacé immédiatement ou gardé en mémoire pour une courte durée. Cette incapacité à stocker l'intégralité du contenu d'un flux soulève la nécessité de construire des résumés concis [33] qui fournissent une vue approximative de l'historique du flux. L'échantillonnage peut être utilisé pour construire ces résumés, comme il peut être utilisé simplement comme un mécanisme de réduction de charge (ou *load-shedding*) quand le système de gestion de flux fait face à une forte affluence des données.

Nous supposons, pour cette partie, qu'une trajectoire T est représentée sous sa forme géométrique comme étant la suite ordonnée des positions datées, occupées par l'objet mobile qui l'a engendrée :

$$T = (id, \langle P_1(t_1, x_1, y_1), P_2(t_2, x_2, y_2), ..., P_i(t_i, x_i, y_i), ...\rangle) \ ,$$

où id représente l'identifiant de la trajectoire. Chaque point P_i est composé de l'estampille temporelle t_i qui représente sa date de capture et de la position elle même, ici exprimée sous la forme de la paire (x_i, y_i) qui représente les coordonnées de l'objet mobile à la date t_i sur le plan Euclidien. Échantillonner la trajectoire T revient à ne garder qu'un sous-ensemble de ses points de départ.

Ce sous-ensemble (appelé trajectoire compressée ou trajectoire échantillonnée et noté T_C) doit couvrir l'intégralité de T (c-à-d contenir le tout premier et le tout dernier point de T) et ne doit pas contenir de points artificiels. Le processus d'échantillonnage doit obéir à certaines règles [23] et doit notamment permettre d'obtenir (i) une réduction durable de la taille des données et (ii) des données compressées qui contiennent de faibles erreurs d'approximation qui sont, de préférence, paramétrables. Dans le contexte particulier des flux de données, d'autres contraintes viennent s'ajouter à ces règles : le traitement (i) doit être effectué « à la volée » au fur et à mesure que les données arrivent au lieu d'attendre que l'intégralité d'une trajectoire soit disponible et (ii) doit avoir une faible complexité (temporelle et en espace mémoire) afin de s'accommoder avec le taux d'arrivée élevé des données et les ressources potentiellement limitées du système.

Les techniques d'échantillonnage traditionnelles (ex. l'échantillonnage uniforme, l'échantillonnage réservoir [40], etc.) ne sont pas adaptées pour échantillonner des données de trajectoires étant donné qu'elles ne tiennent pas compte de l'aspect spatiotemporel de ces dernières. Par conséquent, d'autres techniques ont été proposées pour le cas spécifique des trajectoires [23, 42, 45, 46]. Ces techniques sont principalement inspirées par les domaines de la simplification de polylignes, la généralisation cartographique et la compression de séries temporelles. En comparant ces approches par rapport aux critères évoqués auparavant, nous distinguons deux familles :

1. les algorithmes qui garantissent des bornes supérieures pour les erreurs d'approximation introduites lors de l'échantillonnage mais qui ont, en contre partie, des complexités élevées. Parmi ces algorithmes, nous citons l'algorithme TD-TR [23] (qui ne peut pas être utilisé dans un contexte de flux de données puisqu'il nécessite la présence de la totalité des données au préalable) et l'algorithme OPW-TR [23] (qui utilise le principe de fenêtres ouvrantes pour échantillonner les trajectoires à la volée) ;

2. les algorithmes avec faibles complexités mais qui n'offrent pas de garanties vis-à-vis des erreurs d'approximation. Ceci est notamment le cas des algorithmes de la famille Threshold et de l'algorithme STTrace, proposés par Potamias et al. [45]. L'utilisation de tels algorithmes ne permet pas de se prononcer sur la qualité des réponses fournies aux requêtes et analyses appliquées aux trajectoires échantillonnées.

Nous proposons l'algorithme STSS comme une alternative qui permet d'obtenir à la fois des garanties sur les erreurs d'approximation et de faibles complexités temporelle et en espace mémoire.

L'algorithme STSS (SpatioTemporal Stream Sampling)

STSS (*SpatioTemporal Stream Sampling*) est un algorithme d'échantillonnage à la volée pour les flux de trajectoires. Il est basée sur l'idée de prédiction linéaire

(similairement à [45]) : l'algorithme essaye de capturer le comportement actuel de l'objet mobile et l'utilise pour prédire ses positions futures. La capacité de l'algorithme à prédire ces positions « correctement » guidera le processus d'échantillonnage et sera utilisée comme critère pour décider si les points doivent être gardés ou supprimés.

Le comportement actuel de l'objet mobile est capturé grâce à une « fonction de mouvement » comme suit : étant donné deux points $P_k = (t_k, x_k, y_k)$ et $P_j = (t_j, x_j, y_j)$ (avec $t_k < t_j$), nous supposons que l'objet mobile continuera à se déplacer dans la même direction et avec la vitesse moyenne observées entre P_k et P_j. Par conséquent, pour tout instant antérieur t ($t > t_j$), la position de l'objet mobile peut être prédite conformément à la formule suivante :

$$(x, y) = (a_x t + b_x, a_y t + b_y) \ .$$

Avec :

$$\begin{aligned} a_x &= \frac{x_j - x_k}{t_j - t_k} \ , \\ b_x &= x_k - a_x t_k = x_j - a_x t_j \ , \\ a_y &= \frac{y_j - y_k}{t_j - t_k} \ , \\ b_y &= y_k - a_y t_k = y_j - a_y t_j \ . \end{aligned}$$

En réalité, la fonction de mouvement correspond à une extrapolation de la distance Euclidienne synchrone (*Synchronous Euclidean Distance*) introduite dans [42]. La fonction est initialisée avec les deux premiers points de la trajectoire T. Ces deux points sont insérés immédiatement dans la trajectoire échantillonnée T_C. Ensuite, pour chaque nouveau point P_i, une prédiction P'_i est calculée (en applicant la fonction de mouvement). Si la distance entre P_i et P'_i est inférieure au seuil d'erreur d_{Thres} précisé par l'utilisateur, le point est considéré comme étant bien prédit et il remplace le dernier point de T_C. Dans le cas contraire (c-à-d lorsqu'une mauvaise prédiction se produit), le point P_i est ajouté à la fin de T_C et la fonction de mouvement est recalculée en utilisant P_i et P_{i-1}.

La figure 1 illustre ce principe de fonctionnement. Afin de couvrir l'intégralité de la trajectoire d'origine, le point P_1 est immédiatement inséré dans la trajectoire échantillonnée T_C. Lorsque le point P_2 arrive, une première fonction de mouvement est calculée et le point est inséré dans T_C. L'arrivée de P_3 déclenche le calcul d'une prédiction P'_3. Le point étant bien prédit, il remplace P_2 dans T_C. À ce niveau du traitement, $T_C = \langle P_1, P_3 \rangle$. Puisque P_4 et P_5 sont également bien prédits, ils viennent remplacer, chacun à son tour, le dernier point de T_C. Par contre, une mauvaise prédiction se produit à l'arrivée de P_6, signalant un changement dans le comportement de l'objet mobile. La fonction de mouvement est donc mise à jour à partir de P_5 et P_6 et ce dernier est ajouté à T_C. À ce niveau, $T_C = \langle P_1, P_5, P_6 \rangle$. L'algorithme continue en suivant ce principe pour échantillonner les points restants de la trajectoire. La figure 2

montre un exemple d'une trajectoire réelle, échantillonnée par l'algorithme STSS avec différentes valeurs du seuil d'erreur d_{Thres}.

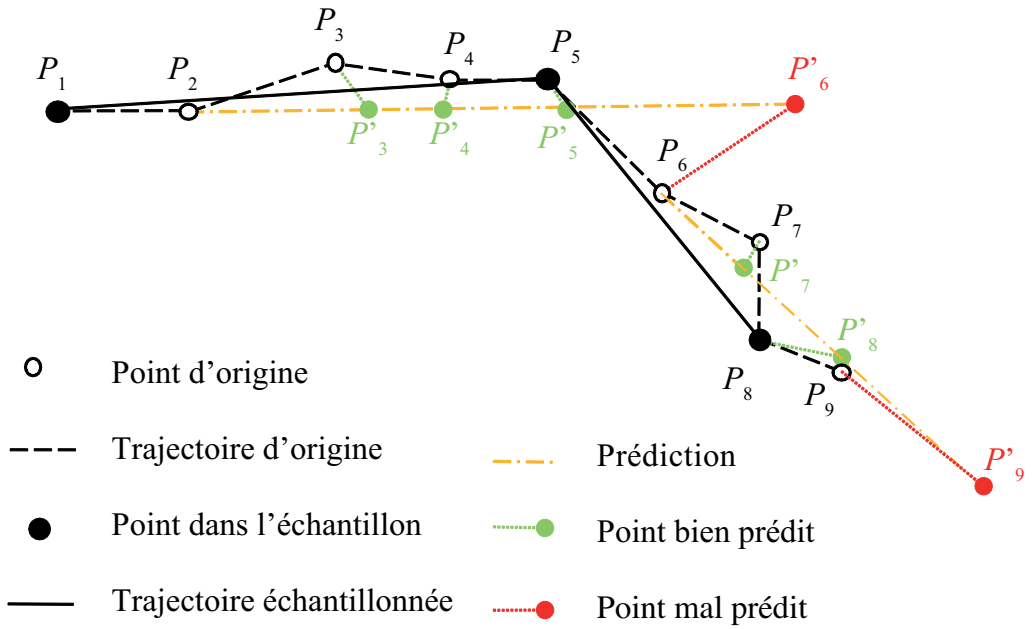
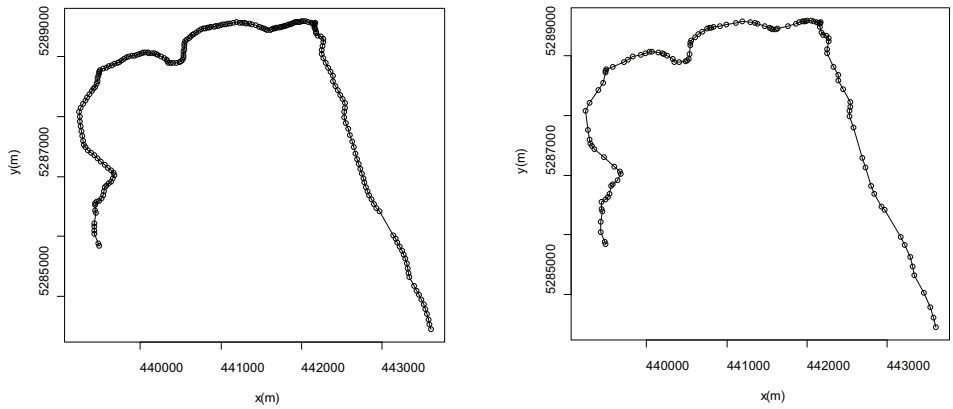


Figure 1 – Principe de fonctionnement de l'algorithme STSS.

STSS traite chaque point en un temps constant, ce qui résulte en une complexité de $O(n)$ pour traiter une trajectoire contenant n points. L'algorithme permet aussi de contrôler l'erreur d'approximation résultant de l'échantillonnage. En effet, quand STSS est configuré avec une valeur donnée pour d_{Thres}, l'erreur d'approximation pour tout point non retenu par l'algorithme est inférieure à $2d_{\text{Thres}}$ (une preuve formelle est fournie dans la section 2.3.2).

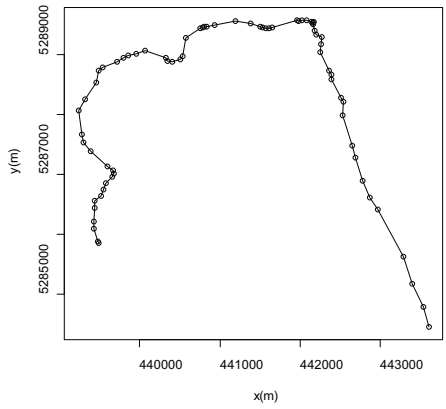
Nous avons comparé STSS avec les algorithmes TD-TR et OPW-TR [23]. Ces deux algorithmes permettent d'avoir des erreurs d'approximation bornées mais présentent un coût algorithmique plus élevé puisqu'ils ont une complexité quadratique (c-à-d $O(n^2)$ pour traiter une trajectoire de n points). Notre étude expérimentale a montré que notre approche, bien que produisant de bons résultats, est moins efficace que TD-TR et OPW-TR. Ce résultat s'explique par le fait que STSS décide immédiatement si un point donné doit être gardé dans la trajectoire échantillonnée ou non (afin de réduire la complexité algorithmique du traitement), alors que les deux autres algorithmes prennent leurs décisions en se basant sur un nombre plus conséquent de points (ceux inclus dans la fenêtre ouvrante maintenue par OPW-TR ou la totalité des points de la trajectoire pour TD-TR).

Nous avons également conduit une étude comparative avec l'algorithme STTrace [45] dans laquelle STSS a surpassé STTrace et a fourni des trajectoires échantillonnées de qualité supérieure pour le même nombre de points. Ce résultat est prévisible puisque STTrace permet de contrôler l'espace de stockage

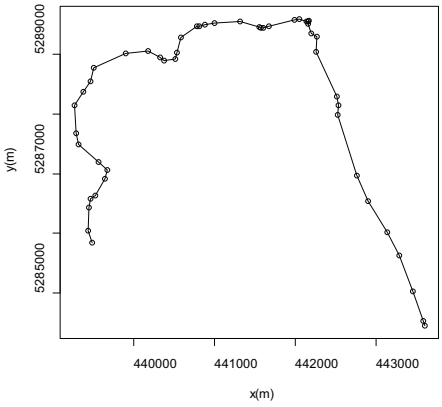


(a) Trajectoire d'origine (228 positions)

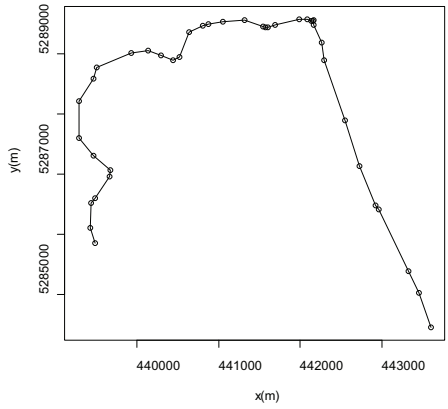
(b) $d_{\text{Thres}} = 5\text{m}$ (117 positions)



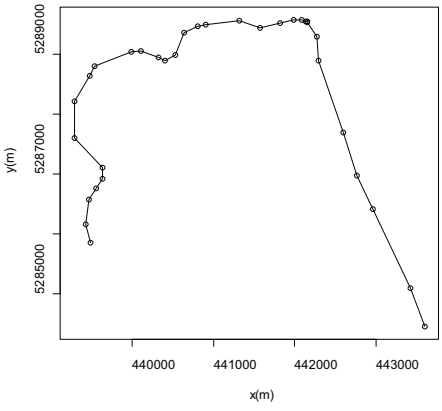
(c) $d_{\text{Thres}} = 25\text{m}$ (72 positions)



(d) $d_{\text{Thres}} = 50\text{m}$ (49 positions)



(e) $d_{\text{Thres}} = 75\text{m}$ (40 positions)



(f) $d_{\text{Thres}} = 100\text{m}$ (32 positions)

FIGURE 2 – Exemple d'une trajectoire échantillonnée en utilisant l'algorithme STSS avec différentes valeurs du seuil d'erreur de prédiction d_{Thres}.

occupé par les trajectoires échantillonnées au détriment de la qualité de celles-ci. Nous soulignons donc l'importance de fournir des garanties et des bornes pour les erreurs d'approximation introduites lors de l'échantillonnage. Ces erreurs risquent, en effet, d'impacter les résultats fournis en réponse aux requêtes et analyses conduites sur les données par la suite.

Nos travaux sur l'échantillonnage de flux de trajectoires sont décrits en détail dans le chapitre 2 de cette thèse.

Classification non supervisée de trajectoires contraintes par un réseau routier

Le monitoring du trafic routier est effectué, dans la majorité des cas, grâce à des capteurs dédiés qui permettent d'estimer le nombre de véhicules traversant la portion routière sur laquelle ils sont installés. Les coûts prohibitifs d'installation et de maintenance pour ce genre de capteurs limitent leur déploiement au réseau routier primaire (c-à-d les autoroutes et les grandes artères seulement). Par conséquent, ce genre de solutions produit une information incomplète sur l'état du réseau routier, ce qui complique l'extraction de connaissances sur la dynamique des mouvements dans ce réseau et sur l'adéquation entre le réseau et son usage. Afin de combler ce manque d'information, une approche complémentaire peut consister à exploiter des traces GPS d'objets mobiles recueillies par des dispositifs ad hoc (par exemple des smartphones). Ces traces peuvent être obtenues lors de campagnes d'acquisition spécifiques (bus, taxis, flotte d'entreprise, etc.) ou par des mécanismes de crowdsourcing en proposant à des utilisateurs de soumettre leurs propres trajets. On peut ainsi obtenir un volume important de données couvrant le réseau de façon beaucoup plus complète que les capteurs dédiés. Ces données peuvent ensuite être échantillonnées (par les approches évoquées auparavant) puis explorées grâce à des méthodes d'apprentissage et de fouille de données.

Le clustering (ou classification non supervisée) figure parmi les techniques les plus utiles à de telles fins exploratoires. Il consiste à partitionner un ensemble d'observations $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ en un ensemble de groupes (dits classes ou clusters) $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ de façon à regrouper au sein d'une même classe les observations similaires au sens d'un critère bien défini. Le deuxième problème de recherche auquel nous nous sommes intéressés dans le cadre de cette thèse est celui du clustering de trajectoires dans le cas où celles-ci sont contraintes par un réseau sous-jacent limitant leur mouvement (en particulier le réseau routier).

Les travaux liés au clustering de trajectoires peuvent être classés en deux grandes catégories : (i) l'étude de similarité entre trajectoires et (ii) la conception d'algorithmes de clustering adaptés aux trajectoires. Plusieurs distances sont proposées pour comparer des trajectoires en mouvement libre entre elles [61, 63, 64, 60, 65]. Toutes ces distances sont inadaptées au cas de trajectoires

contraintes par un réseau puisqu'elles se basent sur la distance euclidienne et ignorent les restrictions topologiques imposées par le réseau. Dans [68], Hwang et al. introduisent l'une des premières mesures de similarité adaptées au mouvement contraint. Les trajectoires sont filtrées en appliquant une similarité spatiale sur le réseau (passage par des points d'intérêt prédéfinis) puis le résultat est raffiné en appliquant un critère temporel. D'autres distances spatiotemporelles sont également présentées par Tiakas et al. [71] et Chang et al. [21].

Plusieurs approches de clustering sont proposées pour le cas particulier des trajectoires d'objets mobiles. Ces approches diffèrent, généralement, selon le choix de la représentation des données (géométrique ou symbolique), les dimensions prises en compte (spatiale, temporelle ou les deux) et la granularité du clustering (trajectoires entières, portions de trajectoires, etc.). Nanni et Pedreschi [89] adaptent l'algorithme OPTICS [76] au cas des trajectoires : la variante T-OPTICS regroupe des trajectoires entières, alors que la variante TF-OPTICS regroupe des sous-trajectoires. Dans [51], les auteurs introduisent la notion de *flock patterns* qui consistent en des groupes d'objets mobiles se déplaçant ensemble dans un disque de rayon donné. La notion de *convoy pattern* est introduite dans [55] et utilise l'algorithme DBSCAN [75] pour regrouper des objets mobiles sur plusieurs instants temporels consécutifs. Lee et al. [58] proposent l'algorithme TRACLUS : les trajectoires sont d'abord simplifiées avec un algorithme de type MDL (*Minimal Description Length*) puis des sous-trajectoires sont regroupées ensemble avec une adaptation de l'algorithme DBSCAN. Tous ces algorithmes font l'hypothèse d'un mouvement libre et utilisent des distances euclidiennes pour les comparaisons.

Dans [8], les auteurs proposent un algorithme pour découvrir des chemins denses résultant des déplacements sur un réseau routier en exploitant le principe de densité introduit par DBSCAN. La méthode est étendue dans [11] qui ajoute la prise en compte du temps. Roh et Hwang [10] proposent l'algorithme NNCluster qui exploite les calculs de plus court chemin dans le réseau routier pour évaluer la distance entre les trajectoires et les partitionner.

Une nouvelle tendance dans le clustering de trajectoires a vu le jour très récemment. Elle consiste à utiliser des techniques issues de l'analyse et du clustering de graphes en les adaptant au contexte des trajectoires. Dans [85], les auteurs construisent un graphe modélisant le nombre de « rencontres » entre les trajectoires et calculent différentes statistiques sur la structure obtenue. Une autre approche de ce genre est proposée par Guo et al. [59] où un graphe est construit avec comme sommets les points constituant les trajectoires et comme arcs le nombre de trajectoires ayant passé à la fois par les deux points reliés. Ce graphe est partitionné pour découvrir des zones d'intérêt regroupant des points souvent visités conjointement. Bien que les auteurs de ces travaux citent des applications dans le cas contraint, leurs méthodes exploitent des calculs de distance euclidienne entre points de trajectoires non contraints.

Pour cette partie, nous avons repris le modèle symbolique, qui est largement utilisé dans la littérature [7, 8, 9, 10] et qui est plus adapté pour représenter

des trajectoires dans le contexte des réseaux routiers. Dans ce modèle, le réseau routier est représenté en tant qu'un graphe orienté $\mathcal{G} = (\mathcal{V}, \mathcal{S})$. L'ensemble des sommets $\mathcal{V}$ représente les intersections et les points terminaux des routes, tandis que l'ensemble des arcs $\mathcal{S}$ représente les segments routiers qui les relient. Une trajectoire T effectuée par un objet mobile donné sur ce réseau est modélisée par la séquence de segments routiers traversés :

$$T = (id, \langle s_1, s_2, ..., s_l \rangle) \ ,$$

où id est l'identifiant unique de T, l sa longueur (nombre de segments qu'elle contient) et $\forall 1 \leq i < l, s_i$ et s_{i+1} des segments connectés appartenant à $\mathcal{S}$. La figure 3 illustre trois trajectoires contraintes. Conformément au modèle symbolique, celles-ci sont représentées comme suit : $T_1 = \langle s_1, s_7, s_{11}, s_{12}, s_{13} \rangle$, $T_2 = \langle s_1, s_4, s_3 \rangle$ et $T_3 = \langle s_{10}, s_{11}, s_8, s_5, s_6 \rangle$.

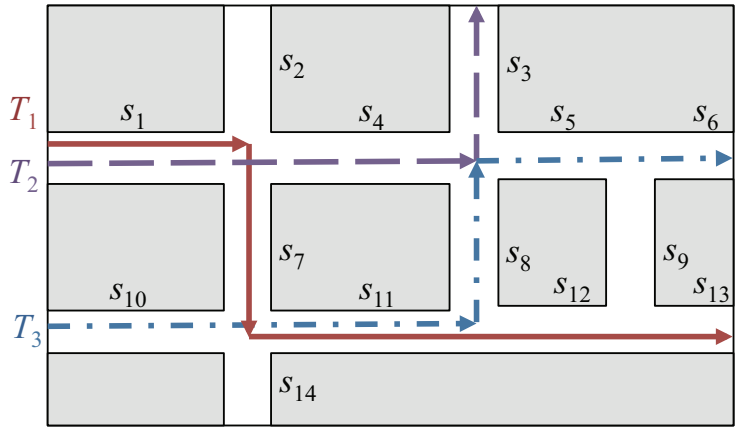


Figure 3 – Exemple de trois trajectoires contraintes par un réseau routier.

Clustering de trajectoires contraintes par optimisation de la modularité

Dans un premier temps, nous nous sommes intéressés à regrouper des trajectoires similaires. Pour évaluer la similarité de celles-ci, nous nous basons sur les segments routiers qu'elles ont visités. Ainsi, nous considérons que deux trajectoires sont similaires si elles partagent un grand nombre de segments routiers. Plus ce nombre augmente, plus elles sont similaires (jusqu'à devenir identiques si elles contiennent exactement les mêmes segments) et, vice versa, plus ce nombre décroit, plus elles deviennent dissimilaires (jusqu'à devenir totalement différentes si elles ne contiennent aucun segment en commun). Cette façon de faire est inspirée par les travaux existants dans le domaine de la recherche d'information où chaque document est considéré comme un sac de mots (ou *bag-of-words*) et où les documents sont comparés par rapport aux mots qu'ils partagent (tout en négligeant l'ordre d'apparition de ces mots).

Formellement, en inspectant une trajectoire donnée T nous affectons à chaque segment routier s qu'elle contient un poids $\omega_{s,T}$:

$$\omega_{s,T} = \frac{n_{s,T} \cdot \text{length}(s)}{\sum_{s' \in T} n_{s',T} \cdot \text{length}(s')} \times \log \frac{|\mathcal{T}|}{|\{T_i : s \in T_i\}|} \ .$$

Avec $n_{s,T}$ le nombre de fois que la trajectoire T a visité le segment s (généralement 1), length(s) la longueur spatiale de s, $|\mathcal{T}|$ le nombre total de trajectoires dans le jeu de données $\mathcal{T}$ et $|\{T_i : s \in T_i\}|$ le nombre de trajectoires qui ont visité le segment s. La première partie de $\omega_{s,T}$ correspond à la contribution du segment à la longueur totale de la trajectoire inspectée : plus le segment contribue à une trajectoire, plus il devient important pour comparer celle-ci avec d'autres trajectoires et vice versa. La seconde partie, quant à elle, permet de pénaliser les segments routiers très fréquents (qui sont visités souvent par les trajectoires) et qui ne sont donc pas particulièrement pertinents pour la formation de clusters de trajectoires. Cette méthode de pondération est une adaptation de la méthode tf–idf (*term frequency – inverse document frequency*), souvent utilisée dans la recherche d'information, au cas de trajectoires routières.

Enfin, nous mesurons la similarité entre deux trajectoires T_i et T_j en utilisant une similarité cosinus dont les valeurs varient entre 0 et 1 (0 indique deux trajectoires totalement indépendantes et 1 indique deux trajectoires identiques) :

$$\text{similarity}(T_i, T_j) = \frac{\sum_{s \in \mathcal{S}} \omega_{s,T_i} \cdot \omega_{s,T_j}}{\sqrt{\sum_{s \in \mathcal{S}} \omega_{s,T_i}^2} \cdot \sqrt{\sum_{s \in \mathcal{S}} \omega_{s,T_j}^2}} \ .$$

Afin d'effectuer le clustering des trajectoires étudiées, nous commençons par représenter les relations de similarité qu'elles entretiennent sous forme d'un graphe non-orienté et pondéré $\mathcal{G}_\mathcal{T} = (\mathcal{T}, \mathcal{E}_\mathcal{T}, \mathcal{W}_\mathcal{T})$ appelé graphe de similarité entre trajectoires (cf. figure 4). Chaque trajectoire est représentée sous forme d'un sommet dans ce graphe. Une arête relie deux trajectoires T_i et T_j si et seulement si elles partagent au moins un segment routier en commun (c-à-d elles ont une similarité cosinus non nulle), auquel cas l'arête est pondérée avec la valeur de leur similarité.

Hormis le fait qu'elle soit simple à comprendre, nous avons opté pour cette représentation des interactions entre trajectoires sous forme d'un graphe parce qu'elle met l'accent sur le fait que des trajectoires totalement différentes ne doivent, a priori, pas être regroupées du fait de l'absence d'une arête qui les relie directement.

Plusieurs approches de clustering de graphes existent dans la littérature [97, 98]. Dans le cadre des travaux qui ont été menés pendant cette thèse, nous avons opté pour l'utilisation de la détection de communautés par maximisation de la modularité [101], et ce en raison de sa popularité et des résultats prometteurs obtenus par les approches qui s'y apparentent en pratique. Étant donné un graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ constitué de l'ensemble de sommets $\mathcal{V} = \{v_1, v_2, ..., v_n\}$

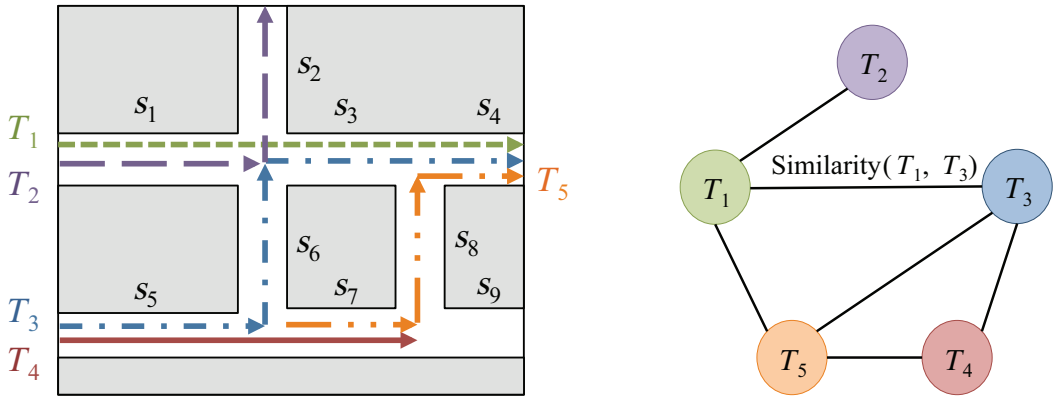


Figure 4 – Exemple d'un graphe de similarité induit par cinq trajectoires.

et de l'ensemble d'arêtes pondérées $\mathcal{E}$ avec $\omega_{ij} \geq 0$ et $\omega_{ij} = \omega_{ji}$ et étant donnée une partition des sommets de ce graphe en K clusters (appelés dans ce contexte « communautés ») $C_1, ..., C_K$, la modularité de la partition est exprimée comme suit :

$$\mathcal{Q} = \frac{1}{2m} \sum_{k=1}^{K} \sum_{i,j \in C_k} \left(\omega_{ij} - \frac{d_i d_j}{2m} \right) .$$

Où $d_i = \sum_{j \neq i} \omega_{ij}$ et $m = \frac{1}{2} \sum_i d_i$. La modularité mesure donc la qualité du clustering en inspectant la disposition des arêtes au sein des communautés de sommets : une modularité élevée indique que les arêtes à l'intérieur des communautés sont plus nombreuses (ou possèdent des poids plus élevés) que dans le cas d'un graphe où les arêtes sont distribuées de façon aléatoire.

Pour effectuer le clustering du graphe de similarité entre trajectoires $\mathcal{G}_\mathcal{T}$, nous utilisons l'algorithme décrit dans [102]. L'algorithme prend en entrée le graphe $\mathcal{G}_\mathcal{T}$ et se charge de trouver une partition optimale (c-à-d une partition qui maximise la mesure de modularité). Cette partition est validée en mesurant sa « significativité » (en comparant sa modularité à la modularité obtenue sur les partitions optimales de graphes aléatoires ayant une structure similaire à celle du graphe $\mathcal{G}_\mathcal{T}$). Si la partition est valide, l'algorithme est repris de façon récursive sur chacune des communautés (autrement dit, le sous-graphe formé par les sommets de la communauté et leurs arêtes internes est isolé et le clustering est effectué sur celui-ci). La récursivité s'arrête lorsqu'aucune des sous-partitions ne peut être partitionnée davantage.

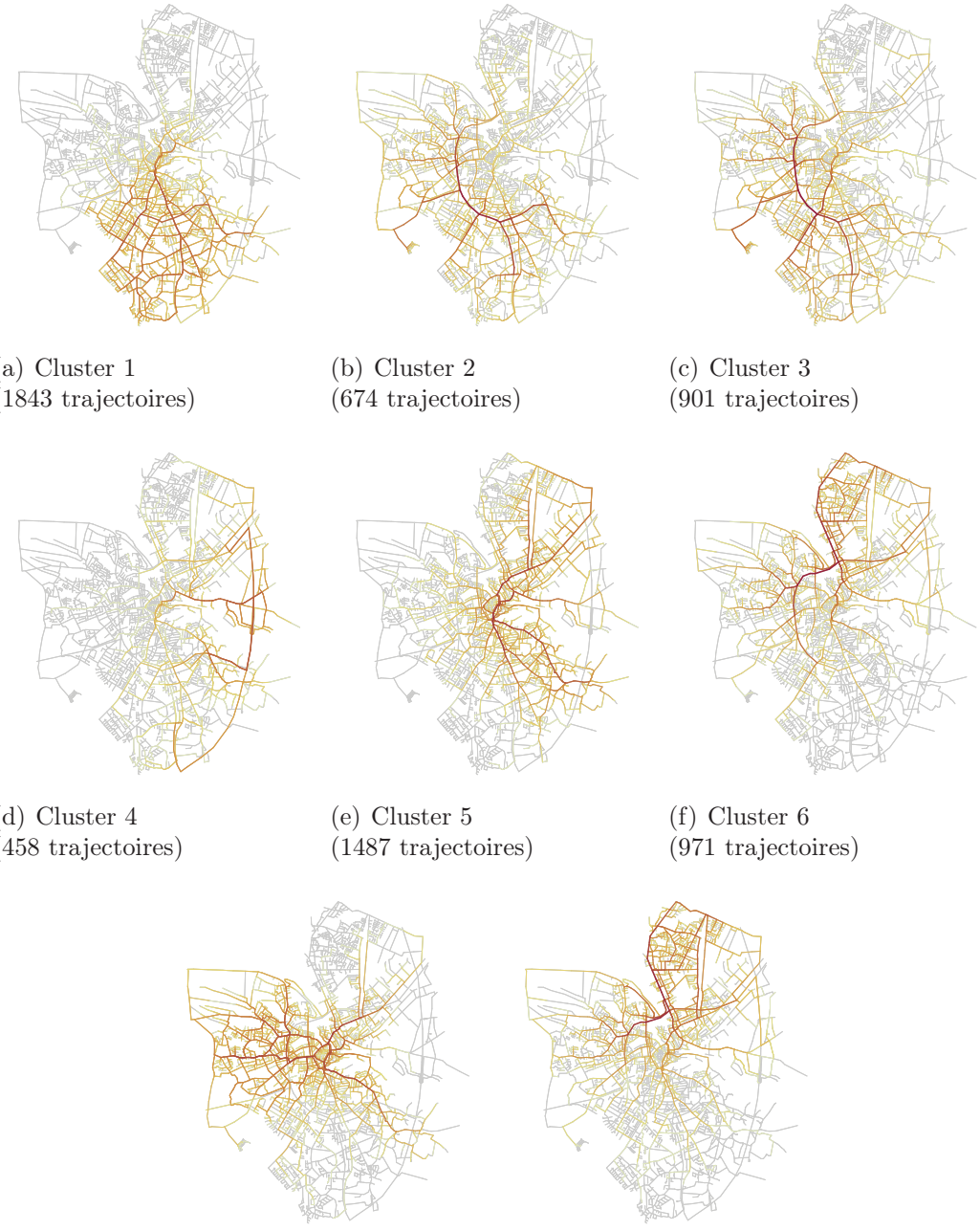
Le résultat est une hiérarchie de clusters de trajectoires imbriqués qui s'étale sur plusieurs niveaux. Ceci est un atout majeur lors de l'analyse de grands jeux de données susceptibles de contenir un nombre élevé de clusters : l'utilisateur peut commencer, au niveau le plus haut de l'hiérarchie, avec un nombre limité de clusters avec une simplification grossière pour comprendre rapidement les tendances générales de mouvement puis accéder, par zooms successifs, à plus

de détails dans les clusters qui l'intéressent. La figure 5 représente un jeu de données composé de 10000 trajectoires engendrées avec le générateur de Brinkhoff [6] sur le réseau routier d'Oldenburg, tandis que la figure 6 montre les huit clusters de trajectoires appartenant au premier niveau de la hiérarchie de clusters obtenue par application de notre approche.

FIGURE 5 – Jeu de données composé de 10000 trajectoires en mobilité sur le réseau routier de la ville d'Oldenburg. Les segments routiers sont coloriés en fonction de leur taux d'utilisation (nombre de trajectoires les ayant fréquentés) : une couleur jaune pale indique un segment peu fréquenté, alors qu'une couleur rouge foncée indique un segment très fréquenté.

Afin d'évaluer notre approche, nous l'avons comparée à la version de base de l'approche NNCluster proposée par Roh et Hwang [10] sur une multitude de jeux de données étiquetés (c-à-d qui contiennent des clusters pré-déterminés) de petite taille que nous avons générés ainsi que sur le grand jeu de données évoqué auparavant et illustré dans la figure 5. La comparaison des deux méthodes a été basée sur un ensemble de critères de qualité internes (taux de chevauchement intra-clusters) et externes (pureté et entropie [105] et indice de Rand ajusté [106]). Les résultats obtenus montrent que notre approche produit des clusters plus pertinents par rapport aux critères susmentionnés.

Notre approche de clustering de trajectoires par optimisation de la modularité est exposée en détail dans la section 3.3 dans laquelle nous discutons, en plus des généralités qui ont été présentées ici, de plusieurs aspects tels que la complexité de l'approche, l'extraction de trajectoires représentatives à partir des clusters, etc.



(a) Cluster 1
(1843 trajectoires)

(b) Cluster 2
(674 trajectoires)

(c) Cluster 3
(901 trajectoires)

(d) Cluster 4
(458 trajectoires)

(e) Cluster 5
(1487 trajectoires)

(f) Cluster 6
(971 trajectoires)

(g) Cluster 7
(2429 trajectoires)

(h) Cluster 8
(1237 trajectoires)

Figure 6 – Les huit clusters de trajectoires dans le premier niveau de la hiérarchie de clusters. Les segments routiers sont coloriés suivant le même principe que dans Figure 5.

Clustering de segments routiers par optimisation de la modularité

Une approche alternative pour effectuer le clustering de données de trajectoires contraintes par un réseau routier consiste à considérer que les observations qu'on désire regrouper sont les segments routiers plutôt que les trajectoires. Nous proposons une extension de l'approche que nous venons de présenter, qui permet d'effectuer le clustering de segments routiers.

Afin d'évaluer la similarité entre segments routiers, nous procédons par analogie au cas des trajectoires et considérons chaque segment comme le « sac des trajectoires » qui l'ont parcouru. Intuitivement, deux segments routiers sont similaires s'ils sont parcourus souvent par les mêmes trajectoires et sont dissimilaires dans le cas contraire. Plus formellement, nous utilisons une similarité cosinus pondérée pour comparer deux segments s_i et s_j :

$$\text{similarity}(s_i, s_j) = \frac{\sum_{T \in \mathcal{T}} \omega_{T,s_i} \cdot \omega_{T,s_j}}{\sqrt{\sum_{T \in \mathcal{T}} \omega_{T,s_i}^2} \cdot \sqrt{\sum_{T \in \mathcal{T}} \omega_{T,s_j}^2}} \ .$$

Ici, $\omega_{T,s}$ est le poids attribué à une trajectoire T afin de caractériser son importance pour un segment s donné en fonction de son nombre de passages sur le segment en question et le nombre total de segments qu'elle a visités :

$$\omega_{T,s} = \frac{n_{s,T}}{\sum_{T' \in \mathcal{T}} n_{s,T'}} \cdot \log \frac{|\mathcal{S}|}{|s' \in \mathcal{S} : s' \in T|} \ .$$

Les valeurs de similarité entre toutes les paires de segments routiers sont utilisées pour construire un graphe de similarité entre segments $\mathcal{G}_{\mathcal{S}} = (\mathcal{S}, \mathcal{E}_{\mathcal{S}}, \mathcal{W}_{\mathcal{S}})$. Ici, chaque sommet dans le graphe correspond à un segment routier (et non pas à une trajectoire comme c'était le cas de l'approche précédente). Une arête relie deux segments s_i et s_j si et seulement s'ils ont une similarité non nulle (c-à-d il existe au moins une trajectoire qui les a visités tous les deux). Dans ce cas, l'arête est pondérée avec la valeur de cette similarité. Un graphe de similarité entre segments routiers est illustré dans la figure 7.

Afin de partitionner le graphe de similarité entre segments routiers, nous utilisons le même algorithme de détection de communautés [102, 103] que nous avons utilisé auparavant. Le résultat est une hiérarchie de clusters de segments routiers fortement corrélés. Ces clusters peuvent être analysés, par exemple, pour détecter d'éventuels goulots d'étranglement et localiser les zones mal desservies dans le réseau routier. Ils peuvent également être croisés avec les clusters de trajectoires découverts grâce à l'approche que nous avons présentée dans la section précédente afin de caractériser le rôle que certaines portions de routes jouent dans le traffic. À titre d'exemple, la figure 8 montre un groupe de segments routiers jouant le rôle d'un « hub routier », à savoir une portion de route que des clusters de trajectoires qui diffèrent dans leurs zones de départ et/ou d'arrivée empruntent.

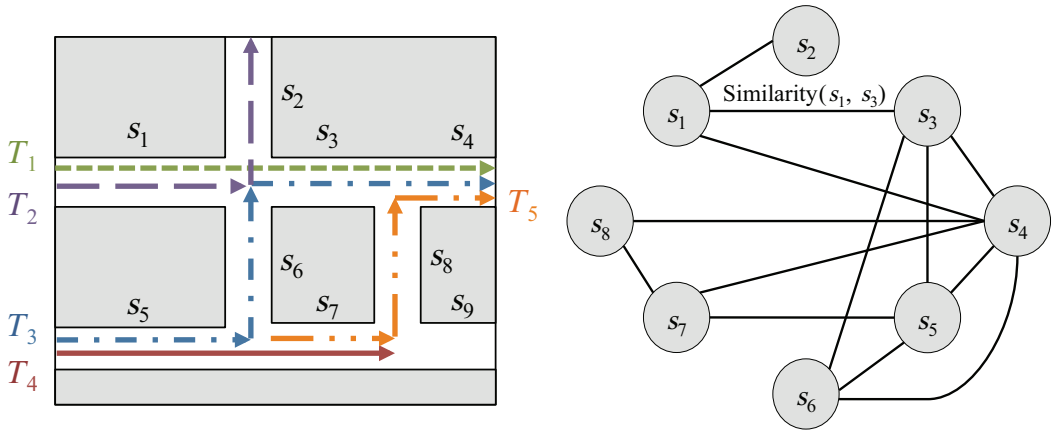


Figure 7 – Exemple d'un graphe de similarité entre segments routiers induit par les visites que huit segments ont reçues de la part de cinq trajectoires.

(a) Cluster de segments routiers constituant un hub (b) Premier cluster de trajectoires traversant le hub (c) Deuxième cluster de trajectoires traversant le hub

Figure 8 – Exemple d'un cluster de segments routiers (a) qui forment un hub traversé par deux clusters de trajectoires (b) et (c) originaires de deux zones différentes et qui l'empruntent pour se rendre à deux destinations différentes.

L'approche de clustering de segments routiers par optimisation de la modularité ainsi que l'ensemble d'expérimentations que nous avons conduites à ce sujet sont décrites en détail dans la section 3.4 du manuscrit.

Co-clustering simultané de trajectoires et de segments routiers

L'une des limitations de nos approches de clustering par optimisation de la modularité est que chacune d'elles est découplée de l'autre : le partitionnement des trajectoires (resp. segments routiers) est effectué uniquement en se basant sur leurs interactions sans essayer de caractériser la nature de leurs relations avec les segments routiers (resp. trajectoires). Il incombe donc à l'utilisateur d'effectuer séparément le clustering des trajectoires et celui des segments

routiers, puis de les croiser manuellement pour essayer d'expliquer les clusters de segments routiers en fonction des clusters de trajectoires et vice versa.

Afin de lever cette limitation, nous proposons une troisième approche qui permet d'effectuer le clustering des trajectoires et des segments routiers de façon simultanée. Cette nouvelle approche se base sur la représentation non pas sous la forme de graphes simples comme c'était le cas auparavant mais sous forme d'un graphe biparti $\mathcal{G} = (\mathcal{T}, \mathcal{S}, \mathcal{E})$ décrivant les relations entre les trajectoires et les segments routiers. Ce graphe biparti est composé de deux types de sommets : (i) l'ensemble de sommets $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ correspond aux trajectoires, tandis que (ii) l'ensemble de sommets $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ correspond à l'ensemble de segments composant le réseau routier qui ont été visités au moins par une trajectoire. Une arête $e \in \mathcal{E}$ relie une trajectoire T et un segment routier s si T a visité le segment s au moins une fois. Cette représentation sous forme de graphe biparti est illustrée dans la figure 9.

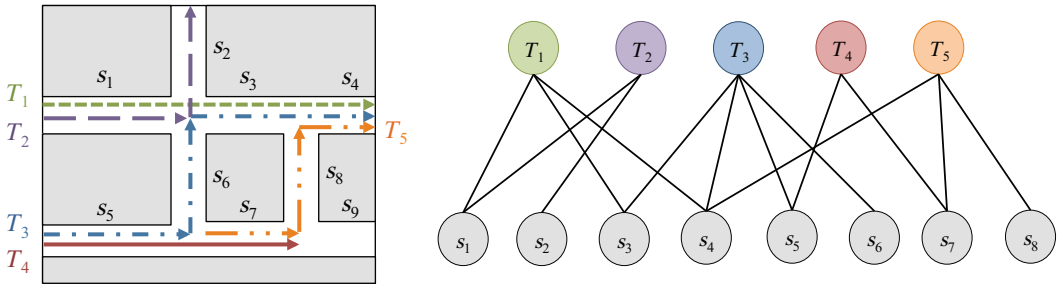


FIGURE 9 – Exemple d'un graphe biparti modélisant les interactions entre cinq trajectoires et les huit segments routiers qu'elles ont visités.

Nous proposons d'étudier le clustering du graphe biparti en appliquant une approche de co-clustering sur sa matrice d'adjacence. Dans cette matrice, les segments sont représentés en colonnes, les trajectoires en lignes et l'intersection d'une ligne et d'une colonne indique le nombre de passages d'une trajectoire sur un segment. Le but d'un co-clustering est de réordonner les lignes et les colonnes de manière à faire apparaitre et à extraire des blocs de densités homogènes dans la matrice d'adjacence du graphe biparti $\mathcal{G}$. Une fois ces blocs extraits, on en déduit deux partitions obtenues simultanément, une de segments et une de trajectoires.

Une structure de co-clustering, que nous notons $\mathcal{M}$, est définie par un ensemble de paramètres de modélisations décrits dans le tableau 1. Le but d'un algorithme de co-clustering est d'inférer la meilleure partition du graphe.

En appliquant ce type d'approches, les trajectoires sont regroupées si elles parcourent des segments communs et les segments sont regroupés s'ils sont parcourus par des trajectoires communes. L'avantage de cette technique est qu'elle ne requière ni pré-traitement sur les données, ni définition de mesure de similarité entre trajectoires ou entre segments (comme dans les approches précédentes).

Graphe biparti $\mathcal{G}$	**Modèle de co-clustering $\mathcal{M}$**
$\mathcal{T}$: ensemble des trajectoires	$C_\mathcal{T}$: ensemble des clusters de trajectoires
$\mathcal{S}$: ensemble des segments	$C_\mathcal{S}$: ensemble des clusters de segments
$\mathcal{E} = \mathcal{T} \cap \mathcal{S}$: ensemble des passages des trajectoires sur les segments	$C_\mathcal{E}$: co-clusters de trajectoires et de segments

Table 1 – Notations.

Nous avons choisi, dans le cadre de cette thèse, d'utiliser l'approche MODL [125] afin d'inférer la structure de co-clustering. MODL est non-paramétrique et a des capacités de passage à l'échelle nous permettant de l'utiliser pour analyser de grands jeux de données de trajectories. Un critère est construit suivant une approche MAP (Maximum A Posteriori) :

$$\mathcal{M}^* = \underset{\mathcal{M}}{\operatorname{argmax}}\, P(\mathcal{M})P(\mathcal{D}|\mathcal{M}) \ .$$

D'abord, une probabilité a priori $P(\mathcal{M})$ dépendant des données est définie. Elle spécifie les paramètres de modélisation en attribuant à chacun d'eux une pénalisation correspondant à leur longueur de codage minimale. Ainsi, plus une structure de co-clustering sera parcimonieuse, moins elle sera coûteuse. Ensuite, la vraisemblance des données connaissant le modèle $P(\mathcal{D}|\mathcal{M})$ est définie. Elle mesure le coût de recodage des données $\mathcal{D}$ avec les paramètres du modèle $\mathcal{M}$. Le modèle de co-clustering le plus probable est le modèle le plus fidèle aux données initiales. En d'autres termes, la vraisemblance favorise les structures informatives. La définition du critère global est donc un compromis entre une structure de co-clustering simple et synthétique, et une structure fine et informative.

D'un point de vue algorithmique, l'optimisation est réalisée à l'aide d'une heuristique gloutonne ascendante, initialisée avec le modèle le plus fin, c'est-à-dire avec un segment et une trajectoire par cluster. Elle considère toutes les fusions entre les clusters et réalise la meilleure d'entre elles si cette dernière permet de faire décroitre le critère optimisé. Cette heuristique est améliorée avec une étape de post-optimisation, pendant laquelle on effectue des permutations au sein des clusters. Le tout est englobé dans une métaheuristique de type VNS (*Variable Neighborhood Search* [130]) qui tire profit de plusieurs lancements de l'algorithme avec des initialisations aléatoires différentes. L'algorithme est détaillé et évalué dans [125].

Les co-clusters découverts peuvent être analysés afin de caractériser le traffic routier et le rôle des différentes portions de routes dans celui-ci. On peut notamment étudier la contribution des co-clusters à l'information mutuelle. L'information mutuelle, qui est une mesure fréquemment utilisée en co-clustering, permet de quantifier les dépendances entre les partitions de segments et de trajectoires. Elle est toujours positive et est d'autant plus importante que les clusters de trajectoires parcourent des clusters de segments uniques. La

contribution à l'information mutuelle, notée $mi(c_S, c_T)$, permet quant à elle de quantifier l'apport d'un co-cluster formé d'un cluster de trajectoire c_T et d'un cluster de segments c_S à l'information mutuelle du modèle. Elle est définie comme suit :

$$mi(c_S, c_T) = P(c_S, c_T) \log \frac{P(c_S, c_T)}{P(c_S)P(c_T)} \ ,$$

où $P(c_S, c_T)$ désigne la probabilité pour un passage d'appartenir à une trajectoire de c_T et de couvrir un segment de c_S, $P(c_S)$ la probabilité de parcourir un segment du cluster c_S et $P(c_T)$ la probabilité d'être sur une trajectoire de c_T. Une visualisation des contributions à l'information mutuelle des différents co-clusters que nous avons découverts en analysant un jeu de données de trajectoires est montrée dans la figure 10.

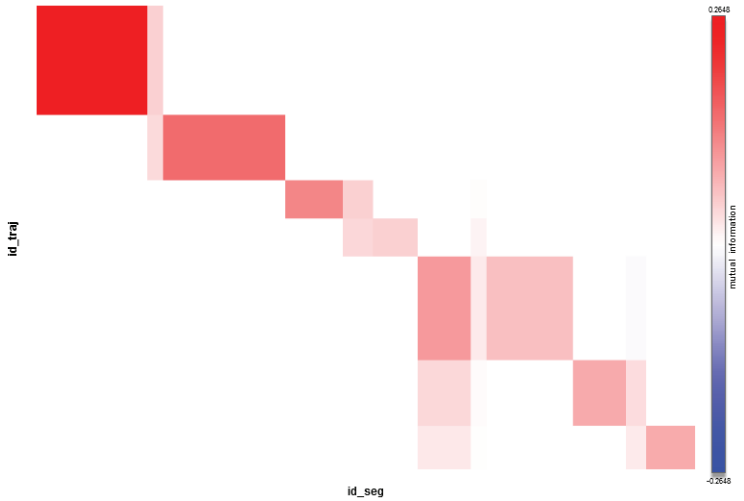


FIGURE 10 – Visualisation des contributions à l'information mutuelle des co-clusters découverts dans un jeu de données de trajectoires. Les clusters de trajectoires sont répertoriés sur la verticale, tandis que les clusters de segments sont répertoriés horizontalement. Chaque intersection entre un cluster de trajectoires et un cluster de segments représente un co-cluster. La couleur de la cellule qui correspond à ce dernier indique l'importance de sa contribution à l'information mutuelle.

Deux interactions distinctes entre clusters de segments et clusters de trajectoires peuvent être caractérisées :

- certains clusters de segments interagissent uniquement avec un seul cluster de trajectoires. Les segments routiers correspondent souvent dans ce cas à des axes secondaires et aux zones de départ et/ou d'arrivée des trajectoires (figure 11) ;
- d'autres clusters de segments sont parcourus par plusieurs clusters de trajectoires. Dans ce cas, les segments forment un « hub routier » (à

l'image de celui déjà montré dans la figure 8) auxquels les trajectoires convergent à partir de zones différentes ou divergent pour aller à des destinations différentes. La détection de la présence de ces hubs peut être très utile dans le contexte de la gestion du traffic routier : si des situations de congestion se produisent fréquemment au niveau d'un hub routier, des solutions peuvent être développées en fonction de la nature des groupes de trajectoires qui le parcourent (en proposant, par exemple, un trajet alternatif à l'un de ces groupes afin d'alléger la charge du hub).

(a) 34 segments (b) 40 segments (c) 77 segments

Figure 11 – Exemples d'axes routiers secondaires conduisant vers les zones périphériques du réseau routier.

Notre approche de co-clustering, qui est le fruit d'une collaboration avec Romain Guigourès et Marc Boullé d'Orange Labs, est décrite dans le chapitre 4. Un cas d'étude et une étude expérimentale visant à comparer nos trois approches sont également présentés dans ce même chapitre.

Conclusion et perspectives

Les travaux effectués pendant cette thèse ont porté sur deux axes de recherche liés à l'analyse de données de mobilité : (i) l'échantillonnage de flux de trajectoires et (ii) la classification non supervisée (clustering) de trajectoires dans les réseaux routiers.

Nous avons proposé l'algorithme STSS qui permet d'effectuer l'échantillonnage spatiotemporel à la volée des données de trajectoires transmises en flux continu. L'algorithme est efficace en terme de temps de calcul et offre une borne supérieure paramétrable pour les erreurs d'approximation introduites par l'échantillonnage. Il est adapté à une utilisation non seulement pour la construction de résumés historiques des trajectoires collectées mais également en tant que mécanisme de réduction de charge si le système fait face à un fort taux d'arrivée des données.

Nous avons également proposé trois approches pour effectuer le clustering de données de trajectoires dans le cas où elles sont contraintes par un réseau

sous-jacent, particulièrement le réseau routier. L'utilisation de ces approches permet d'extraire des connaissances utiles pour comprendre le traffic routier et le caractériser. Elles permettent, par exemple, d'extraire des groupes de trajectoires qui effectuent les mêmes trajets, de retrouver des segments routiers souvent visités ensemble, de caractériser les rôles que différentes portions routières remplissent, etc.

Plusieurs perspectives et extensions sont possibles à l'issue de ce travail. Parmi celles-ci nous citons les trois suivantes (nous en citons d'autres dans le chapitre 5) :

- compression multi-trajectoires : les différentes approches d'échantillonnage que nous avons explorées effectuent l'échantillonnage de chaque trajectoire séparément. Une approche alternative peut consister à inspecter toutes les trajectoires simultanément et exploiter certaines redondances qu'elles peuvent présenter (ex. passages par les mêmes points) afin de réduire la taille des données en mutualisant certaines informations (ex. garder une seule trajectoire pour représenter un groupe de trajectoires qui se sont déplacées ensemble). Cette piste est intéressante dans la mesure où elle se situe à la jonction des deux problèmes de recherche que nous avons étudiés séparément dans le cadre de ce travail ;

- gestion du temps dans le clustering : les approches de clustering que nous avons proposées évaluent la similarité entre les observations (trajectoires et segments) d'un point de vue purement spatial. Il serait intéressant d'intégrer la dimension temporelle dans ce processus d'autant que celle-ci joue un rôle très important dans le contexte de détection et d'analyse de congestions routières ;

- inférence de relations sociales à partir de données de mobilité : beaucoup de données de mobilité sont engendrées par une activité sociale qui leur est sous-jacente (plusieurs amis peuvent se déplacer naturellement en groupe en jouant à un jeu en réalité augmentée ou pour se rendre à un événement donné, etc.). Il serait intéressant d'explorer la possibilité de caractériser les relations sociales entre individus (ex. présence de liens d'amitié par exemple) en analysant les corrélations entre leurs données de mobilité.

General Introduction

General Context

In the last couple of decades, outstanding advancements have been achieved in the fields of telecommunications and geo-positioning. Such advancements are reflected by instances like the transition from the use of "dumb" mobile phones to smartphones that are constantly connected and perfectly capable of locating their owners, or the increasing number of vehicles equipped with GPS (Global Positioning System) navigation devices. The progress witnessed by these fields granted us the possibility of monitoring the whereabouts of "moving objects" of different natures (e.g. GPS-equipped vehicles, pedestrians using their smartphones and PDAs, RFID-tagged animals, etc.), but also the ability to transmit these positions and store them in view of future analysis and usage.

Fundamentally, any entity whose different position changes can be tracked and registered may be considered as a moving object. When these position changes are correlated, they induce a "trajectory" describing the journey of the moving object from one place to another throughout time. This new type of data was received with great enthusiasm and many companies are constantly racing to offer their customers the best of what LBS (Location-Based Services) can provide. This can be perceived through the great number of commercially-available products whose core added-value stems from the use of location data. Examples of such products include Waze [1], a crowdsourced turn-by-turn navigation tool that offers real-time routing and traffic jam avoidance based on positioning information collected from the users; OpenStreetMap [2], by far the most complete publicly-available map of the world, built mainly from trajectories that contributors collected using their location-aware devices; and the list goes on. Very recently, mobility data were even used to provide a new gaming experience with the augmented reality game Ingress [3].

Since classic Database Management Systems (DBMS) are not well adapted to the constantly evolving, spatiotemporal nature of mobility data, considerable efforts have been made by the research community in order to develop more suitable alternatives. These alternatives are generally known as Moving Object Databases (MOD) [4] and aim to offer the possibility of modeling, storing, and querying trajectory data efficiently. With such operations made possible, researchers moved to studying more complex problems related to the analysis and data mining of trajectory data in order to extract insightful knowledge

about moving objects of different natures. This was mainly motivated by the great potential of these tasks and their usefulness in a multitude of domains as they open the door to a wide variety of applications, such as:

- Study of animal migration: the trajectories of different animal species can be collected using RFID (radio-frequency identification) tags then analyzed in order to reveal their migratory patterns. The data can also be correlated with other geographic information in order to study, for example, the repercussions of phenomena such as deforestation or road planning on these patterns.

- Meteorology: studying the past paths followed by reoccurring meteorological phenomena, such as hurricanes, can enhance the accuracy of predicting their future landfall locations. This can greatly contribute to reducing their damage and saving human lives.

- Fraud detection: fleet management companies can analyze the trajectories of their vehicle fleets in order to detect outliers and suspicious behaviors (e.g. a cab driver who is deliberately taking unusual and lengthy routes, a delivery truck making excessively frequent stops, etc.) and react accordingly.

- Carpooling: by correlating the daily commutes of different drivers, we can detect opportunities for commuters to share their rides and reduce their travel costs.

One particular domain that can benefit greatly from moving object trajectory data is traffic analysis and monitoring. Recent reports, such as the Texas A&M Transportation Institute's 2012 Urban Mobility Report [5], paint a gloomy picture of road traffic as they show alarming and concerning figures about economical losses and environmental damage caused by road congestions and traffic jams. Collecting trajectories from vehicles commuting on the road network can complement traditional traffic monitoring approaches (mainly conducted using dedicated sensors) and can contribute greatly to our understanding of flow dynamics in the road network. For example, detecting patterns in the collected data can reveal the presence of exceptionally dense and congested areas, the tendency of vehicles to take a particular path to visit a given area, underserved areas that do not benefit from sufficient roads, etc. Such patterns can reveal the inadequacies between the road network's capacity and its real usage and can consequently be used for more informed decision-making in future road planning.

The work described in this thesis is positioned within this general context since we are interested in studying two research problems related to moving object trajectories.

Studied Research Problems

Research on moving object trajectories and moving object databases covers a wide variety of subjects [4] like spatiotemporal indexing techniques, compression techniques, privacy preservation, analytical tasks (pattern mining, clustering, supervised classification, outlier detection), etc. In this thesis, we study two problems related to mobility data: (i) sampling moving object trajectories in data stream environments, and (ii) clustering trajectory data in the particular context of road networks.

Sampling Moving Object Trajectory Data Streams

Modern GPS devices are capable of logging their positions at a high rate (one position per second). Collecting and storing the entirety of such trajectories, especially when they are generated by a large number of moving objects, is therefore impractical and, in many contexts, infeasible as it entails severe computational and storage overheads. Moreover, trajectories logged at such high rates often contain redundancies (e.g. when a vehicle is at a red light and keeps reporting the same position). Consequently, adapted compression techniques are needed in order to reduce the size of the data. Among these techniques, sampling is a natural and appealing choice which consists in deleting unnecessary and redundant positions in a given trajectory. Since sampling is a lossy compression process, one of its main challenges is to propose adapted algorithms that preserve most of the spatiotemporal features of the trajectory and provide guaranteed (and preferably controllable) bounds for the resulting compression errors.

We study the trajectory sampling problem in the particular context of streaming environments in which data are transient and need to be processed on-the-fly (as soon as they arrive). This introduces additional restrictions since proposed techniques are interdicted from backtracking on the data indefinitely and need to be efficient in order to cope with high arrival rates.

Clustering Moving Object Trajectory Data in Road Network Environments

Cluster analysis is a popular unsupervised learning task often used for exploratory purposes. Given a set of observations, clustering's main objective is to discover groups (called clusters) of "similar" observations w.r.t. a given criterion. Applying this technique to trajectory data can help uncover valuable knowledge about the behavior of moving objects and their general mobility trends and patterns.

While the problem was studied extensively in the case of static data, clustering of moving object trajectories attracted attention only recently and is still at an early stage. The complex and particular nature of trajectory data brings additional research challenges since it requires defining new similarity

and distance measures suitable for comparing trajectories. Additionally, moving objects in real applications often evolve on networks (e.g. road networks for vehicles, air corridors for airplanes, etc.). These networks underly and restrict the motion of the moving objects. Such restrictions must consequently be accounted for during the clustering process in order for the discovered clusters to be meaningful.

The biggest part of the work we present in this thesis is dedicated to studying different formulations of the moving object trajectory data clustering problem in the particular context of road network environments.

Contributions

The main contributions of this work can be summarized as follows:

- We propose the STSS (Spatiotemporal Stream Sampling) algorithm: a technique that addresses the needs of trajectory sampling in streaming environments. STSS combines the two desirable properties we mentioned earlier: (i) it benefits from a low, linear time complexity, and (ii) it guarantees an upper bound for the compression error. Moreover, this error bound is directly configurable by the user.

- We propose a network-constrained trajectory clustering approach based on community detection in graphs. Often in road network environments, trajectories are depicted as series of traveled road segments. We take advantage of this representation and define a similarity measure that evaluates the resemblance between trajectories by intelligently comparing the common road segments they visited. The similarities between trajectories are used to build a trajectory interaction graph that we cluster using a well-known quality criterion in order to discover relevant and meaningful groups of trajectories with similar behaviors. Contrary to "flat" clustering approaches, ours provides the user with a hierarchy of nested clusters that can be explored progressively with increasing levels of detail, which can be practical when analyzing clusters discovered in large datasets.

- We also propose an approach to clustering road segments based on the visits they receive from trajectories. This approach is defined analogically to the trajectory clustering approach we mentioned earlier: we use a graph representation to depict similarities between pairs of road segments then we use community detection to discover a hierarchy of road segment clusters. A cluster in this case corresponds to a group of segments that are frequently visited together, an information that can potentially be used to understand and predict the propagation of traffic jams.

- Finally, we propose a co-clustering approach in order to partition trajectories and road segments simultaneously (instead of the separate approaches

we just described). This approach produces co-clusters that contain both trajectories and road segments regrouped based on the homogeneity of the visits the segments receive from the trajectories. These co-clusters can be used to characterize the road traffic and discover interesting structures such as hubs (i.e. portions of the road network used by multiple groups of vehicles traveling to different areas), etc. This last contribution is the result of a collaboration with Romain Guigourès and Marc Boullé from Orange Labs.

Outline of the Thesis

This thesis is composed of five chapters. In Chapter 1, we start by discussing how trajectory data can be represented and present the data models we will be using for our work. We briefly talk about how trajectories can be acquired in real life, and, since we will be testing some of our approaches on synthetic datasets, we also discuss how moving object trajectories can be generated artificially.

Sampling moving object trajectories in streaming environments is treated in Chapter 2. We start by giving our formal definition of the problem and state its main objectives then move on to discuss existing work in both data stream processing and trajectory sampling. Guided by the shortcomings we noticed in related work, we present the STSS (Spatiotemporal Stream Sampling) to address the need for both efficiency and quality requirements for sampling trajectory streams. We also provide an experimental study consisting in a performance comparison between STSS and other existing algorithms and conclude on general remarks about the pros and cons of our approach.

We study clustering trajectory data in road network environments using graph-based approaches in Chapter 3. We start by formalizing two problems: (i) the network-constrained trajectory clustering problem, and (ii) the road segment clustering problem. We present a survey of existing work on trajectory similarity and clustering algorithms, and give a brief glimpse into graph clustering. In light of the observations we made about existing trajectory clustering techniques, we present our approach to clustering network-constrained trajectories based on modeling their similarities using a graph representation and clustering this graph in order to discover meaningful clusters corresponding to natural group structures. We demonstrate our approach through a comprehensive case study and evaluate its quality using multiple synthetic datasets. In the second portion of Chapter 3, we extend this approach to the road segment clustering problem. We also test this second approach on synthetic datasets and conclude with remarks on both propositions.

We continue studying the network-constrained trajectory data clustering problem in Chapter 4 in which we try to exploit the duality we noticed between the two original problems we defined in Chapter 3. To do so, we try to combine both problems and study the co-clustering of both trajectories and

road segments simultaneously. We formalize an alternative model that uses a bipartite graph to describe interactions between both entities and partition the adjacency matrix of this graph to extract co-clusters implicating highly interrelated groups of trajectories and groups of segments. We then point out, through experimental studies, some key differences between this co-clustering approach and the approaches presented in Chapter 3.

We conclude this thesis in Chapter 5 in which we revisit and summarize the main contributions of this work and present possible extensions and open issues.

Chapter 1

Representation, Acquisition, and Generation of Moving Object Trajectories

With the widespread use and availability of location-aware devices (such as GPS, smartphones, etc.), modeling, managing, and mining moving trajectories became very active research areas and received a considerable amount of interest in the last few years. To put it simply, a moving object can be intuitively defined as an object whose position changes over time. Pedestrians and vehicles are obvious examples of moving objects but the concept extends to any entity whose trajectory can be tracked and recorded (e.g. migratory animals, hurricanes, etc.).

When in presence of a new type of data (in this case, mobility data), one of the first questions that need to be answered is: "how can the data be represented?" Choosing an appropriate data model is of paramount importance and has a considerable impact on practically any task that will be conducted on the data later on. Two other questions are: "how is the data acquired?" and "if no real data are available, how can we still test and experiment with new approaches and techniques?"

In this chapter, we discuss the aforementioned questions and seize the opportunity to introduce the basic concepts, data models, and tools that will be used in the rest of this thesis. In Section 1.1, we present two trajectory data models: (i) the geometric data model used to represent trajectories with no restrictions on their motion, and (ii) the symbolic data model used to express trajectories that must adhere to constraints dictated by the presence of an underlying network (e.g. the road network). In Section 1.2, we talk about how trajectory data can be acquired and, if not available, generated using tools like the Brinkhoff generator [6]. We also present, in this same section, a simple data generation strategy that we defined to answer specific needs of some of the experimental studies that we report later in this thesis.

1.1 Representing Trajectory Data

In this section, we present the two data models that will be used throughout this thesis to represent trajectory data. Section 1.1.1 presents the geometric model in which trajectories are modeled based on their real spatial and temporal features. Section 1.1.2 presents the data model used to symbolically represent vehicle trajectories in road network environments. Alternative models can be found in [4].

1.1.1 Geometric Model

Ideally, a trajectory of a moving object would be acquired as a smooth, continuous curve described through a function from time to geographical space. However, due to inherent limitations of acquisition devices, this is far from reality. In reality, trajectories are acquired as series of discrete and possibly noisy sample positions that need to be interpolated in order to have an estimate of the missing parts.

In the "geometric" model, trajectories are represented as sets of timestamped positions in Euclidean space (generally in the two or three dimensional space) without accounting for any additional details or constraints related to the surroundings of these trajectories. Formally, and without loss of generality, a two-dimensional moving object trajectory in the geometric model can be defined as follows:

Definition 1 (Two-Dimensional Trajectory)**.** The trajectory T of an object o moving over a two-dimensional Euclidean space is an ordered series of timestamped positions (1.1):

$$T = (id, \langle P_1(t_1, x_1, y_1), P_2(t_2, x_2, y_2), ..., P_i(t_i, x_i, y_i), ...\rangle) \ . \tag{1.1}$$

id is a unique identifier assigned to the trajectory T and its positions. For each reported position P_i, the pair (x_i, y_i) represents the coordinates of o captured at the date (or timestamp) t_i.

As mentioned before, interpolation is needed to "link the dots" and approximate the positions of the moving object at missing timestamps. The most commonly used approach is piecewise linear interpolation: for a given timestamp t belonging to a time interval $[t_i, t_{i+1}]$ represented by the two sample points P_i and P_{i+1}, the position (t, x, y) of the moving object o can be estimated according to Formula (1.2):

$$(t, x, y) = (t_i, x_i, y_i) + \frac{t - t_i}{t_{i+1} - t_i}(t_{i+1} - t_i, x_{i+1} - x_i, y_{i+1} - y_i) \ . \tag{1.2}$$

Piecewise linear interpolation makes a number of strong assumptions: (i) the moving object is supposed to move in a single, straight direction in-between

successive sample points while (ii) having a constant speed (the minimal average speed to travel from (x_i, y_i) to (x_{i+1}, y_{i+1})). This results in abrupt changes in speed and direction between transitions. However, linearly-interpolated trajectories are fast to compute and easier to handle than those interpolated with more sophisticated techniques.

Alternatively, interpolation with Bezier curves can be used to model the spatial coordinates as three-degree polynomials of the time coordinates. While Bezier curves result in smoother transitions, the interpolated trajectories they yield are harder to manipulate (e.g. with operations such as intersections calculation) than their linearly-interpolated counterparts.

As stated in [4], "all these interpolation methods have one thing in common. The more sample points there are (i.e. the closer they are in time) the more accurate the trajectories will be." This observation leads to an inevitable trade-off between accuracy and computation and storage space requirements: more positions mean more accuracy but also more space and more processing time, and vice versa. Therefore, compression techniques are needed in order to intelligently reduce the size of trajectories while preserving their characteristic features. This aspect is discussed in more detail in Chapter 2 where we present the STSS trajectory sampling algorithm.

One of the concepts underlying the geometric model is the concept of routes. Projecting a trajectory on the Euclidean space yields its route, defined as follows:

Definition 2 (Route)**.** The route T_R of a trajectory T is its projection on the $X - Y$ plane. In other words, T_R is the purely spatial component of T deprived from its temporal information.

It is therefore possible for two or more different trajectories to share the exact same route if they pass by the exact same places but at totally different times. An example of a trajectory and its route is shown in Figure 1.1.

1.1.2 Symbolic Model

In most realistic applications, moving objects do not move freely and are subject to constraints and limitations imposed by their surroundings. For instance, vehicles move along the highways and roads of the road network, airplanes must stay within well-defined air corridors during their flight, etc. Such trajectories are called "network-constrained trajectories." It is advisable, in such situations, to include the constraints of the underlying network in the data model.

In this thesis, we are interested in analyzing trajectories of vehicles evolving in road networks. Consequently, we present the data model used in this particular context. The model, however, can be easily adapted to other network-constrained trajectories.

Existing work involving trajectories in road network environments [7, 8, 9, 10] implicitly agree on the use of a symbolic data model in which a road

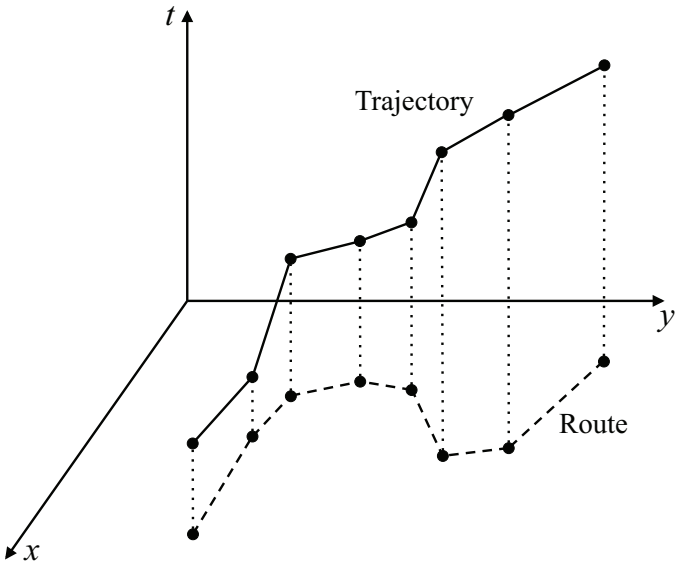


Figure 1.1 – Example of a trajectory obtained through piecewise linear interpolation of sampled positions and its corresponding route.

network is represented as a graph depicting intersections and road portions. This representation is formalized through Definition 3.

Definition 3 (Road Network)**.** The road network is represented as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{S})$. The set of vertices $\mathcal{V}$ represents intersections and terminal points of roads, whereas the set of directed edges $\mathcal{S}$ represents the road segments interconnecting them. A directed edge $s = (v_i, v_j)$ indicates that a road segment links the two vertices v_i and v_j and that it can be traveled from v_i in the direction of v_j but not the other way around (unless otherwise stated by another edge).

An example of a simple road network and its graph representation is depicted in Figure 1.2.

Given the graph representation in Definition 3, vehicles traveling along the road network produce trajectories that can be modeled conformably to Definition 4.

Definition 4 (Network-Constrained Trajectory)**.** A constrained trajectory T that travels along the road network $\mathcal{G}$ can be modeled as a sequence of visited segments:

$$T = (id, \langle s_1, s_2, ..., s_l \rangle) \ . \tag{1.3}$$

id being the identifier of the trajectory, l its length (i.e. number of visited road segments) and $\forall 1 \leq i < l, s_i$ and s_{i+1} are connected segments belonging to $\mathcal{S}$.

Figure 1.3 depicts three trajectories (T_1, T_2, and T_3) evolving in a road network composed of fourteen unidirectional road segments. The trajectories

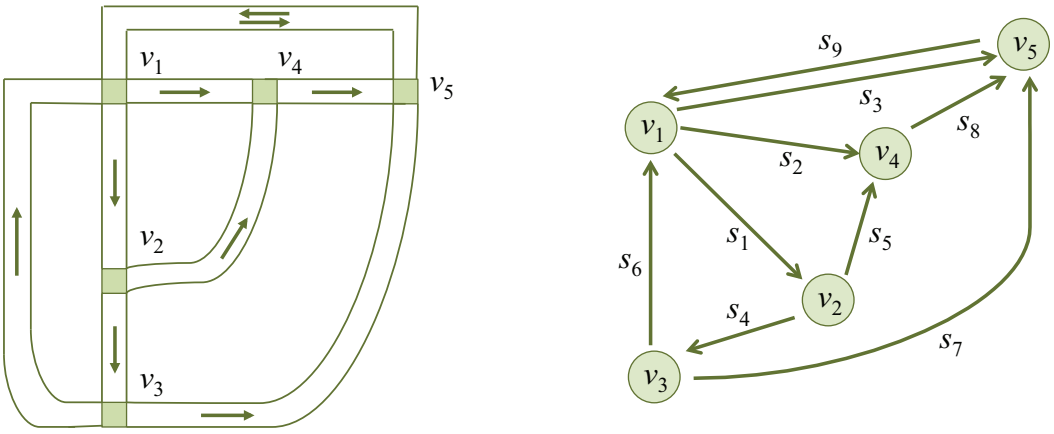


Figure 1.2 – A road network composed of five intersections and eight roads (one of which is bidirectional). The road network is represented as a directed graph where vertices represent intersections and edges represent road segments with their travel direction (therefore, both directions on the road linking v_1 and v_5 are represented with two separate edges).

are expressed in the symbolic model as follows: $T_1 = \langle s_1, s_7, s_{11}, s_{12}, s_{13} \rangle$, $T_2 = \langle s_1, s_4, s_3 \rangle$, and $T_3 = \langle s_{10}, s_{11}, s_8, s_5, s_6 \rangle$.

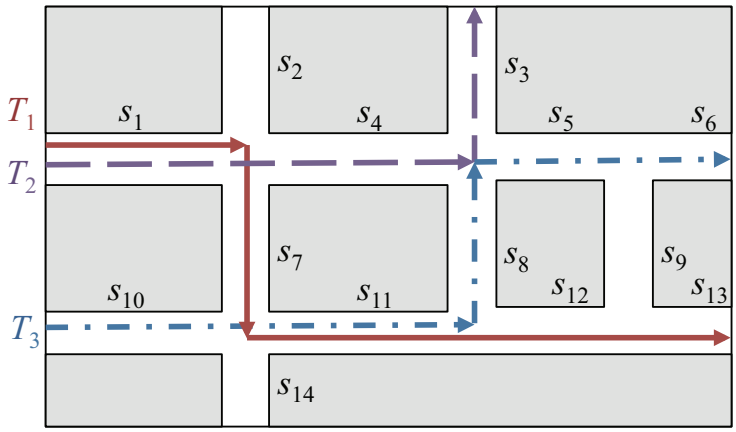


Figure 1.3 – Three trajectories traveling on a road network (we make the assumption that road segments are unidirectional). In the symbolic model as presented in Definition 3, each trajectory is represented as the series of visited road segments. Therefore: $T_1 = \langle s_1, s_7, s_{11}, s_{12}, s_{13} \rangle$, $T_2 = \langle s_1, s_4, s_3 \rangle$, and $T_3 = \langle s_{10}, s_{11}, s_8, s_5, s_6 \rangle$.

It is obvious that Definition 4 describes network-constrained trajectories from a purely spatial point of view. The definition can be extended to include time by following the proposition in [11]:

Definition 5. A spatiotemporal constrained trajectory T that travels along the road network $\mathcal{G}$ can be modeled as follows:

$$T = (id, \langle (t_1, s_1), (t_2, s_2), ..., (t_l, s_l) \rangle) \ . \tag{1.4}$$

id is the identifier of the trajectory and l its length. A couple (t_i, s_i) is registered for each segment with s_i being its identifier and t_i the date of entry[1] of T on the segment. Again $\forall 1 \leq i < l, s_i$ and s_{i+1} are connected segments belonging to $\mathcal{S}$.

Definition 5 can be further extended to include a finer granularity with the specification of the position of the trajectory on the visited segments [12, 13]:

Definition 6. A spatiotemporal constrained trajectory T that travels along the road network $\mathcal{G}$ can alternatively be modeled according to (1.5):

$$T = (id, \langle (t_1, s_1, o_1), (t_2, s_2, o_2), ..., (t_l, s_l, o_l) \rangle) \ . \tag{1.5}$$

For each triplet (t_i, s_i, o_i), t_i indicates the timestamp at which the position is registered and o_i the offset (distance) from the starting point of the segment s_i. $\forall 1 \leq i < l, s_i$ and s_{i+1} designate either the same segment or two connected segments belonging to $\mathcal{S}$.

Notice that, in reality, trajectories are acquired as GPS logs containing latitude and longitude coordinates at various dates (among other information) and are fairly easy to convert to the geometric model presented in Section 1.1.1 using projections. In order to convert these trajectories to the symbolic data model, an additional step, called map-matching [7, 9], is required in order to map the data points to the road network's graph and deduce the sequence of visited road segments. Map-matching techniques must be capable of handling trajectories efficiently and effectively even in the presence of noisy GPS data points or when the sampling rate is low. Map-matching is beyond the scope of this work.

Clustering trajectory data in road network environments is the main subject of Chapter 3 and Chapter 4 in which we will, consequently, use the symbolic data model. In these chapters, we choose to use Definition 4 to represent trajectories.

1.2 Acquiring and Generating Trajectory Data

We first discuss how trajectory data can be acquired in Section 1.2.1. In many situations, real data are not made available (e.g. due to privacy considerations when individuals are involved). In such situations, synthetic datasets can be used instead. Therefore, we briefly talk about trajectory data generation in Section 1.2.2.

[1] As represented in this model, the date at which the trajectory exits the last road segment is unknown.

1.2.1 Acquiring Trajectory Data

The most common way of harvesting trajectory data is through dedicated data acquisition campaigns that make use of "probes" (e.g. fleets of taxis, buses, or vehicles belonging to a given company; animals tagged with GPS chips; etc.). Many academic research projects used data acquired using this strategy. For example, the famous GeoPKDD (Geographic Privacy-aware Knowledge Discovery and Delivery) project [14], which was one of the first projects involving moving object trajectory data, used the Milan dataset which is composed of traces collected over a month from GPS-equipped taxis (the dataset was not made publicly available which gave the GeoPKDD project a certain edge over other work at the time). A good amount of such acquired trajectory datasets are publicly available. These include: (i) the trucks and buses datasets [15] which consist of traces of school buses and trucks in the metropolitan area of Athens[2], (ii) the GeoLife GPS trajectories dataset [16], (iii) The Cook and Dupage counties data [17], and many others.

In recent years and thanks to the democratization of ad-hoc location-aware devices, practically any kind of moving object can play the role of a probe. This includes pedestrians and cyclists equipped with smartphones and PDAs as well as any vehicle fitted with a GPS. This opens the door to a new way of collecting trajectory data by adopting a crowdsourced approach in which volunteers contribute their trajectory logs. The power of crowdsourcing can be witnessed through products like Waze [1] and OpenStreetMap [2]. Waze is a free GPS navigation tool in which the map of the road network is constructed and constantly updated based on GPS logs tracing the trajectories of volunteer users. The application constantly learns from the behavior of the users' routes and travel times in order to provide real-time traffic updates and alternative routing. Created back in 2004, OpenStreetMap is a collaborative mapping project that aims to produce a free, complete and editable map of the world. The site is nourished with the contributions of more than three hundred thousand members (as estimated in 2012) who collect GPS data and aerial photography. The data is made available for mappers who are volunteers in charge of integrating the data into the map. We believe that crowdsourcing is undeniably a very interesting way of acquiring rich and extensive trajectory data especially in contexts such as road network environments (as will be discussed in Chapter 3).

As underlined previously, GPS logs (with their multitude of formats such as NMEA, GPX, etc.) are essentially series of timestamped positions described in longitude, latitude, and (potentially) altitude, often expressed using the WGS84 world geodesic system [18]. Coordinates in this system can be easily mapped to Euclidean space using projections such as UTM (Universal Transverse Mercator) to conform with the geometric model presented in Section 1.1.1. However, in road network environments, it is more complicated to represent such data using the symbolic model (Section 1.1.2) as it requires prior knowledge of the road network's graph and the use of map-matching algorithms (which

[2]We use the trucks dataset in the experimental study presented in Chapter 2.

are often proprietary). Moreover, collecting trajectory data also gives rise to issues related to precision and quality due to imperfections inherent to GPS receivers. Such problems are out of the scope of this work in which we make the hypothesis that the data we deal with represent the ground-truth.

1.2.2 Generating Trajectory Data

In almost all domains, using real datasets for validation and experimentation is suitable. However, in some cases, such data can be simply unavailable or unusable (e.g. due to their inadequacy with the projected experimental study). This holds true in the case of analysis of trajectory data. For instance:

- Collecting data about the whereabouts of individuals poses serious privacy concerns (this matter of privacy is discussed in [4]) and projects collecting such data are rarely inclined to share them publicly.
- Analysis of trajectories in road network environments using the model in Section 1.1.2 supposes not only the availability of the trajectories of vehicles moving along the roads but also disposing of the road network's graph and a powerful and precise map matching algorithm.
- To validate the robustness and scaling capabilities of various techniques (such as indexing and access methods, clustering algorithms, etc.), large datasets are needed as smaller datasets are useless and do not allow drawing reliable conclusions.
- Etc.

Therefore, synthetic datasets generated through simulation of moving objects' behavior are viable alternatives to real data in such situations. In Chapter 3 and Chapter 4, we will study the problem of clustering trajectory data in road network environments. Due to the lack of a reliable map-matching algorithm, we will be conducting our experiments using synthetic data. Existing trajectory data generators include: (i) The Brinkhoff generator [6]; (ii) Oporto [19] for generating interacting moving objects that have few or no restrictions on motion; (iii) TSF (Traffic Simulation Framework), used for the ICDM 2010 traffic prediction for intelligent GPS navigation challenge; (iv) SUMO [20], an open-source simulation tool for urban mobility based on realistic motion models; etc. In our study, we used the Brinkhoff generator (Section 1.2.2.1) as well as a simple network-constrained trajectory clusters generator (described in Section 1.2.2.2) that we developed to answer some of our experimental needs.

1.2.2.1 Brinkhoff Generator

The Brinkhoff generator [6] was the first generator to be conceived to simulate "network-based" moving object trajectories. We give a very brief and simplified presentation of this tool in what follows.

In the Brinkhoff generator, the road network is represented using a graph much like in the model we described in Section 1.1.2. Each edge (i.e. road segment) belongs to a given class that defines its maximum authorized speed and its maximum capacity. If the number of moving objects present on the edge at a given time exceeds its maximum capacity, the maximum speed is further decreased (as to model a traffic jam). These edge classes model the characteristics of roads (highways, urban areas, rural areas, etc.) in real life. Much like edges, moving objects belong to classes defining their maximum speed and other characteristics. The generator also has the capability of generating "external objects" that can be used to model phenomena such as weather and their influence on traffic.

The first step of generating the trajectory of a moving object is to choose its departure point (vertex) in the road network's graph. This can be done with one of three strategies:

- The data-space oriented approach, in which a point (x, y) is calculated by using a distribution function over the space covered by the network and its nearest vertex is chosen as the starting point.

- The region-based approach, in which a tessellation is used to separate vertices of the network into regions and the density of each region is used to determine the starting point.

- The network-based approach, which assumes a uniform distribution in which each vertex has the same probability of being selected.

The latter strategy is a natural way of correlating the distribution of starting vertices to the network's density. The destination vertex, on the other hand, is calculated not only by applying one of the aforementioned strategies but also by accounting for preferences concerning desired route lengths and time (the behavior of the generator can be changed by rewriting parts of its code).

Once departure and destination are fixed, the route of the moving object is calculated (the route is a near-match of the fastest path) and the moving object starts traveling along it. The route is reassessed periodically and on certain triggers (e.g. detection of a drastic change in speed w.r.t. the expected speed on the traversed edge) to mimic road jams and external events. During the simulation, moving objects can report their exact geographic position periodically or they can report their positions only as vertex traversals. The latter reporting option is very useful since the symbolic counterparts of the trajectories can be deduced very easily without the need for map-matching.

We chose to use the Brinkhoff generator in our experimental studies mainly for its simplicity of use, its good performances and capacity of generating large trajectory datasets efficiently, and its popularity amongst existing work on trajectory analysis [21, 8, 11, 22].

1.2.2.2 Generating Network-Constrained Trajectory Clusters

In Chapter 3 and Chapter 4, we study the problem of clustering network-constrained trajectory data in which we are interested, among other things, in discovering groups of similar trajectories that moved on the same parts of a given road network. We desired to test the approaches we propose in this context on labeled datasets in which such groups are pre-established and known in advance. The standard behavior of the Brinkhoff generator is to generate each trajectory "independently" of the others (i.e. the start and end vertices for each moving object are fixed with the strategies we presented in Section 1.2.2.1 while disregarding the start and end vertices of the other objects). This means that no such prior classes are known. Consequently, we imagined and implemented a basic and simple strategy for generating network-constrained trajectory clusters. The main idea behind this strategy is the following. We desire to have a dataset composed of groups of trajectories where each group travels from one area in the road network to another different area while following more or less the same route. We also desire for these generated groups to present some kind of patterns such as to depict real life scenarios (e.g. two or more groups of individuals converge to the same destination to attend a sports event, commutes of a group of people in-between a residential area and an industrial area, etc.).

The road network in our generator is represented as a directed graph using the symbolic model we presented earlier. Since we use the same `*.node` and `*.edge` network description files provided with the Brinkhoff generator, we also integrate the characteristics of the road segments (edges) in order to handle routing.

In order to generate a trajectory cluster, a set of vertices that will be used as the start area of the cluster and a set of vertices to be used as its destination area must be fixed. In order to define the set of start vertices, a vertex in the road network's graph is drawn randomly (similarly to the network-based strategy in the Brinkhoff generator). This vertex plays the role of a "kernel" to build the set of start vertices: all vertices that are reachable[3] from the kernel within a user-defined maximum distance are fetched and included in the start set. The set of destination vertices is built in the same manner with the exception that the user can define a "desired" minimum length for the routes in the cluster. In this case, the generator repeatedly draws the kernel vertex of the destination set until the shortest path between both the start kernel and the destination kernel exceeds the threshold.

Once both the start and destination sets are determined, the cluster can be generated. For each trajectory in the cluster, a vertex is randomly drawn from the start (resp. destination) vertex set and is chosen to be its departure (resp. arrival) vertex. The trajectory is then generated as the set of road segments corresponding to the shortest commute time (calculated using the

[3]Reachability here uses shortest path calculations based on the spatial length of the road segments.

spatial length and maximum speed properties of the edges) between the two vertices. The number of trajectories within clusters is randomly fixed between two bounds defined by the user.

Besides generating simple clusters, our generator is also programmed to produce the following patterns:

- Inverted clusters, which are two clusters where the start vertex set of one cluster is the end vertex set of the other (Figure 1.4). This pattern depicts behaviors such as vehicles moving from a residential area to an industrial area in the morning then going back in the evening.
- Converging clusters, which are clusters that depart from different start areas and converge to a common destination (Figure 1.5). This pattern depicts flocking to an area of interest (on special occasions such as attending a sport event, etc.).
- Diverging clusters, in which groups depart from the same area then diverge into separate paths leading to different areas (Figure 1.6). Among real scenarios where this pattern can be observed is the situation when different groups of vehicles return to their respective residential areas after having attended a given event.

(a) (b)

Figure 1.4 – Example of two inverted clusters of trajectories. The departure area of one cluster (depicted in green) is the destination area of the other cluster (in red) and vice versa.

While the aforementioned patterns are intentionally created by the generator, they also do occur naturally along with other types of patterns. For example, Figure 1.7 depicts a situation in which a combination of the converging clusters and diverging clusters occurs simultaneously as two groups of trajectories depart

(a) (b)

Figure 1.5 – Example of two converging clusters of trajectories. Here, the two groups start at different areas and converge to a common destination while potentially partially sharing their routes.

(a) (b)

Figure 1.6 – Example of two diverging clusters of trajectories. Here, the two groups start at the same part of the road network but diverge to separate destinations.

from different areas, join routes together then diverge into their separate ways to reach their destinations.

(a) (b)

Figure 1.7 – Example of two trajectory clusters that start at different parts of the road network, converge and share a portion of their route then separate and diverge to different areas.

As underlined before, the main objective behind this basic tool is to be able to test the performances of our approaches in simple yet non trivial scenarios and assess their capabilities to identify the presence of interacting groups of trajectories. Datasets produced with the help of the strategy we just described will be used in Chapter 3 and Chapter 4.

1.3 Conclusions

In this chapter, we introduced the data models that we will use to represent trajectory data all along this thesis. We also briefly discussed some examples of how trajectory data can normally be collected in real life. Given that in parts of this work we will have recourse to synthetic datasets, we presented the two generators that we used to produce the concerned data.

We present the contributions of this thesis starting from the next chapter where we explore the problem of sampling trajectory data in streaming environments in order to reduce their size and facilitate their storage.

Chapter 2

Sampling Moving Object Trajectory Streams

In this chapter, we address the problem of moving object trajectory sampling. Current location-aware devices are capable of calculating and reporting their position at a very high rate (up to one data point per second). However, when interested in continuously harvesting the trajectories of thousands (and even hundreds of thousands) of moving objects, keeping a permanent record of every single one of these positions can lead to undesirable overheads both storage-wise (requiring potentially infinite storage) and computationally (as the very large volume of data can seriously hinder visualization and mining tasks conducted afterwards). Therefore, compression techniques (such as wavelets, histograms, etc.) are needed in order to reduce the size of the data. Sampling is a natural and intuitive technique that is often used to this effect. By discarding inherently redundant data points (e.g. a vehicle standing by at a red light and reporting the same position), trajectory sampling can help shrink the volume of the stored data by reasonably trading off precision for conciseness.

We are interested in the trajectory sampling problem in the particular context of data streaming environments. In such systems, data arrive continuously as an infinite, ordered sequence of items (an item, here, being a new position reported by a given moving object). Consequently, processing must be performed on-the-fly and the use of blocking offline algorithms (i.e. algorithms that require the entirety of the data to be available) is impossible. Additionally, the used algorithms must have low time complexities in order to be able to withstand the high load of incoming data.

The remainder of this chapter is structured as follows. First, we state our formulation of the moving object trajectory sampling problem and we break down its objectives in Section 2.1. In Section 2.2, we give a brief introduction to data stream processing and we review existing trajectory-dedicated sampling techniques. Our contribution is detailed in Section 2.3 where we present our SpatioTemporal Stream Sampling (STSS) algorithm. An experimental study is provided in Section 2.4. Finally, conclusions are drawn in Section 2.5.

2.1 Problem Statement

Trajectory data are acquired from moving objects in the form of a series of discrete GPS coordinates (commonly in the WGS84 world geodesic system [18] which can be projected into an Euclidean space afterwards). As such, the geometric model (presented in Section 1.1.1) is more intuitive to represent such trajectories. Consequently, we consider raw trajectories that are represented in compliance with Definition 1 (page 8).

As already mentioned in Section 1.1.1, the more location points available, the more accurately the trajectory can be interpolated. Unfortunately, due to storage and computational limitations, keeping the integrity of the positions of all the monitored moving objects, even if feasible, is impractical. Hence, trajectory compression is needed in order to reduce the size of stored trajectories. Informally speaking, compressing a trajectory means replacing its original series of location points with another, more compact one. We formally define a compressed trajectory as follows.

Definition 7 (Compressed Trajectory)**.** A compressed (or sampled) trajectory T_{C} of a trajectory T is a subset of the series of original points forming T. T_{C} is obtained through down-sampling of T by discarding unnecessary points and yet preserving its essential movement features. It also must cover the integrity of the original trajectory from start to finish (i.e. it must imperatively include the first and last positions in T). No artificial new points are to be created during this compression process.

We interchangeably use both the terms compression and sampling to designate this process of reducing the number of points needed to represent a given trajectory.

The objectives of this compression process are, as stated by Meratnia and de By [23]:

- To obtain a lasting reduction in data size.
- To obtain a data series that still allows various computations at acceptable (low) complexity.
- To obtain a data series with known, small margins of error, which are preferably parametrically adjustable.

Reducing the size of the data helps speed up various time-consuming computations such as analytical and mining tasks. Since sampling is a lossy compression technique, however, this data reduction comes at the cost of introducing further approximation errors into the data (as it already contains some due to GPS inaccuracies). Therefore, it is advisable to use algorithms that can control these errors (by giving some error bound guarantees depending on their parameters). This way, the error can be controlled depending on the precision required for the application domain at hand (e.g. a fleet management

application can decide to tolerate an inaccuracy of up to 100m while an application for monitoring animal migration may deem that a 1km error is tolerable).

In addition to the aforementioned objectives, we define the following requirements which are inherently related to processing data streams:

- Online processing: compression algorithms can be classified as either offline (batch) or online (incremental) algorithms based on whether they require the availability of the whole data series or not. Offline algorithms do, online algorithms don't. Only the latter can be used to compress data streams on-the-fly, in real-time as they arrive.
- Low computational complexity: single-pass algorithms where each new incoming data point is processed only once and in constant time are the most adequate (with regard to time) to compress very large data streams.
- Low in-memory complexity: which designates the amount of memory used by the algorithm when it's executed and can be perceived as the size of the window that the algorithm maintains over the data stream. Preferably, only a small set of data is kept in memory.

In the next section, we give a brief presentation of data stream processing and we present a survey of existing work on moving object trajectory sampling.

2.2 Related Work

We first give a general presentation of data stream processing (Section 2.2.1) then we discuss related work on trajectory sampling (Section 2.2.2).

2.2.1 Data Stream Processing

In [24], Golab and Özsu define a data stream as a structured, infinite, and ordered series of items. The structure of the stream is described through a schema detailing for each attribute its name and type. The main difference between these items and tuples stored in a classic database is this concept of order, usually made explicit through a timestamp or date attribute.

Data streams are widely adopted in many domains where large volumes of data are generated and exchanged continuously (e.g. telecommunications, financial analysis, sensor networks, etc.). Simply put, Data Stream Management Systems (DSMS) are systems capable of handling multiple data streams while disposing of limited resources w.r.t. CPU time and in-memory storage. Whether these DSMS are generic (e.g. STREAM [25], TelegraphCQ [26], and Aurora [27]) or tailored for a specific need (e.g. Gigascope [28] and NiagaraCQ [29]), they represent a shift from the classic data-pull paradigm adopted by Database Management Systems to a data-push paradigm.

In the classic data-pull model, data are stored permanently on disk and queries are formulated and executed only once (i.e. punctually in time). The result of a query is immediately calculated based on the whole stored information and returned to the user. By contrast, in the data-push model, data are "pushed" into the system from various sources (e.g. sensors, GPS devices, etc.). Once defined, queries are run continuously for a given period of time on the arriving data. Therefore, the answers to these continuous queries [30] may vary over their lifespan as new items arrive. Compared to classic queries, continuous queries present a number of particularities mainly inherent to the dynamic nature of data streams [24, 31, 32]:

- Due to the high arrival rate of data streams, queries must be processed in real-time (on-the-fly). Backtracking over the data is generally impossible and each element is processed only once in a single pass (unless temporary storage is considered).
- A buffer can be allocated in order to temporarily store some elements of the stream. However, the space intended to this effect must be limited.
- Due to the infinite nature of streams, blocking operators (i.e. operators, such as joins, that require the presence of all the data) are unfeasible. Instead, such operators can use a limited window that covers the recent elements of the stream.
- Due to the fluctuations and high arrival rates of streams, load shedding mechanisms must be provided in order to protect the system's performance and prevent it from degradation.

As previously mentioned, once an item is processed, it's either discarded immediately or retained in a window for a limited amount of time. This inability to store the entirety of a stream suggests the use of approximate summary structures [33] (also called synopses). Summaries are constructed using data reduction techniques in order to store a concise, approximate representation of the content of the stream. These summaries can be used: (i) to answer newly defined queries concerning the old parts of the stream, (ii) to give approximate answers to queries that include blocking operators such as joins, (iii) to conduct data mining tasks that require a historical view that spans over a considerable amount of time, etc.

Existing summary techniques include sketches [34, 35, 36], histograms [37, 38], wavelets [39], etc. Among these, sampling techniques are the most intuitive and easy to grasp. In [40], Vitter presents a random sampling technique using a reservoir that can be applied to data streams. In order to construct a sample containing k data points, the approach starts by automatically inserting the k first elements of the stream into the sample. Then, for each element that has the order i in the stream, the element is inserted in the sample with a probability $\frac{k}{i+1}$ (in which case a randomly selected element is removed from

the sample). Another stream sampling technique is the StreamSamp algorithm [41]. StreamSamp uses a combination of random sampling and tilted windows to construct a summary where the quality of the data decays over time. At their arrival, new elements are inserted into a sample of order 0 (the one with the highest quality) based on a fixed random sampling rate α. Once all the samples of a given order l are filled (the number of samples per order as well as their size are user-defined), the oldest two samples of l are resampled and merged in a single sample and moved to the order $l + 1$. In this way, as the stream continues flowing, its old elements are represented with fewer samples of lesser quality.

These sampling techniques are, however, generic techniques that can be applied to streams of different natures. In this regard, they are insensitive to the spatiotemporal properties of trajectory data and are consequently ineffective for sampling such streams. In the following section, we review some approaches that were specifically tailored to spatiotemporal trajectory sampling.

2.2.2 Moving Object Trajectory Sampling

Existing work in trajectory compression is mainly inspired by advances in the fields of line simplification, cartographic generalization, and time series compression. As already mentioned in Section 2.1, the proposed compression techniques can be classified either as: (i) online techniques, capable of processing trajectories in real-time as they are streamed into the system; or (ii) offline (batch) techniques that require all the data points of a trajectory to be present in order to compress it.

Another classification can be established based on the operating mode of these techniques. Under this perspective, a given trajectory compression approach can fall under one of the following classes [23]:

- Top-Down approaches: the whole set of data points composing the trajectory is recursively partitioned until a given condition is met. These are offline approaches by nature.

- Bottom-Up approaches: neighbor individual data points are merged together until a given criterion is met.

- Windows-based approaches: a window is moved along the data points of the trajectory (from one end to the other) and the content of this window is compressed.

Generally, the halting condition is either: (i) error-related (e.g. the approximation error for one of the points exceeded a user-defined value), or (ii) space-related (e.g. the number of retained points reaches a given threshold).

In the following, we study some of the representatives of each of the aforementioned categories.

2.2.2.1 Top-Down Approaches

In [42, 23], Meratnia and de By present two top-down algorithms for trajectory compression. Both algorithms are based on the Douglas-Peuker (DP) algorithm [43]. The DP algorithm (depicted in Figure 2.1) is a linear simplification technique that considers a whole trajectory and tries to construct a simplified version of it by retaining only a subset of its original points. Initially, the algorithm considers the compressed trajectory formed by a single segment linking the first and last points of the original trajectory. For each intermediate point, its distance to its orthogonal projection on the compressed trajectory is calculated. If the maximum distance is superior to a user-defined threshold, then the point that caused the maximum distance is inserted into the compressed trajectory. The algorithm carries on recursively on each of the two parts of the compressed trajectory (i.e. the segment formed by the first trajectory point and the newly retained point and the segment formed by the newly retained point and the last trajectory point) until the distance threshold is no longer exceeded by any of the unretained points.

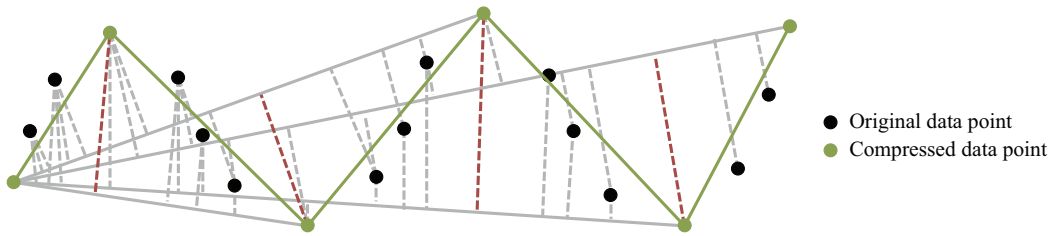


Figure 2.1 – The Douglas-Peuker algorithm [43]. Figure adapted from [23].

The first variation presented by Meratnia and de By, called Top-Down Speed-Based (TD-SP), replaces the distance to the orthogonal projection (a purely spatial criterion) with a speed-based criterion: the retained point is the point where the highest speed variation was observed (which corresponds to a drastic acceleration or deceleration). The speed variation is calculated for a point P_i based on its previous and following points using Formula (2.1).

$$\text{Speed variation}(P_i) = \left| \frac{\text{distance}(P_{i+1}, P_i)}{t_{i+1} - t_i} - \frac{\text{distance}(P_i, P_{i-1})}{t_i - t_{i-1}} \right| . \qquad (2.1)$$

For the second variation, named Top-Down Time-Ratio (TD-TR), the authors introduce the concept of Synchronous Euclidean Distance (SED). Instead of using P_i's orthogonal projection on a segment $P_\text{s}P_\text{e}$, the timestamp of P_i is used to calculate a time-aware estimated position $P'_i(x'_i, y'_i)$ according to Formula (2.2). The SED corresponds to the distance between P_i and its estimation P'_i as depicted in Figure 2.2.

$$
\begin{aligned}
x'_i &= x_\mathrm{s} + \frac{t_i - t_\mathrm{s}}{t_\mathrm{e} - t_\mathrm{s}} \cdot (x_\mathrm{e} - x_\mathrm{s}) \ ; \\
y'_i &= y_\mathrm{s} + \frac{t_i - t_\mathrm{s}}{t_\mathrm{e} - t_\mathrm{s}} \cdot (y_\mathrm{e} - y_\mathrm{s}) \ .
\end{aligned}
\tag{2.2}
$$

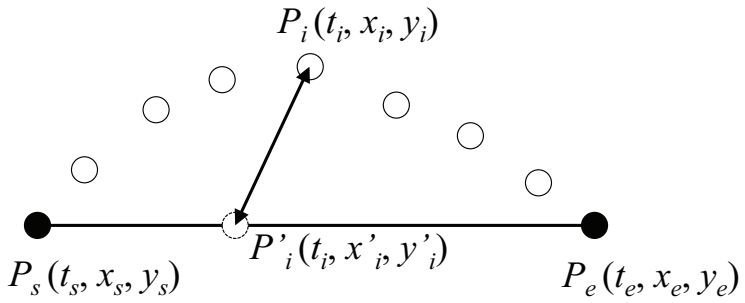


Figure 2.2 – The Synchronous Euclidean Distance. Figure adapted from [23].

The aforementioned algorithms have a time complexity of $O(n^2)$ where n is the number of data points in the original trajectory (this complexity can be reduced to $O(n \log n)$ as discussed in [44]). They require a $O(n)$ in-memory space since the whole trajectory must be maintained in memory. They also offer a guaranteed error bound (which varies depending on the retained variation). However, since these are offline approaches, they cannot be used for real-time processing and sampling of trajectory streams.

2.2.2.2 Opening-Window Approaches

Another approach to sampling trajectory data is by means of window-based techniques. The opening-window approach (OPW) is one of these techniques. An opening window keeps growing and accepting new data points until a given criterion is met. At this point the content of the window is compressed and the process is resumed from the last seen point. The OPW algorithm starts by considering the segment defined by the first point in the trajectory (called the anchor) and its third point (called the float). As long as the distance between all the intermediate points and their orthogonal projections on the segment are below a user-defined distance threshold, the algorithm tries to advance the float further to the next point in the series (the distances are re-evaluated each time). When the threshold is exceeded, the window's content is compressed using one of two strategies:

- Normal Opening Window (NOPW): the point causing the violation of the threshold is inserted into the sample (Figure 2.3).
- Before Opening Window (BOPW): the point preceding the float is inserted into the sample (Figure 2.4).

In both cases, the point inserted into the sample becomes the new anchor and the process is resumed.

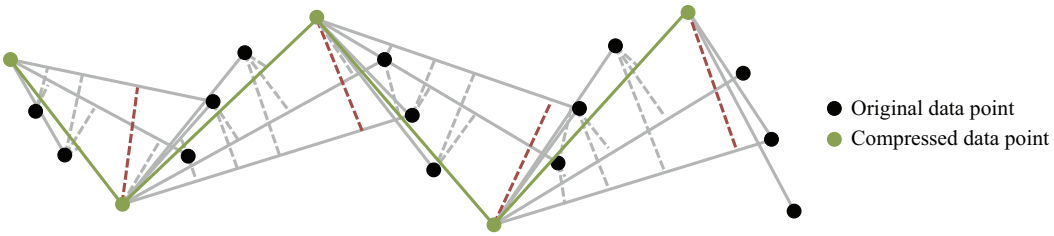


Figure 2.3 – Normal Opening Window compression. The first window opened up to the 6^{th} point, at which point the 4^{th} point (the one exceeding the threshold) is inserted into the sample. The second window opened up to the 10^{th} point and the 8^{th} point was inserted in the sample. The algorithm continues in the same fashion and also inserts the 12^{th} and 16^{th} points. Figure adapted from [23].

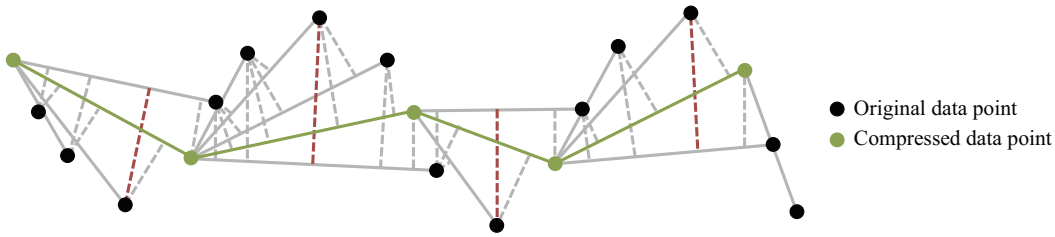


Figure 2.4 – Before Opening Window compression. The first window also opened up to the 6^{th} point (the 4^{th} point violating the threshold) and the 5^{th} point (the one before the float) is inserted into the sample. The second window opened up to the 11^{th} point and the 10^{th} point was inserted into the sample, etc. Figure adapted from [23].

Meratnia and de By [42, 23] enhance the OPW technique using the same distances they defined for TD-TR and TD-SP. Their OPW-SP technique uses the speed variation as an indicator of when to compress the content of the window, whereas their OPW-TR technique uses the SED distance instead of orthogonal projections.

Opening window techniques are computationally intensive as they have a time complexity of $O(n^2)$ to compress a trajectory containing n points. Moreover, in worst case scenarios (which occur in situations where none of the trajectory's data points causes a violation of the threshold) the window can open up to include the whole trajectory, which requires an in-memory space of $O(n)$. OPW-TR and OPW-SP provide the user with guarantees and control over the approximation errors. They are online algorithms that are perfectly capable of sampling trajectories on-the-fly and can be used in a streaming

context under the condition that their in-memory space requirement can be afforded.

2.2.2.3 Bottom-Up Approaches

The bottom-up family of algorithms includes, among others, the thresholds algorithms [45], STTrace [45], and the AmTree approach [46].

The threshold approaches [45] are a family of predictive heuristics for sampling trajectories. Given a moving object, they try to predict its future positions based on the previously seen ones. The positions that the algorithms predict successfully are considered as redundant and are discarded, whereas those that the algorithms fail to predict are retained in the sampled trajectory. The prediction is based on two thresholds: the speed threshold (dv, expressed in percentage) and the orientation threshold ($d\varphi$, expressed in degrees).

The Sample-Based Threshold algorithm bases its prediction on the last two points inserted into the sample (Figure 2.5). These points, the timestamp of the point that the algorithm is trying to predict, and both dv and $d\varphi$ are all used by the algorithm to determine a "safe area." If the inspected point is outside of this area, then it's considered as miss-predicted and is consequently inserted into the sample.

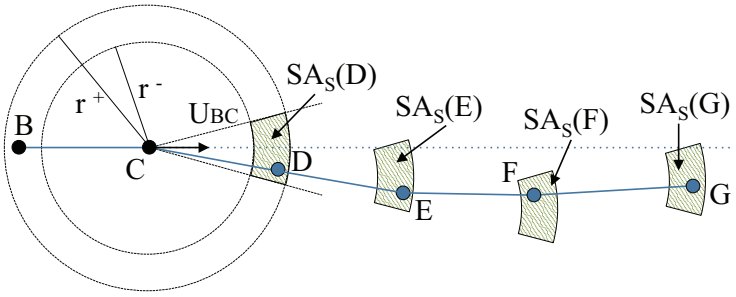


Figure 2.5 – Sample-Based Threshold compression. Using the last two points in the sample (B and C) and the user-provided thresholds (dv and $d\varphi$), a safe area ($\mathrm{SA_S}(D)$) is calculated for the inspected point D. If the point is within the perimeters of this area, then it is considered as well-predicted and is discarded. Otherwise, the point is inserted into the sample. Figure adapted from [45].

The Trajectory-Based Threshold algorithm (Figure 2.6) applies the exact same principle but using the last two positions in the real trajectory (instead of those in the sample) to conduct the prediction.

Both variations are prone to error propagation as the moving object can drift significantly without the algorithms being able to detect it (this situation is depicted in Figure 2.6). Therefore, a third approach, called Combined-Threshold, is proposed. This approach calculates two confidence zones (one based on the sample, the other on the trajectory) and a position is considered as well-predicted only if it belongs to the intersection of both zones.

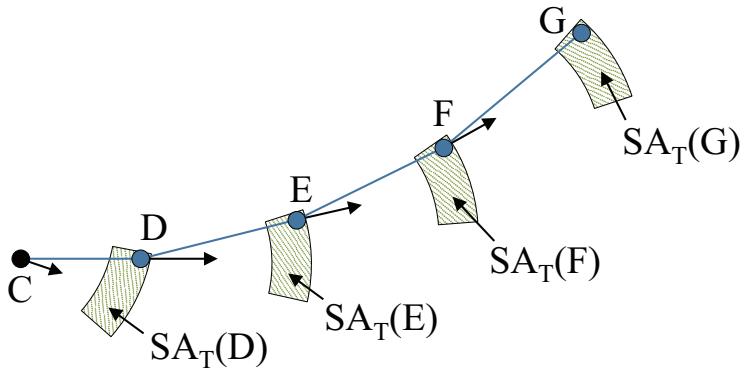


Figure 2.6 – Trajectory-Based Threshold compression. Here, the prediction uses the last two points in the real trajectory instead of the sample. This, however, makes the approach vulnerable to error propagation as the moving object can drift slowly without being noticed. Figure adapted from [45].

The threshold algorithms are online-capable and are very efficient as they can process each new position on-the-fly in $O(1)$ time, thus yielding a $O(n)$ complexity for compressing a trajectory containing n data points. However, one of their major drawbacks is their lack of approximation error guarantees which makes them unreliable w.r.t. compression quality. Furthermore, the configuration of the two thresholds dv and $d\varphi$ can be problematic as sampling results can vary significantly depending even on slight changes of their values.

The STTrace algorithm (also presented in [45]) attempts to address the requirement for limited storage space as it tries to build a sample containing at most M data points (M being user-defined) of the trajectory to which it is applied. The idea is quite intuitive: the first M points of the trajectory are immediately inserted into the sample. For each point in the sample, its SED is calculated based on its immediate neighbors. Afterwards, when a new point arrives, a probing SED is calculated for it (based on the last two points in the sample) and compared to the minimum SED from the points in the sample (i.e. the SED of the point that, if deleted, causes the least distortion in the sample). If the probing SED exceeds the minimum SED in the sample, then the point with the minimum SED is deleted and the new point is inserted into the sample instead. STTrace is capable of online processing and has a time complexity of $O(\frac{1}{n}\log(\frac{n}{M}\log M))$ per incoming position (cf. [45] for the proof). But like the thresholds algorithms, it suffers from the lack of approximation error guarantees.

The AmTree [46] is based on the insightful idea of amnesic storage [47] and aims to keep a sample of a given trajectory where the quality decays over time (i.e. as the data points get older, they are represented with lesser details). AmTree is basically a tree structure that is built and maintained in a bottom-up fashion. At their arrival, the new data points are inserted into the lower level of the tree and, as insertions go on, the oldest points are merged together and

moved to the upper level of the tree (potentially triggering further updates that can span up to the highest level of the tree). The merging, however, is conducted naively and is not based on the spatiotemporal features of the merged data points. The AmTree can be used online and can be updated rapidly (in $O(1)$ per new point) but does not guarantee an error bound.

2.2.2.4 Synthesis

The characteristics of the aforementioned techniques (except for the AmTree, which we exclude for being a naive approach) w.r.t. the criteria we defined in Section 2.1 are summarized in Table 2.1. From this table, we can observe the following trends:

- Algorithms (whether offline or online) that offer guaranteed error bounds tend to have high computational and in-memory costs.
- Efficient algorithms (w.r.t. time and in-memory space) do not generally offer guaranteed error bounds. Therefore, the resulting compressed trajectories can be unreliable and can produce erroneous results when queried.
- A duality exists between storage space and compression errors. The algorithms that control the compression errors do not control the space that is needed to store the sampled trajectory (which will depend essentially on the complexity of the movement pattern exhibited by the moving object) and vice versa. Therefore, the trade-off between the sample's quality and its size is inevitable.

Table 2.1 – Characteristics of different trajectory compression techniques.

Technique	Online	Error bound	Time complexity	In-memory complexity	Control over storage space
TD-TR, TD-SP [23]	no	yes	$O(n^2)$	$O(n)$	no
OPW-TR, OPW-SP [23]	yes	yes	$O(n^2)$	$O(n)$	no
Thresholds [45]	yes	no	$O(n)$	$O(1)$	no
STTrace [45]	yes	no	$O(\log \frac{n}{M} \log M)$	$O(1)$	yes

In the following section, we address the first two observations by proposing the STSS algorithm: a computationally and in-memory efficient algorithm that also provides a guaranteed and configurable compression error bound.

2.3 The SpatioTemporal Stream Sampling Algorithm

We now present the SpatioTemporal Stream Sampling (STSS) algorithm, an online sampling technique tailored for compressing streamed moving object trajectories based on both their spatial and temporal features.

2.3.1 The STSS Algorithm

STSS is based on the intuitive idea of linear prediction (similarly to the threshold family of algorithms): the algorithm tries to capture the currently observed motion pattern of the moving object and uses it to predict its future positions. As long as these positions are predicted correctly, they can be safely excluded from the compressed trajectory.

In order to keep track of the current motion trend and predict forthcoming positions, STSS calculates and keeps up-to-date a simple function (called the motion function) using two data points, say $P_k = (t_k, x_k, y_k)$ and $P_j = (t_j, x_j, y_j)$ with $t_k < t_j$, that are deemed to be representatives of the current behavior of the moving object. For a given ensuing time instant t ($t > t_j$), this motion function yields a predicted position calculated as follows (Formula (2.3)):

$$(x, y) = (a_x t + b_x, a_y t + b_y) \ . \tag{2.3}$$

The slopes (a_x and a_y) and intercepts (b_x and b_y) on both the X and Y axes are calculated using P_k and P_j conformally to Formula (2.4):

$$\begin{aligned} a_x &= \frac{x_j - x_k}{t_j - t_k} \ , \\ b_x &= x_k - a_x t_k = x_j - a_x t_j \ , \\ a_y &= \frac{y_j - y_k}{t_j - t_k} \ , \\ b_y &= y_k - a_y t_k = y_j - a_y t_j \ . \end{aligned} \tag{2.4}$$

In reality, the motion function is an extrapolation of the Synchronous Euclidean Distance. While the latter is defined only for data points that belong to the segment $P_k P_j$, the former is used for time instants that come after (i.e. for data points subsequent to P_j).

Initially, the motion function is determined from the first two points in the streamed trajectory and the first point is immediately inserted into the sampled trajectory. For each new data point P_i, a predicted point P_i' is calculated by applying the motion function to its timestamp. If the distance between P_i and P_i' is below a user-defined threshold d_{Thres}, then P_i is considered as well-predicted. On the other hand, a miss-prediction occurs when the distance between a point P_i and its prediction exceeds d_{Thres}. In this situation, the motion function is reassessed using P_i and its predecessor P_{i-1}.

The sampling strategy is quite simple: the first two points are inserted into the sample. Thereafter, a correctly-predicted point replaces the last point inserted into the sample (the latter can be discarded safely with a deterministic error bound guarantee as will be discussed later in Section 2.3.2). On the other hand, a miss-predicted point is appended to the sample without replacing its last point. This sampling strategy is done incrementally in a single pass and insures that, even while the trajectory is still being streamed, the sample is up-to-date and reflects the state of the entirety of the points seen so far (i.e. it covers the trajectory from its first point up to the last seen one). The pseudo-code of STSS is shown in Algorithm 1.

Algorithm 1 SpatioTemporal Stream Sampling

Input: a streamed trajectory T, a distance threshold d_{Thres}.
Output: the compressed (down-sampled) trajectory T_{C}.

```
1: insert P_1 into T_C    ▷ immediately insert the first position into the sample
2: initialize the motion function using P_1 and P_2
3: insert P_2 in T_C
4: for all incoming position P_i = (t_i, x_i, y_i) ∈ T do          ▷ with i > 2
5:     calculate P'_i                                              ▷ predicted position
6:     if distance(P_i, P'_i) > d_Thres then                       ▷ miss-prediction
7:         recalculate the motion function using P_{i−1} and P_i
8:         insert P_i into T_C
9:     else
10:        replace the last position in T_C with P_i
11:    end if
12: end for
```

Figure 2.7 showcases how STSS works. In order for the compressed trajectory T_{C} to cover the entirety of the original trajectory, the first point P_1 is stored immediately. Once the second data point P_2 arrives, a first motion function is calculated. P_2 is also stored in the compressed trajectory. Upon arrival of P_3, its prediction P'_3 is calculated. Since the point is well-predicted, it replaces the last point of the compressed trajectory. Therefore, at this point: $T_{\text{C}} = \langle P_1, P_3 \rangle$. Since P_4 is also well-predicted, it replaces P_3 in T_{C}. The same goes for P_5 which, once it arrives, takes the place of P_4. P_6, on the other hand, is miss-predicted, which signals a change in the motion pattern. The motion function is updated accordingly (using P_5 and P_6) and P_6 is inserted into T_{C}. Up to this point: $T_{\text{C}} = \langle P_1, P_5, P_6 \rangle$. The process carries on in the same fashion for P_7 and P_8 which are both well-predicted and P_9 which (being miss-predicted) triggers a second update of the motion function (using P_8 and P_9 this time). A trajectory (sampled at a rate of one position per second) as well as its STSS compressed counterparts for different values of the distance threshold d_{Thres} are shown in Figure 2.8.

The time and in-memory complexities of STSS are stated in the following properties:

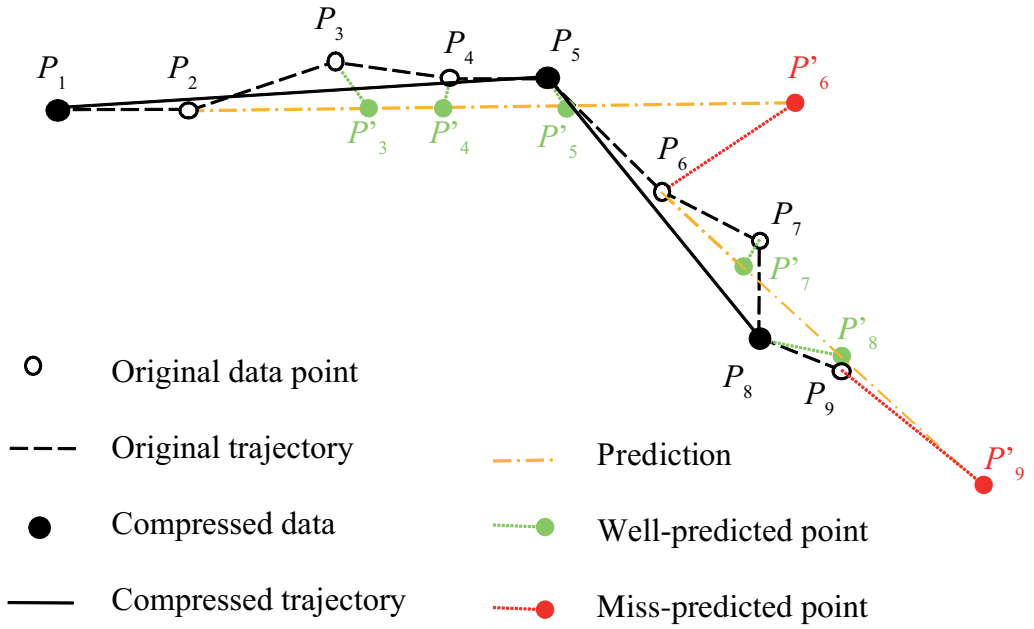


Figure 2.7 – Example of the execution of the STSS algorithm.

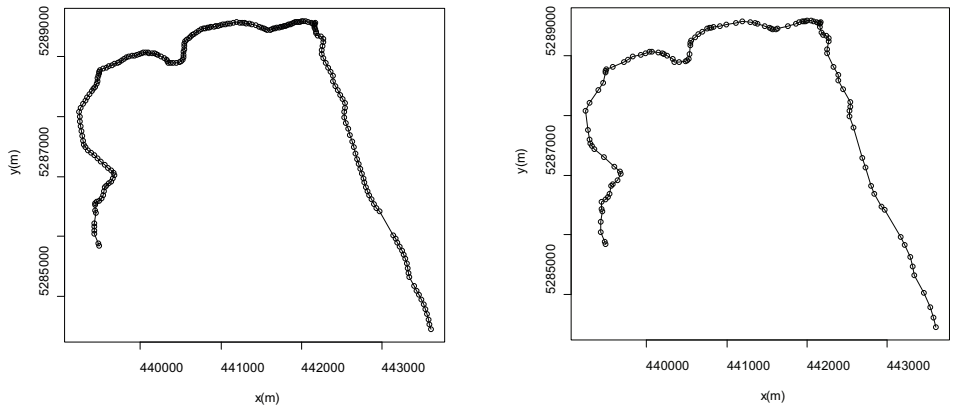
Property 1 (Time Complexity)**.** The time complexity of STSS is $O(1)$ per incoming position. Hence, a trajectory containing n points is processed in $O(n)$ time.

Property 2 (In-Memory Space Requirement)**.** For each moving object, only the latest position is kept in memory and is either stored on disk or pruned as soon as the next position is acquired. Thus, the in-memory space requirement for sampling each trajectory is $O(1)$.

The size of the obtained sampled trajectory is not configurable. This is due to the fact that while the parameter of the algorithm has an influence on the size (increasing the value of d_{Thres} results in smaller samples), the latter mainly depends on the complexity of the original trajectory: a trajectory with frequent abrupt changes in direction and a simpler, smoother trajectory containing the same number of points will not produce samples of equal size if sampled with the same value of d_{Thres}.

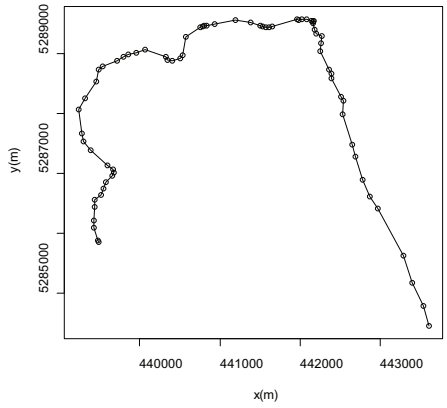
2.3.2 Sampling Error Bound

One of the attractive features of the STSS algorithm is the fact that it provides a guaranteed, theoretically-proven error bound. Moreover, this error bound is directly adjustable by the user since it's inherently related to d_{Thres} (the distance threshold used to control the sampling process). The error bound guarantee is stated as follows:

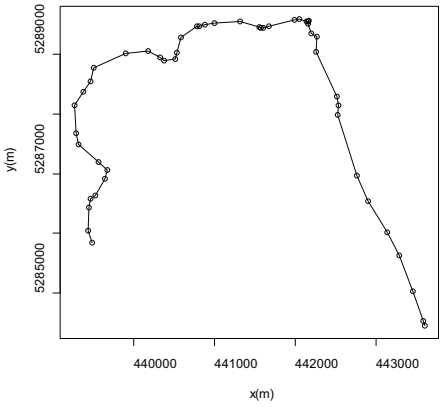


(a) Original trajectory (228 data points)

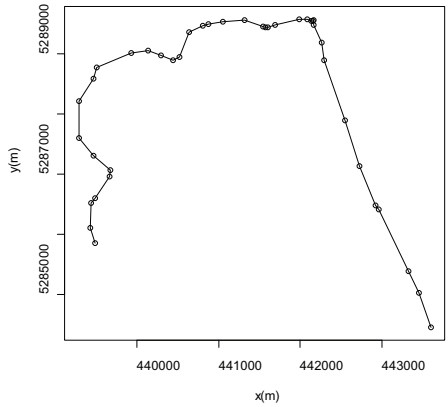
(b) $d_{\text{Thres}} = 5\text{m}$ (117 data points)



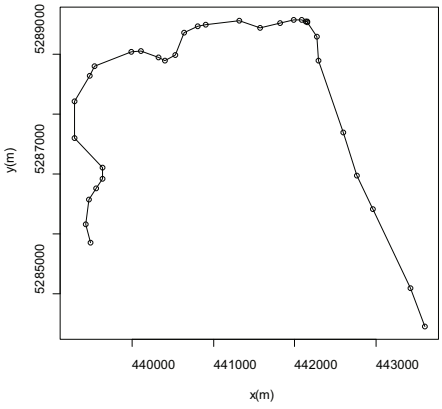
(c) $d_{\text{Thres}} = 25\text{m}$ (72 data points)



(d) $d_{\text{Thres}} = 50\text{m}$ (49 data points)



(e) $d_{\text{Thres}} = 75\text{m}$ (40 data points)



(f) $d_{\text{Thres}} = 100\text{m}$ (32 data points)

Figure 2.8 – A trajectory and its sampled counterparts for different values of d_{Thres}.

Property 3 (Error Bound)**.** The STSS algorithm guarantees a deterministic error bound. For a given distance threshold d_{Thres}, the maximum approximation error resulting from the compression is equal to $2d_{\text{Thres}}$.

Proof. Consider the situation depicted in Figure 2.9. P_s and P_2 were used to calculate the motion function. P_e is the last well-predicted point (i.e. the point immediately after it was miss-predicted). Consequently, the set of points in-between P_s and P_e were all pruned from the sample.

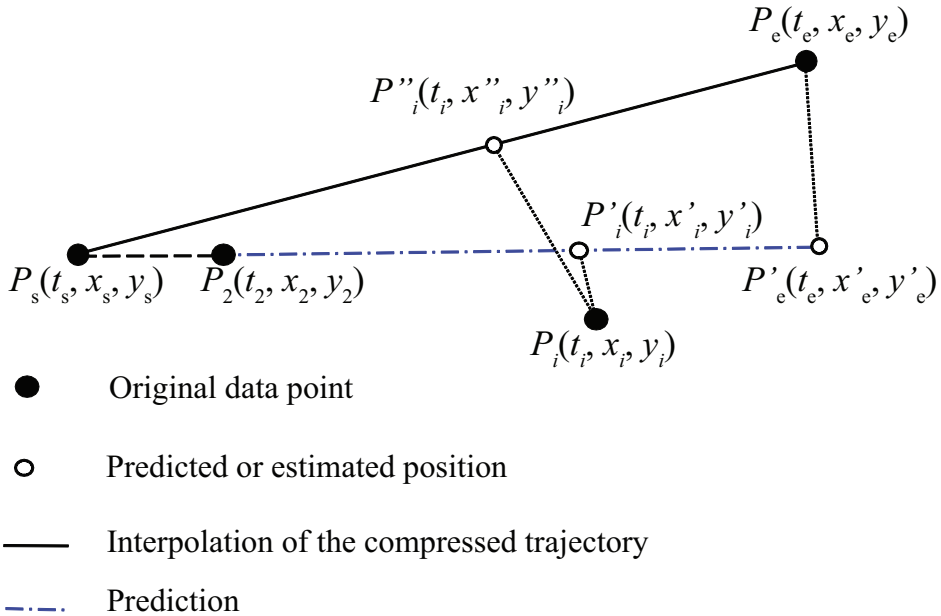


Figure 2.9 – Proof of the error bound.

For each intermediate point P_i, its prediction P'_i is expressed as follows (2.5):

$$P'_i(t_i, x'_i, y'_i) = \left(t_i, \frac{(x'_e - x_s)t_i + x_s t_e - x'_e t_s}{t_e - t_s}, \frac{(y'_e - y_s)t_i + y_s t_e - y'_e t_s}{t_e - t_s} \right) . \tag{2.5}$$

The interpolated position P''_i of P_i on the segment $P_s P_e$ (calculated using the Synchronous Euclidean Distance) is expressed in Formula (2.6):

$$P''_i(t_i, x''_i, y''_i) = \left(t_i, \frac{(x_e - x_s)t_i + x_s t_e - x_e t_s}{t_e - t_s}, \frac{(y_e - y_s)t_i + y_s t_e - y_e t_s}{t_e - t_s} \right) . \tag{2.6}$$

Therefore, using (2.5) and (2.6), the distance between P'_i and P''_i is expressed as follows (2.7):

$$\text{distance}^2(P'_i, P''_i) = \left(\frac{(x_e - x'_e)^2 + (y_e - y'_e)^2}{(t_e - t_s)^2} \right) (t_i - t_s)^2 . \tag{2.7}$$

On the segment P_eP_s, this distance reaches its maximum value for the point P_e, therefore:

$$\max(\text{distance}(P'_i, P''_i)) = \text{distance}(P'_e, P''_e) = \text{distance}(P'_e, P_e) \ . \tag{2.8}$$

Since the Euclidean distance is a metric, we can apply the triangle inequality. Therefore, an upper bound for the approximation error (i.e. the distance between the real point P_i and its interpolation P''_i) does exist and is expressed as follows (2.9):

$$\text{distance}(P_i, P''_i) \leq \text{distance}(P_i, P'_i) + \text{distance}(P'_i, P''_i) \ . \tag{2.9}$$

(2.8) and (2.9) yield the following:

$$\text{distance}(P_i, P''_i) \leq \text{distance}(P_i, P'_i) + \text{distance}(P_e, P'_e) \ . \tag{2.10}$$

P_i being a well-predicted point:

$$\text{distance}(P_i, P'_i) \leq d_{\text{Thres}} \ . \tag{2.11}$$

The same goes for P_e as stated earlier, therefore:

$$\text{distance}(P_e, P'_e) \leq d_{\text{Thres}} \ . \tag{2.12}$$

Combining (2.10), (2.11), and (2.12) yields the final result:

$$\text{distance}(P_i, P''_i) \leq 2d_{\text{Thres}} \ . \tag{2.13}$$

□

2.4 Experimental Results

We now present our experimental study where we compare our STSS algorithm to some existing trajectory sampling algorithms. The retained algorithms are TD-TR, OPW-TR [23], and STTrace [45]. Despite being unusable in a streaming context, we still retained the TD-TR algorithm in order to have some sort of a baseline of what is achievable if offline processing were permitted. We excluded the Thresholds algorithms because their configuration is problematic since it requires fine tuning of two thresholds. We implemented the aforementioned algorithms in Java based on the instructions and pseudo-codes given in the corresponding papers.

We start by comparing STSS to TD-TR and OPW-TR in Section 2.4.1. This comparison is conducted using a dataset composed of 5263 trajectories collected from 10 rented cars evolving in La Martinique (which is an overseas region of France). These trajectories are sampled at a rate of one position per 15 seconds and contain a total of 367691 data points. In Section 2.4.2,

we compare STSS to the STTrace algorithm using the original Athens trucks dataset [15] (available online at [48]). The dataset is composed of 276 real trajectories of trucks moving along the Athens road network. The trajectories have various lengths, accounting for a total of 112203 data points uniformly sampled at a rate of one position per 30 seconds.

The performance criteria as well as the obtained results are detailed in what follows.

2.4.1 Comparison with OPW-TR and TD-TR

TD-TR, OPW-TR, and STSS are all algorithms that give the user the possibility to directly control the tolerable approximation error resulting from the sampling process. In the case of both TD-TR and OPW-TR, configuring the algorithm with a distance threshold d_{Thres} yields approximation errors that are at most equal to this very same threshold. However, in the case of STSS configuring the algorithm with a distance threshold d_{Thres} results in errors that have an upper bound of $2d_{\text{Thres}}$ (as shown in Section 2.3.2).

We compare the three algorithms based on their theoretical error bounds: if we set the distance threshold of TD-TR and OPW-TR to a given value, we set the distance threshold for STSS to half this value. We vary this theoretical error bound from 10m to 100m with a step of 10m and we observe, for each value, the distribution of the real errors (i.e. the distances between real points and their approximations on the compressed trajectories) resulting from the use of each algorithm as well as the compression ratio that it achieved. The compression rate is expressed in Formula (2.14).

$$\text{Compression Ratio} = \frac{\text{number of data points in the compressed dataset}}{\text{number of data points in the original dataset}} . \tag{2.14}$$

Figure 2.10 shows the compression ratios achieved by the three algorithms. As expected, the compression ratio decreases monotonically as the error threshold is set to higher values, resulting in a smaller, more concise dataset. The best compression ratios were unanimously achieved by TD-TR. This predictable result comes from the fact that, contrary to the other algorithms, TD-TR has a global vision of the entirety of the trajectory to be compressed and can, therefore, make wiser choices on which points are to be kept and which points can be deleted. For small theoretical error values (up to 30m), STSS performed better than OPW-TR. This trend was reversed for higher values where OPW-TR outperformed STSS. This suggests that, as the error threshold is set to higher values, OPW-TR tends to keep an increasingly larger window on the data, thus gaining a "more global" vision of the trajectory. This, combined with OPW-TR's backtracking capability makes it capable of making better choices on points to retain. No matter what the value of this error threshold is, STSS is still bound to make its decisions based only on the last seen position which hinders it from achieving better compression ratios.

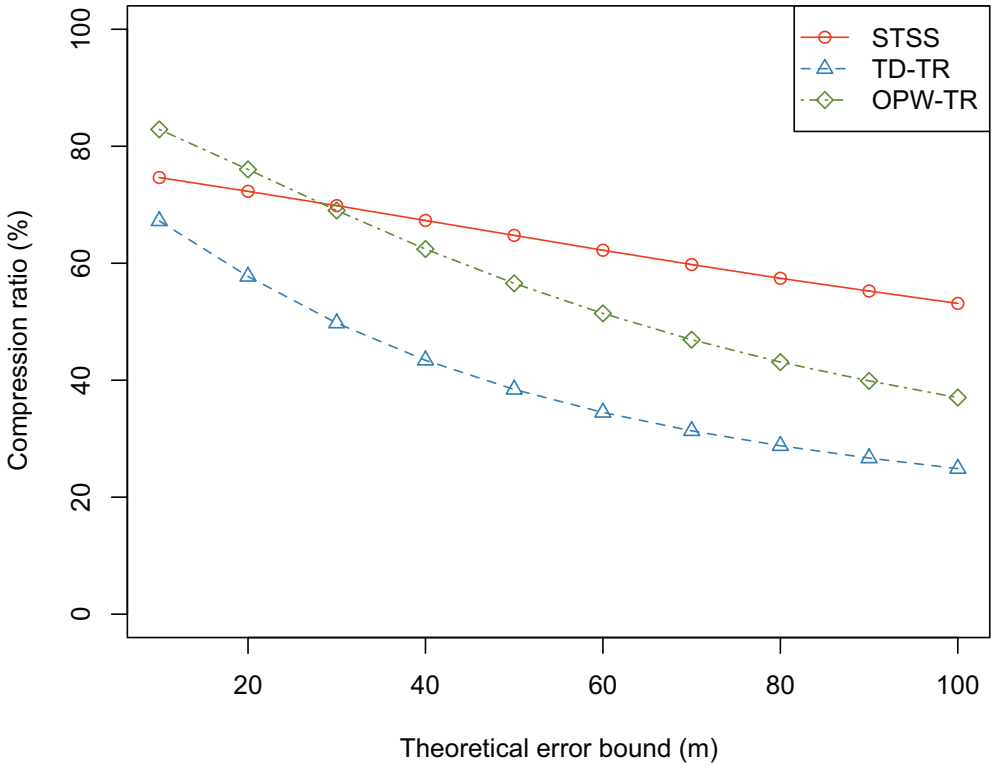


Figure 2.10 – Compression ratios delivered by TD-TR, OPW-TR, and STSS for different values of the theoretical error bound.

The distributions of the real approximation errors for the various values of the theoretical error bound are charted in Figure 2.11. In the case of TD-TR and OPW-TR, the real errors are uniformly distributed on the whole range of values up to the theoretical error bound (the slightly higher error values for TD-TR result from the lower compression ratios it achieved). In the case of STSS, the approximation errors seem to be, for the larger part, comprised in the first half of this value range with only a very small number of points (considered as outliers in the boxplots) within the second half. This indicates that the theoretical error bound guarantee provided by STSS (cf. Section 2.3.2) is rather pessimistic: if STSS is configured with a distance threshold d_{Thres}, the vast majority of data points tend to be well-represented and have errors that do not exceed d_{Thres} but a small portion does exceed it while still being under the $2d_{\text{Thres}}$ bound.

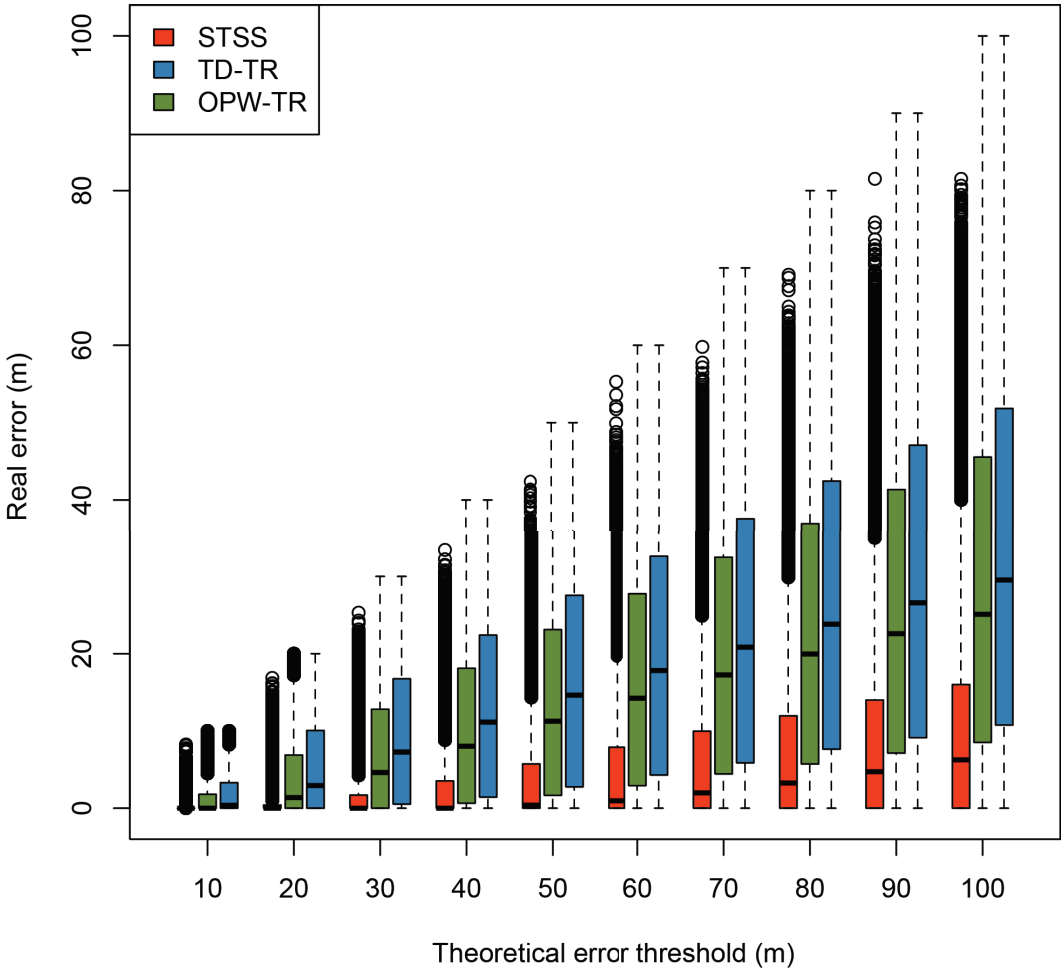


Figure 2.11 – Distributions of approximation error values resulting from application of TD-TR, OPW-TR, and STSS sampling.

2.4.2 Comparison with STTrace

Contrary to STSS, STTrace offers no means to control the approximation error resulting from the compression process. On the other hand, its configuration requires knowledge of the storage size M (number of data points that the user wishes to retain). In order to be able to compare both algorithms, we devised the following approach. For the different compression ratios achieved by STSS, we configured STTrace to use the same amount of space for each trajectory (i.e. each trajectory is compressed with STTrace using the same number of points of its STSS counterpart). We then compared the average and maximum approximation errors achieved by both approaches. The rationale behind this test case scenario is to try to answer the question: "given the same storage space that STTS used, does STTrace compression achieve a better quality?"

The average approximation error is calculated based on Formula (2.15) and the maximum approximation error based on Formula (2.16).

$$\text{Average Approximation Error} = \frac{1}{\sum_{T\in\mathcal{T}}|T|}\cdot\sum_{T\in\mathcal{T}}\sum_{P_i\in T}\text{distance}(P_i, P_i'') \ . \quad (2.15)$$

$$\text{Maximum Approximation Error} = \max_{T\in\mathcal{T}}(\max_{P_i\in T}(\text{distance}(P_i, P_i''))) \ . \quad (2.16)$$

$\mathcal{T}$ is the dataset including all the trajectories. For a data point P_i belonging to a trajectory T, P_i'' is its approximation on the compressed trajectory T_C obtained by sampling from T. $|T|$ is the number of points in T.

Figure 2.12 shows the evolution of the average approximation error for both algorithms at different compression ratios. Again, the approximation error increases as the dataset is more compressed (i.e. as the compression ratio decreases). For high compression ratios, the two algorithms behave quite similarly w.r.t. the average approximation error, which indicates that, in general, the majority of points still retain a comparable quality for both STTrace and STSS. However, for lower compression ratios the quality of STTrace sampling degrades drastically and the difference between the error values for STSS and STTrace can reach one order of magnitude.

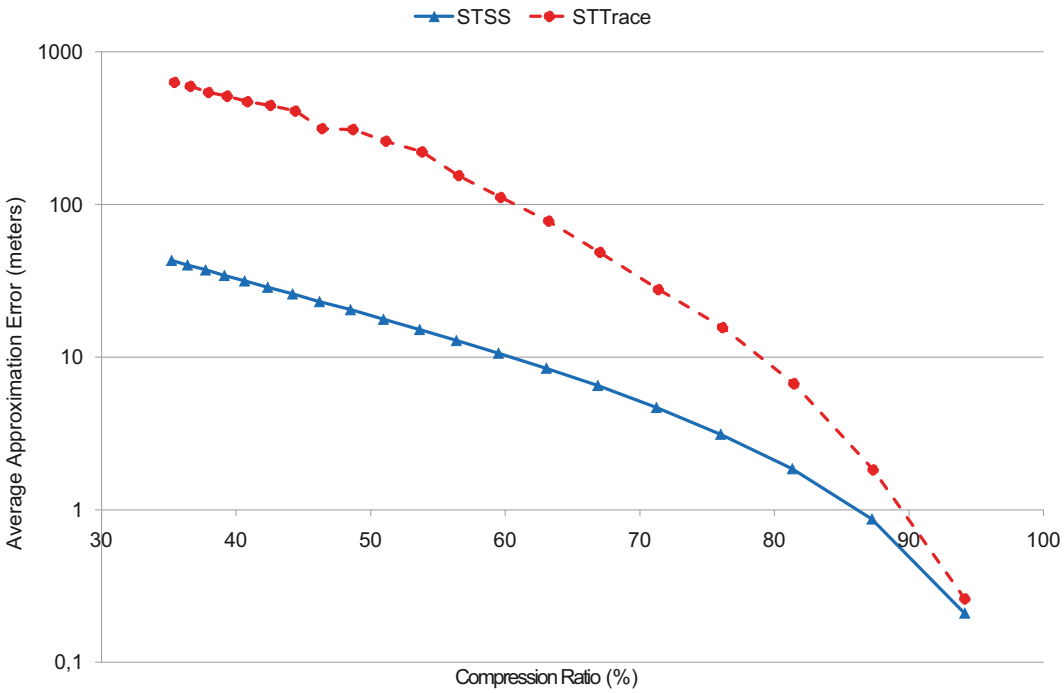


Figure 2.12 – Average approximation errors resulting from STSS and STTrace compressions.

Maximum approximation errors are illustrated in Figure 2.13 which shows that, when using STTrace, some data points can suffer from severe degradation

of their quality even at high compression ratios. This degradation is mainly due to two factors: (i) the point insertion mechanism used to accept new points into the sample, which makes it harder for new points to get inserted in the sample as time goes by (the allowed distortion threshold keeps increasing to a point where new points are hardly accepted); and (ii) discarding a point from the sample is based on its local information (the SED deduced from its immediate predecessor and successor points), which makes it vulnerable to error propagation: a point can be deleted because it is deemed unimportant based solely on its neighbors but once these neighbors get deleted at a later stage, the original point can become very miss-represented.

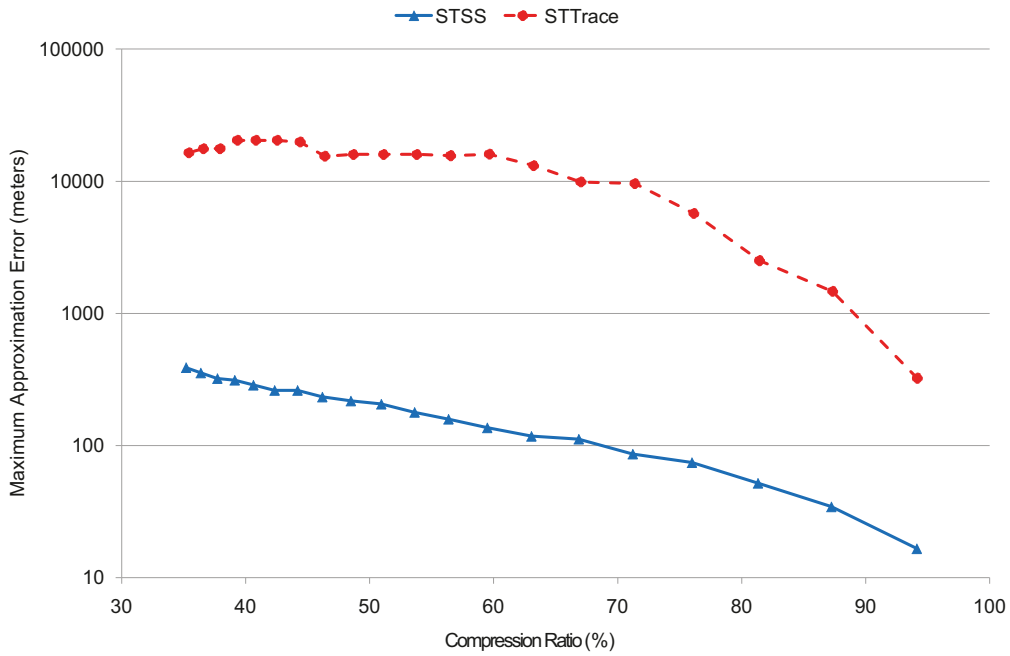


Figure 2.13 – Maximum approximation errors resulting from STSS and STTrace compressions.

Using the same amount of space to compress trajectories, STTrace is outperformed by STSS. The latter achieves a better compression with lower errors especially when severe compression is required. Contrary to STSS, STTrace offers no guarantees on quality degradation and can consequently be unreliable to sample moving object trajectories especially if the compressed trajectories are to be used later on for analysis and mining tasks that require precision.

2.5 Conclusions

In this chapter, we explored the problem of trajectory sampling in streaming environments. We observed that existing trajectory compression techniques

are either (i) computationally expensive but allow the user to control the approximation error; or (ii) computationally efficient but offer no guarantee whatsoever w.r.t. the quality degradation resulting from the compression. To address this shortcoming, we proposed the Spatiotemporal Stream Sampling (STSS) algorithm. STSS combines time and in-memory efficiency with guaranteed compression error bounds. Its sampling process is intelligently guided by both the spatial and temporal features of the trajectory stream to which it's applied. Moreover, STSS is very simple to configure with only one parameter (the distance threshold d_{Thres}) to set. Furthermore, this parameter is directly related to the compression error which can help guide the practitioner set it to a suitable value based on his knowledge of the domain of application and the approximation quality he desires to achieve.

Like other trajectory sampling algorithms presented in this chapter, STSS can be vulnerable to the presence of noise in the original trajectories. In this case, the algorithm is susceptible of interpreting noisy positions as changes in the motion pattern of the moving object, thus retaining more data points and resulting in poor compression. STSS also makes the hypothesis that it disposes of infinite storage to store the produced sample and does not offer a means to control the sample size. In fact, the compression ratio achieved by STSS can be perceived as a rough indicator of the complexity of the movement in the original trajectory. Finally, STSS trades off compression ratio for the sake of time-efficiency. This makes it less effective (compression-wise) compared to higher complexity algorithms. In situations where the computational efficiency is not an issue, we do recommend using algorithms such as OPW-TR over STSS in order to achieve a better compression.

As a concluding remark, since STSS is resource-efficient, it can be used either to construct summaries of trajectory streams that are kept for later processing and analysis tasks or simply as a load shedding mechanism to help reduce the charge of the system in case of high arrival rates. However, we believe that it makes more sense to implement and run STSS on the moving object's side rather on the DSMS side. The advantage of this is twofold: (i) the moving object can, based on the quality of the GPS positions it calculates, decide if these positions should be included in the compression process or if they should be discarded (which might help with STSS's vulnerability to noise); and (ii) streaming only the sampled trajectory can help reduce the communication costs and the load of the central DSMS. Companies such as Masternaut [49] (which is one of Europe's leading telematics providers) do implement similar sampling approaches as part of their fleet management solutions.

Chapter 3

Clustering Network-Constrained Trajectory Data Using Community Detection in Graphs

Traffic congestion has become a major problem that affects many human activities on a daily basis, resulting in both serious transportation delays and environmental damage. According to the Texas A&M Transportation Institute's 2012 Urban Mobility Report [5], travel delays in the United States were estimated to 5.5 billion hours in 2011 compared to roughly 1.1 billion hours back in 1982. The national congestion problem cost 5.5 billion gallons of wasted fuel (1.1 billion gallons in 1982) and caused economical losses as elevated as 121 billion 2011 dollars (only 24 billion 2011 dollars in 1982). The alarming increase in these indicators is mainly the result of the ever increasing number of commuters traveling along the road networks on a daily basis. This brings forth the need for tools for efficient extraction of useful knowledge about traffic and flow dynamics in road networks.

Monitoring the state of the road network is commonly conducted by using dedicated sensors that register the number of vehicles passing by the section where they are installed. The prohibitive cost of deploying and maintaining such sensors limits their deployment to the highways and the road network's main arteries. Furthermore, vehicles cannot be identified uniquely across such sensors and their trajectories along with their motion patterns cannot be deduced. Consequently, the collected data portray a partial and incomplete state of the road network which complicates data mining tasks that aim to extract useful knowledge about flow dynamics and the behavior of drivers moving along the network.

To address these shortcomings, an alternative approach may consist in analyzing GPS logs collected using location-aware devices (e.g. classic GPS, smartphones, PDAs, etc.). These logs can be acquired through probing vehicles, dedicated data acquisition campaigns (using buses, taxis, or an enterprise's fleet of vehicles), or even by means of a crowdsourced approach where different individuals willingly contribute by uploading their commute logs. Therefore,

it is perfectly feasible to collect large amounts of trajectory data, selectively reduce their size and eliminate redundancies with sampling techniques such as STSS (Chapter 2), and store them in dedicated databases (known as Moving Object Databases [4]). These data offer a better coverage of the road network and trajectories flowing in it and can be, later on, explored using data mining and statistical learning techniques.

Clustering is one of such techniques. Given a set of observations, cluster analysis consists in partitioning these observations into groups (called clusters) in such fashion that objects belonging to the same group are more similar to each other (w.r.t. a given criterion) than to objects from other groups. Existing work on trajectory clustering mainly focused on the case of free-form trajectories where moving objects can move unrestrictedly in an Euclidean space. Notable formulations of this problem include flock patterns [50, 51, 52, 53, 54], convoy patterns [55, 56, 57] as well as the well-known partition-and-group framework [58]. Such approaches neglect the presence, in the case of car trajectories as well as in other cases, of an underlying network that constrains the movement. The network's constraints, however, do play a paramount role in determining the similarity between the trajectories to be clustered. Clustering moving object trajectories under road network constraints gained interest only recently with the publication of work such as [8, 11, 10].

In the present chapter, we explore the problem of clustering network-constrained trajectory data. More precisely, we define two clustering problems: (i) the network-constrained trajectory clustering problem which aims at discovering groups of trajectories with similar behavior when moving along the road network, and (ii) the road segment clustering problem which tries to unravel clusters of segments that are frequently visited together by the same moving objects. In order to address these problems, we build upon the interesting idea of using graph-clustering techniques in the context of trajectories, recently introduced in [59]: we transpose our two clustering problems into graph-clustering problems that we try to solve afterwards using a community-detection approach. The result is a hierarchy of nested trajectory (or road segment) clusters that are suitable for exploration at various levels of detail.

The content of this chapter is organized as follows. In Section 3.1, we present our formal definition of the two clustering problems we are trying to solve. Related work is discussed in Section 3.2 where we survey existing techniques for clustering and measuring the similarity of trajectory data as well as work on graph clustering which is relevant to our work. In Section 3.3, we present a graph-based approach to clustering moving object trajectories while accounting for the presence of an underlying road network. We present our experimental study as well as some directives on how the discovered trajectory clusters can be explored in this same section. We extend our approach to cover the case of road segment clustering in Section 3.4. Finally, conclusions are drawn in Section 3.5.

3.1 Problem Formulation

Since we are interested in the trajectory clustering problem in the particular context of road networks, we opt for the symbolic data representation (already presented in Section 1.1.2) : the road network is represented as an oriented graph depicting its intersections and road segments (Definition 3, page 10) and each trajectory that traveled along this road network is represented as the ordered sequence of visited road segments (Definition 4, page 10).

In a real-case scenario, trajectories are collected as GPS logs (sequences of latitude and longitude points) on which a map-matching technique (e.g. [7, 9]) is applied in order to produce the sequence of traveled segments. The map matching step is out of the scope of this work: we make the hypothesis that the trajectories are already and correctly map-matched to the corresponding road segments.

Given both Definition 3 and Definition 4, we observe that two entities can be potentially interesting: (i) the moving object trajectories that traveled the road network, and (ii) the road segments that were visited by those trajectories. Therefore, we define two inter-related clustering problems.

For instance, the practitioner can be interested in discovering groups of moving objects that behaved similarly by visiting the same parts of the road network. Studying such groups as well as their interactions (e.g. two groups that converge to a common waypoint and move together afterwards) can produce beneficial knowledge about the flow dynamics and help measure the road network's adequacy to its real usage. We formalize this problem of clustering trajectories as follows:

Definition 8 (Network-Constrained Trajectory Clustering Problem)**.** Given a road network represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{S})$ and a set of network-constrained trajectories $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ that traveled along it, trajectory clustering aims to partition the set of trajectories $\mathcal{T}$ into a set of disjoint clusters $\mathcal{C}_\mathcal{T} = \{C_1, C_2, ..., C_K\}$ in such fashion that:

- Trajectories that behaved similarly and visited the same parts of the road network (i.e. that share a considerable number of common road segments) should be regrouped together in the same cluster C_i.
- Trajectories that are separated into different clusters C_i and C_j should be as dissimilar as possible and should share the least possible number of common road segments.

Alternatively, we can be interested in grouping similar road segments that are often visited together. The rationale behind this is that segments within a same cluster are expected to behave similarly, so if, for instance, a congestion occurs in a subset of these segments, one can reasonably presume that the congestion will spread and affect the other members of the cluster. Our formulation of this road segment clustering problem is given in the following definition.

Definition 9 (Road Segment Clustering Problem)**.** Given a road network represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{S})$ and a set of network-constrained trajectories $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ that traveled along it, road segment clustering aims to partition the set of road segments $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ into a set of disjoint clusters $\mathcal{C}_\mathcal{S} = \{C_1, C_2, ..., C_K\}$ in such fashion that:

- Segments grouped in the same cluster C_i are visited by a considerable amount of common trajectories (i.e. a trajectory T that visits a segment $s \in C_i$ also visits a fair amount of segments in this same cluster).
- Segments belonging to two different clusters C_i and C_j are visited by as few common trajectories as possible (i.e. they are unlikely to be part of a same trajectory).

Before presenting our approaches to the clustering problems we just defined, we review related work in the following section.

3.2 Related Work

Existing work on trajectory clustering can be divided into two main axes: (i) the study of trajectory similarity and distance measures (Section 3.2.1), and (ii) the development of trajectory clustering algorithms and approaches (Section 3.2.2).

Another aspect that is relevant to our work is graph clustering. We briefly present essential concepts related to this area in Section 3.2.3.

Notice that we deliberately omit mathematical details of the discussed approaches when those are irrelevant in order to relieve the reader from unnecessary details. Those details can be easily retrieved by referring to the cited papers.

3.2.1 Trajectory Similarity and Distance Measures

We distinguish between similarity and distance measures proposed in the context of geometric (or free flow) trajectories and those proposed in the network-constrained context.

3.2.1.1 Comparing Free Moving Trajectories

In terms of similarity-based trajectory retrieval [60], one is interested in the movement shape of the studied trajectories; sequences of sampled vectors are important in measuring the similarity between two trajectories and time components can be ignored. Given this definition, trajectories are compared exclusively based on their spatial features. Similarity and distance measures, in this case, should account for two main aspects: (i) the presence of noise and shifts in data (commonly due to signal disturbances and GPS failures), and (ii)

time shifting resulting from trajectories being sampled at different rates and at different time instants.

The most straightforward way to compare two free moving trajectories T and T' is by using the classic Euclidean distance. However, the use of Euclidean distance is reserved to comparing two trajectories having exactly the same number of points since it is not capable of handling time shifting. Moreover, the Euclidean distance is very sensitive to the presence of noise and outlier data points.

Dynamic Time Warping (DTW) [61, 62] was largely used for comparing time series and can be easily applied to trajectories. DTW tries to find the best matching between the sequences of points in both trajectories T and T' (i.e. the one minimizing the sum of Euclidean distances between paired points). In order to handle time warping, data points can be replicated at will. However, each data point of T must be paired with a data point of T' which makes DTW vulnerable to noise.

In [63], Vlachos et al. use the concept of Longest Common Subsequence (LCSS) to propose a set of distances and similarity measures for trajectories. LCSS proceeds in a similar fashion to DTW as it tries to find the best matching of the points of both trajectories. But in order to offer robustness to noise, data points can be overlooked (i.e. not matched). LCSS concentrates on similar parts of the compared trajectories and totally neglects their dissimilar parts. Consequently, it only gives a rough and coarse estimation of the similarity between trajectories.

Edit distance with Real Penalty (ERP) [64] and Edit Distance on Real sequence (EDR) [60] are adaptations of the string edit distance to moving object trajectories. Both calculate the minimum number of operations required to transform the trajectory T into T' (with two possible operations: replacement of a data point with another and time warping). Unlike ERP, EDR is robust to the presence of noise. Chen et al. claim that both ERP and EDR achieve better results than the coarse LCSS-based measures proposed in [63].

Other work on free moving trajectory similarity includes the One-Way Distance [65] (which is also based on the spatial shape of trajectories) as well as [66, 67] where the proposed distance measures are both shape and time dependant.

When it comes to comparing trajectories in the presence of a road network, the aforementioned propositions are inappropriate. This inadequacy is illustrated in Figure 3.1 which depicts three trajectories T_A, T_B, and T_C moving along a portion of a road network: when only geometric features of the trajectories are considered, T_C is more similar to T_B than T_A. However, if we account for the underlying road network's constraints, it is clearly visible that T_A is closer to T_B than T_C is.

Moreover, these distances and similarity measures are often computationally-expensive. Therefore, mechanisms (such as dimensionality reduction, pruning techniques, etc.) are often used to enhance retrieval time.

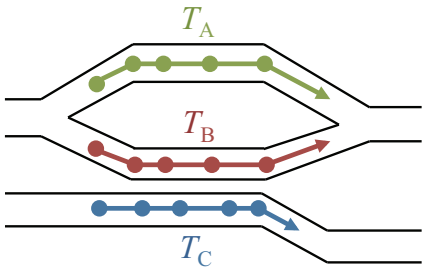


Figure 3.1 – Measuring trajectory similarity on road networks. Figure adapted from [10].

3.2.1.2 Measuring the Similarity of Network-Constrained Trajectories

To the best of our knowledge, the first attempt to measure the similarity of network-constrained trajectories is the one described in [68]. The authors proceed in two steps: (i) a filtering step based on spatial similarity, and (ii) a refinement step based on temporal distance. During the filtering step, trajectories are compared to a set of pre-established Points of Interest (POI) that can represent important road intersections and places. Only trajectories that pass by all the POI are retained for further refinement. The temporal distance between two trajectories T and T' is simply an Lp distance based on the time differences between the dates at which T and T' visited each POI. The closer in time these visits are, the more the two trajectories are considered similar (3.1):

$$dist_t(T, T') = (\sum_{i=1}^{k} |t(T, p_i) - t(T', p_i)|^p)^{\frac{1}{p}} \ . \tag{3.1}$$

$P = \{p_1, p_2, ..., p_k\}$ designates the set of POI and $t(T, p_i)$ the date (time-stamp) when trajectory T visited the POI p_i.

In [69], the authors introduce the additional concept of Times of Interest (TOI), which can represent congestion times, important events, etc. They complement the approach introduced in [68] by proposing two additional methods: the first is based on temporal filtering and spatial refinement (inversely to the previously described method) while the second uses a spatio-temporal similarity and distance in both the filtering and refinement steps. Approaches presented in [68, 69] cannot be used in an unsupervised clustering context since they require POI and TOI to be defined in advance. A similar approach is presented by Zhao et al. in [70].

Other approaches define cost functions to evaluate the cost of traveling along the road network and use them as a basis for comparison between trajectories. In most cases, the cost function (that we will denote c) is based on shortest path computations (potentially using information such as transit time, the spatial length of the borrowed path, etc.). One example of such cost-based approaches

is the one presented by Tiakas et al. in [71]. The authors adopt the same road network model as presented in Definition 3. A trajectory T moving on the network, on the other hand, is modeled as the set of timestamped vertices (and not segments) traversed by the moving object:

$$T = \langle (v_1, t_1), (v_2, t_2), ..., (v_n, t_n) \rangle \ . \tag{3.2}$$

Tiakas et al. propose a spatial distance and a temporal one. The spatial distance between two trajectories T and T' of equal lengths is calculated based on the network's graph as follows:

$$D_{net}(T, T') = \frac{1}{n} \sum_{i=1}^{n} d(v_i, v'_i) \ . \tag{3.3}$$

Where $d(v_i, v'_i)$ is the distance between vertices v_i and v'_i, calculated using any valid travel cost function c (e.g. shortest path calculation):

$$d(v_i, v'_i) = \begin{cases} 0 & \text{if } c(v_i, v'_i) = c(v'_i, v_i) = 0 \ ; \\ \frac{\min(c(v_i, v'_i), c(v'_i, v_i))}{\max(c(v_i, v'_i), c(v'_i, v_i))} & \text{otherwise .} \end{cases} \tag{3.4}$$

The proposed temporal distance attempts to measure the resemblance between trajectories w.r.t. the time required to travel in-between vertices. In order to evaluate the overall distance between trajectories, the authors suggest to either (i) calculate the weighted sum of both the spatial and temporal distances, or (ii) use both separately in combination with user-defined thresholds. In both cases, it is hard to apply the proposition in a clustering context since it would require fine tuning of the thresholds (or weights) to appropriate values.

Another approach to measuring network-constrained trajectory similarity is described by Chang et al. in [21]. The authors use the same model as defined in definitions 3 and 4 but additionally suppose that lifespans (start time and end time) of trajectories are also known. A spatial distance between road segments is defined as the average of the network distance (which is naturally based on shortest path calculation) between their start vertices and end vertices. Given two segments $s = (v_1, v_2)$ and $s' = (v'_1, v'_2)$, this distance is expressed as follows:

$$D_{edge}(s, s') = \frac{c(v_1, v'_1) + c(v_2, v'_2)}{2} \ . \tag{3.5}$$

With c, here again, being a function that calculates the cost of moving between a source and a destination vertex. Given two trajectories T and T', a matching algorithm is applied in order to find the best pairing between road segments in T and those in T': each segment in T is matched with its closest segment in the remaining segments of T' (i.e. segments are not reused). The spatial distance between T and T' is the sum of distances between the pairs of matched segments.

Chang et al. also propose a temporal distance in [21]. Based on its lifespan, each trajectory is assigned a time triplet: time range of a day (R), day of a week

(D), and week (W). The temporal distance between T and T' is calculated using a varying combination of the differences between their time triplets. The spatial distance and temporal distance can both be combined (either by addition or multiplication) and used as a spatiotemporal distance to compare trajectories.

Xia et al. [22] define Jaccard-inspired temporal and spatial similarities to compare network-constrained trajectories:

$$Sim(T, T') = \frac{L_c(T, T')}{L(T) + L(T') - L_c(T, T')} \ . \tag{3.6}$$

Where $L_c(T, T')$ is the spatial or temporal length of common parts between T and T', and $L(T)$ the total (spatial or temporal) length of T. The spatial and temporal similarities can be combined by multiplication in order to obtain a spatiotemporal similarity that accounts for both dimensions.

Distance measures that are based on cost functions (e.g. shortest path calculation) using the road network's graph are natural and quite intuitive. However, depending on the graph's size and the complexity of the implemented cost function, they can sustain severe overheads, which limits their applicability in the context of clustering big trajectory datasets.

3.2.2 Trajectory Clustering

Cluster analysis was studied extensively in the context of static data. A considerable number of approaches were proposed in the literature, such as partitioning approaches (e.g. k-means [72]), hierarchical approaches (AGNES, DIANA [73], BIRCH [74], etc.), density-based approaches (e.g. DBSCAN [75] and OPTICS [76]), etc. Prior and ongoing research on trajectory clustering consisted mainly in adapting those existing techniques to the new context of moving object data.

Several classifications can be established for trajectory clustering techniques based on the many aspects they involve. These aspects are the following:

- Data representation: depending on the context, trajectories can be represented geometrically (as a set of positions in Euclidean space) or symbolically (as a set of visited points of interest).
- Dimensionality: similarity between trajectories can be evaluated based on: (i) their spatial features only, (ii) their temporal features only, or (iii) both their spatial and temporal features at the same time.
- Granularity: clustering can involve (i) trajectories in their entirety, (ii) sub-trajectories (or segments), (iii) separate data points, etc.
- Integration of an underlying network's constraints: in contexts such as road networks, moving objects abide by the rules of an underlying network that constrains their movements. Some approaches choose to integrate the network's constraints while evaluating trajectory similarity

whereas others neglect them. This aspect is intimately related to data representation.

The most known trajectory data clustering approaches are reviewed in what follows. These include (in order of appearance): (i) work on managing moving clusters, (ii) flock patterns, (iii) convoy patterns, (iv) the partition-and-group framework, (v) work on trajectory clustering and analysis with graph-based approaches, and finally (vi) clustering in the particular case of network-constrained trajectories.

3.2.2.1 Moving Clusters

In [77], Li et al. introduce the notion of Moving Micro-Clusters (MMC) which is an extension of the concept of micro-clusters proposed in [74] for clustering static data. An MMC, according to [77], "denotes a group of moving objects that are not only close to each other at current time, but also likely to move together for a while." Formally, an MMC is a group of n similar objects. Each object o_i is represented under the form $(x_i, y_i, vx_i, vy_i, t)$ with (x_i, y_i) being the position of o_i and (vx_i, vy_i) its velocity at instant t $(i = 1, 2, ..., n)$. A profile $(x, y, vx, vy, t) = (\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i, \sum_{i=1}^{n} vx_i, \sum_{i=1}^{n} vy_i)/n$ is maintained for each MMC. This makes it possible to treat MMC in the same fashion as simple moving objects and establish a hierarchy of nested MMCs with various levels of abstraction. In a similar fashion to what is proposed in the BIRCH algorithm [74], Li et al. also keep a Clustering Feature (CF) to describe each MMC. The CF can be updated incrementally to account for MMC merging and splitting operations. Initially, the MMCs are generated using the k-means algorithm in combination with a distance that accounts for both the position and velocity of the moving objects. Once these MMCs are discovered, their collisions (resp. expansions) over time are managed using Minimum Bounding Rectangles (MBR) resulting into merging (resp. splitting) operations.

A very similar approach to [77] is presented by Jensen et al. in [78]: the Moving-Object Clustering approach is a direct adaptation of the BIRCH algorithm [74] to trajectory clustering. The similarity between moving objects, at a given time instant, is evaluated using a weighted Euclidean distance and a modified version of BIRCH is used to manage appearances and disappearances of moving objects as well as to handle moving cluster merges and splits (and updates of their cluster features).

A different vision of moving object clustering is presented in [79] where Kalnis et al. define a moving cluster as a sequence of spatial snapshot clusters $MC = \{c_1, c_2, ..., c_k\}$ such as, for each two consecutive snapshot clusters, the ratio of common members between both clusters to their overall members does not get below an integrity threshold θ ($\forall 1 \leq i < k, \frac{|c_i \cap c_{i+1}|}{|c_i \cup c_{i+1}|} \geq \theta$). An example of a moving cluster according to this definition is depicted in Figure 3.2. The fundamental difference between this definition and the techniques that will be explored in later sections is that membership in the moving cluster is not

mandatory during the whole lifespan of the cluster: moving objects can freely leave or join the moving cluster at anytime as long as the integrity constraint is respected.

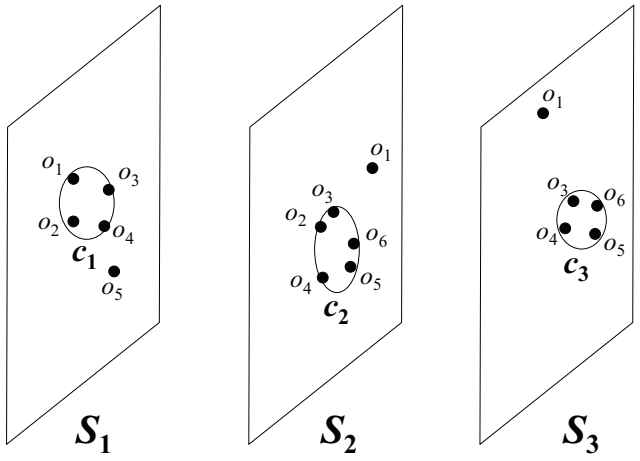


Figure 3.2 – Example of a moving cluster according to the definition in [79] (for $\theta = 0.5$). At each snapshot (S_1, S_2, and S_3), DBSCAN is used to regroup moving objects based on density. At the first time instant (represented by the snapshot S_1), the moving cluster is formed by the snapshot cluster c_1 regrouping four moving objects (o_1, o_2, o_3, and o_4). At the second time instant, the moving cluster is compared to the detected snapshot clusters. Here, the moving cluster is successfully extended using the snapshot cluster c_2 since the ratio of common objects is superior to the defined threshold θ. The same goes for c_3 in the third snapshot (S_3). Notice that objects o_5 and o_6 joined the moving cluster and o_1 left it midway which is a fundamental difference in comparison to other formulations that will be seen later on. Figure adapted from [79].

The straightforward approach to discovering moving clusters [79] consists in applying DBSCAN [75] at each time instant in order to discover separate snapshot clusters then calculate intersections of these snapshot clusters and report moving clusters incrementally. Two heuristics are also provided in [79] in order to enhance processing time.

3.2.2.2 Flock Patterns

Flock patterns were first described by Laube et al. in [50]. Originally, a flock pattern is defined as a set of moving objects that happen to belong, at a particular time instant, to a disk of radius ϵ while traveling in the same direction. Benkert et al. [51] criticize this definition and argue that the moving objects should also be correlated over a given duration for the flock pattern to be relevant. We retain their definition (also adopted by Vieira et al. in [52]): a $Flock(\epsilon, minPts, Dur)$ is a group of at least $minPts$ moving objects that stayed in proximity with each other (i.e. within a same disk of radius

ϵ) for exactly *Dur* consecutive time instants. Figure 3.3 illustrates two flocks $f_1 = \{T_1, T_2, T_3\}$ (spanning between time instants t_1 and t_3 and containing the disks c_1^1, c_1^2, and c_1^3) and $f_2 = \{T_4, T_5, T_6\}$ (from t_2 to t_4 with disks c_2^2, c_2^3, and c_2^4).

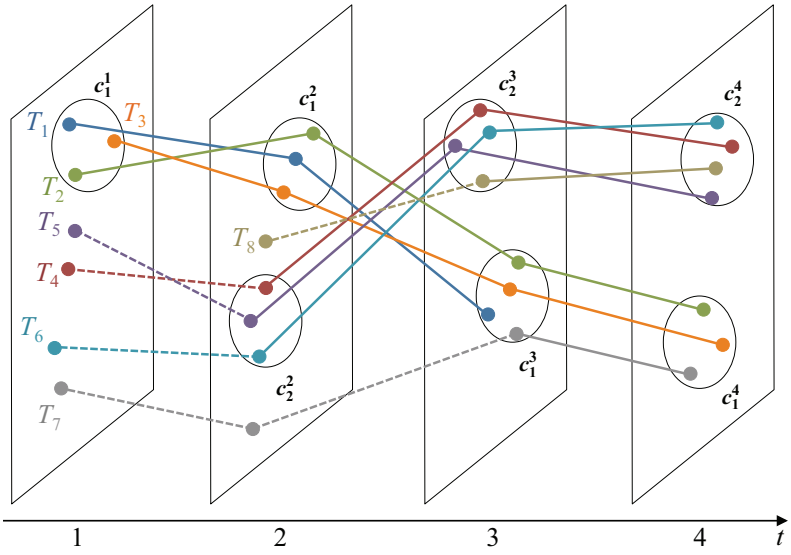


Figure 3.3 – Example of two flock patterns (according to the definition in [51, 52]). At each timestamp (here t_1, t_2, t_3, and t_4) trajectories are regrouped with circular disks c_i^j. The disks are compared across successive time instants to extract flocks. Here, two flocks f_1 and f_2 are extracted. f_1 is formed by the three discs c_1^1, c_1^2, and c_1^3 that were discovered in the time instants t_1, t_2, and t_3 respectively. f_1 regroups three moving objects represented by trajectories T_1, T_2, and T_3. f_2, on the other hand, spans over timestamps t_2, t_3, and t_4 and contains trajectories T_4, T_5, and T_6 since they were contained in the disks c_2^2, c_2^3, and c_2^4. Figure adapted from [52].

A straightforward approach to discovering flock patterns is described in [52]. It proceeds as follows: at each time instant t_i, all the disks containing $minPts$ objects are generated. These disks are then intersected with potential flocks that were retained at the previous time instant t_{i-1}. If the intersection between a potential flock f and a disk c still contains more than $minPts$ objects, then the result is retained for the next iteration. When a flock f is successfully extended over a period of *Dur* timestamps, it is reported immediately. Heuristics are also provided [52] in order to reduce the time complexity of flock discovery.

Variants of the flock pattern include the maximum duration flock pattern proposed in [53, 54] which consists in trying to extend the flock patterns beyond the *Dur* consecutive time instants instead of reporting them immediately.

3.2.2.3 Convoy Patterns

Jeung et al. [55, 56] criticize the disk constraint in flock patterns and qualify it as very constraining and counter-intuitive since it can easily ban moving objects from groups they naturally belong to (cf. Figure 3.4). Instead, the authors argue that moving objects should be regrouped based on a density criterion.

A $Convoy(\epsilon, minPts, minDur)$ is a group of moving objects that stay density-connected (w.r.t. the two parameters ϵ and $minPts$) for at least $minDur$ consecutive timestamps. The groups of density-connected moving objects can be discovered using a density-based algorithm such as DBSCAN [75].

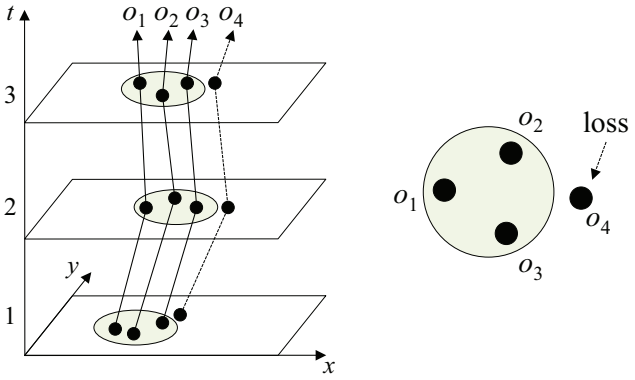


Figure 3.4 – Example of a lossy flock. It is clear that moving objects o_1, o_2, o_3, and o_4 should naturally be regrouped together. This can be easily achieved by a density-based approach (i.e. using DBSCAN). However, the moving object o_4 is excluded from the flock pattern composed by o_1, o_2, and o_3 due to the restriction on the disk shape used as the grouping criterion. Figure adapted from [56].

A convoy pattern query consists in retrieving and reporting maximum length convoys in a dataset of trajectories $\mathcal{T}$. This can be achieved naively by applying the CMC (Coherent Moving Cluster) [56] algorithm: at each timestamp t_i, the DBSCAN algorithm is applied to the positions of all moving objects at t_i in order to build spatial snapshot clusters. All combinations of snapshot clusters are then inspected in order to find and report the present convoys. The authors also present the CuTS (Convoy Discovery using Trajectory Simplification) family of heuristics. These heuristics use line simplification algorithms (e.g. DP and TD-TR) in order to reduce the number of points to be processed and enhance processing time.

The convoy discovery algorithms originally proposed in [56] suffered from accuracy problems, resulting in some valid convoys being skipped and false convoys being reported. The issue was addressed later in [57] where more precise algorithms are described.

3.2.2.4 The Partition-and-Group Framework

The Partition-and-Group Framework [58] is one of the most known work on trajectory clustering. The TraClus (Trajectory Clustering) algorithm implementing this framework proceeds in two steps (Figure 3.5): (i) a partitioning step where trajectories are down-sampled, then (ii) a grouping step during which clusters of sub-trajectories are discovered.

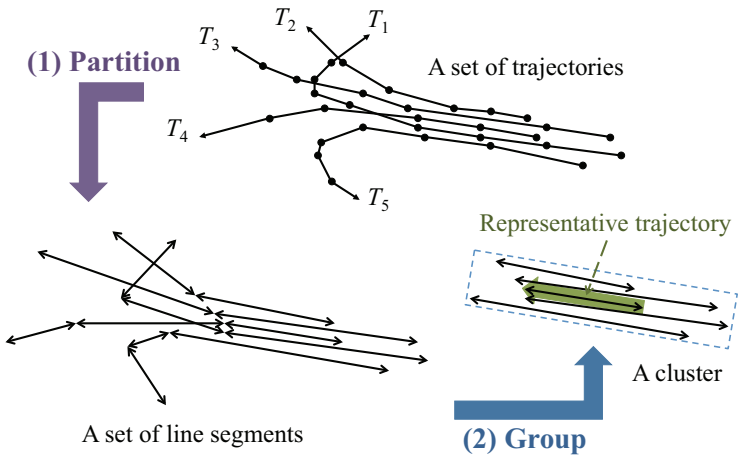


Figure 3.5 – Outline of the Partition-and-Group Framework. Figure adapted from [58].

During the partitioning phase, each trajectory is compressed using a MDL (Minimal Description Length) [80] algorithm: only a subset of points (called characteristic points) are retained and the trajectory is represented as the set of partitions (i.e. segments) linking these characteristic points. This phase is merely the application of a sampling technique such as the ones we reviewed in Chapter 2.

In the grouping phase, the similarity between partitions is evaluated using a three-component distance (which is a combination of a perpendicular, a parallel, and an angular distance), and a modified version of the DBSCAN [75] algorithm is applied in order to discover clusters of density-connected partitions. For each cluster, an artificial representative trajectory is built based on the member partitions.

The Partition-and-Group framework is a purely spatial approach since it does not consider the temporal features of the studied trajectories. The framework is extended in [81] where it is used for trajectory outlier detection and combined to region-based hierarchical clustering in [82] and used for trajectory classification. It also inspired work on micro clusters in [83].

3.2.2.5 Trajectory Clustering and Analysis using Graph-Based Approaches

Guo et al. [59] were, to the best of our knowledge, the first to use the insightful idea of applying graph-based clustering to trajectory data in an attempt to establish topological relationships among trajectories and locations they visited. The authors consider geometrically-represented trajectories (i.e. sets of GPS points). First, GPS points are smoothed with the help of a Delaunay triangulation. The result is a reduced set of representative GPS points (which helps eliminate data redundancies). Trajectories are then mapped to these representative points (based on their original GPS points) using a modified shortest paths algorithm.

In the second step, a graph is used to model relationships between GPS points: each GPS point is represented as a vertex in the graph and edges are drawn between these points based on trajectories they have in common (vertices that share the exact same trajectories are merged and considered as a single vertex). The graph is then clustered using the spatially-constrained graph partitioning method described in [84] in order to discover a hierarchy of regions of interest. These regions represent interesting patterns since they stand for areas to which a considerable number of trajectories are migrating. They are also used by the authors to generalize and regroup trajectories (each trajectory is assigned to the region that covers it the most) for a better understanding of motion patterns.

In [85], Brilhante et al. build graphs depicting relationships between moving object trajectories based on their interactions: each trajectory in the dataset $\mathcal{T}$ is represented as a vertex in the graph and edges are created between trajectories based on the frequency of their encounters (i.e. the number of intersections between their polygons). The authors extract features such as clustering coefficient and average shortest path length from the trajectory networks that they build and use them to study properties such as the presence of a power law distribution and the small world effect. However, no attempt is made to cluster the proposed trajectory graphs.

All of the propositions reviewed so far dealt with the case of free moving trajectories. Therefore, the used distances and similarity measures focus on the geometric properties of the data and do not account for the presence of any network-related constraints. Existing work in this context also includes [86] where the authors use DBSCAN to discover interesting places in trajectories based on their time dimension, discovery of frequently-traveled paths [87], fuzzy trajectory clustering [88], T-OPTICS and TF-OPTICS [89] which are adaptations of the OPTICS [76] algorithm to trajectory clustering, trajectory clustering using regression models and Expectation-Maximization (EM) algorithms [90], etc.

3.2.2.6 Clustering Network-Constrained Trajectories

Kharrat et al. present a density-based technique for clustering network-constrained trajectories in [8]. Using the same data model described in definitions 3 and 4, the authors start by building a transitions matrix containing the number of trajectories transiting between pairs of consecutive road segments. The matrix is clustered using the NETSCAN algorithm (which is a density-based algorithm inspired by DBSCAN) in order to discover dense paths in the road network. A dense path is a sequence of consecutive road segments with small variations of the density of trajectories (transitions) traveling in-between them. Once dense paths are discovered, a second phase consists in clustering the trajectories: each trajectory is compared to all dense paths it interacts with. If the similarity (based on the length of common parts) is above a user-defined threshold, then the trajectory is added to the cluster represented by the concerned dense path.

This approach is extended in [11] with the integration of time. The network-constrained trajectories are timestamped with the date of entry on each road segment. The new approach proceeds in two steps. First, the time interval covered by the dataset of trajectories is divided into sub-intervals and the NETSCAN algorithm is applied on the transitions matrix of each sub-interval in order to discover dense paths. Then, dense paths belonging to different sub-intervals are correlated using the DENSITYLINK algorithm in order to characterize their temporal evolution, represented as an evolution graph where vertices represent dense paths and edges represent their transitions over time.

The NNCluster algorithm for network-constrained trajectory clustering is presented in [10]. The authors first define a network-based distance based on shortest path calculations in a very similar fashion to distances proposed in [71, 21] (therefore, we omit the mathematical details and refer the reader to [10]). A baseline clustering algorithm is presented: trajectories are clustered using classic agglomerative hierarchical clustering [73] with the exception that cluster-merging operations are conducted using a custom linkage based on representative trajectories. The representative trajectory of a cluster C_i is the one that minimizes its average distance to other participants in the cluster (3.7):

$$rt(C_i) = \underset{T \in C_i}{\operatorname{argmin}} \frac{1}{|C_i|} \sum_{T' \in C_i} \text{distance}(T, T') \ . \tag{3.7}$$

For each merging operation, the algorithm finds the two clusters with the closest representative trajectories (distance-wise) and fuses them together. Once the dendrogram of nested trajectory clusters is retrieved, the Davies-Bouldin (DB) Index [91] of each level is evaluated and only the level with the best value is reported as the final result. NNCluster is a variation over this baseline algorithm where a k-NN based heuristic is first applied in order to retrieve initial clusters, thus reducing distance computations and enhancing processing time.

The work presented by Liu et al. in [92] can be seen as an extension of moving clusters (cf. Section 3.2.2.1) to the case of constrained trajectories. The authors maintain a set of Cluster Units (CU) [93] representing sets of moving objects moving closely together on the same road segments. The CUs are inspected on each passing of a new road intersection in order to decide whether they should be split or merged with other CUs moving on adjacent road segments.

Other efforts on clustering trajectories in a constrained context also include [94, 95, 96].

3.2.2.7 Synthesis

A synthesis of the characteristics of major work on trajectory clustering (in both the free moving and network-constrained cases) is presented in Table 3.1.

Table 3.1 – Characteristics of major proposals in trajectory clustering.

Approach	Dimensions		Clustering granularity	Clustering type	Network-constrained
	Time	Space			
Li et al. [77]	yes	yes	moving objects	partitionning	no
Jensen et al. [78]	yes	yes	moving objects	hierarchical	no
Kalnis et al. [79]	yes	yes	moving objects	density-based	no
Vieira et al. [52]	yes	yes	moving objects	-	no
Jeung et al. [55, 56]	yes	yes	moving objects	density-based	no
Lee et al. [58, 82, 81]	no	yes	segments	density-based	no
Guo et al. [59]	no	yes	data points	graph-based	no
Nanni et al. [89]	yes	yes	trajectories	density-based	no
Pelekis et al. [88]	no	yes	trajectories	fuzzy-clustering	no
Kharrat et al. [8]	no	yes	road segments	density-based	yes
Kharrat et al. [11]	yes	yes	road segments	density-based	yes
Roh et al. [10]	no	yes	trajectories	hierarchical	yes

From what we exposed so far, we draw the following observations:

- The majority of prior work focused on the case of free trajectories at the expense of the (more realistic) network-constrained case. However, this tendency was inverted in the latest years.

- A considerable portion of existing approaches consists in adaptations of density-based algorithms (DBSCAN and OPTICS). Such algorithms are somewhat hard to configure and require fine tuning of their parameters (ϵ and $minPts$ in the case of DBSCAN). Moreover, such approaches suppose that trajectories regrouped together have a homogeneous density, which is not always verified (cf. [10]).

- Proposed approaches are based on flat clustering. Even in rare occurrences where hierarchical clustering is used (e.g. [10]), only one level is reported as the end result. We argue that flat clustering can produce a high number

of clusters when analyzing large datasets, which can be a drawback for the user.

Considering these observations, we find that the approach presented by Guo et al. in [59] is conceptually promising. Conserving all the levels from a hierarchical clustering in order to analyze the trajectory data at various levels of abstraction can be very helpful. The authors' use of graph-based clustering in order to solve the trajectory data clustering problem is also very insightful and can provide a serious alternative to the predominant density-based approaches. Consequently, we maintain these ideas and build upon them in our work.

3.2.3 Graph Clustering

Graphs (or networks) are used extensively to model relationships and interactions between various types of entities in many domains (e.g. social networks, biology, mobile phone networks, etc.). Graph clustering is also an active area of research. In graph clustering (a.k.a. graph partitioning or community detection in graphs), we are interested in identifying clusters (commonly called communities) of highly connected components: we wish for as few as possible edges (with small weights) in-between communities and for edges within the same group to be numbered and have high weights.

Many approaches to graph partitioning were proposed in the literature[1] (complete surveys of which can be found in [97, 98]). For example, label-propagation graph clustering [99] works by labeling the vertices with unique labels and then updating the labels by majority voting in the neighborhood of the vertex. Another famous graph partitioning technique is spectral clustering [100]. Spectral clustering is, at its core, a relaxed version of the graph cut problem in which the aim is to retrieve a partition of the graph into K sets of vertices while minimizing the number (or weights) of the edges that traverse the cuts (i.e. edges between different vertex sets). The K sets can be retrieved either by a multiway approach (i.e. the graph is directly split into K clusters) or by a recursive bipartitionning approach (i.e. the vertices are recursively split into two clusters at a time until K clusters are retrieved).

Recently, modularity-based community-detection [101] gained a considerable popularity and was widely adopted, despite its limitations, as a method of choice to partitioning graphs [98]. Again, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, with vertices $\mathcal{V} = \{v_1, v_2, ..., v_n\}$, weighted edges $\mathcal{E}$ such as $\omega_{ij} \geq 0$ and $\omega_{ij} = \omega_{ji}$, and given a partition of the vertices into K clusters (or communities) $C_1, ..., C_K$, the modularity of the partition is expressed according to Formula (3.8):

$$\mathcal{Q} = \frac{1}{2m} \sum_{k=1}^{K} \sum_{i,j \in C_k} \left(\omega_{ij} - \frac{d_i d_j}{2m} \right) . \tag{3.8}$$

[1] The approaches discussed here are inherently different from blockmodeling techniques that will be presented later in Chapter 4.

With $d_i = \sum_{j \neq i} \omega_{ij}$ and $m = \frac{1}{2} \sum_i d_i$. The modularity measures the quality of the clustering by inspecting the arrangement of the edges within the communities of vertices. A high modularity is an indicator that the edges within the communities outnumber (or have higher weights than) those in a similar randomly-generated graph (i.e. one that does not present a community structure), and communities discovered using modularity optimization have a structure that is similar to the structure of cliques. We will be using modularity-based clustering in the context of our proposals.

3.3 A Graph-Based Approach to Clustering Network-Constrained Trajectories

We now study the network-constrained trajectory clustering problem. In order to do so, we propose an approach based on community detection in graphs. The outline of the approach is the following. First, we evaluate the resemblance between pairs of trajectories using a similarity measure that is inspired by information retrieval and weighting techniques (Section 3.3.1). Then, the calculated similarity values are used to build a graph depicting the relationships between trajectories (Section 3.3.2). The final step consists in using a modularity-based community detection algorithm in order to discover a hierarchy of nested trajectory clusters that are suitable for multi-level exploration and analysis (Section 3.3.3).

Additionally, we discuss various aspects related to our framework (such as cluster exploration, extraction of cluster representatives, and time complexity) in Section 3.3.4. Experimental results are also presented in Section 3.3.5.

3.3.1 Similarity Measure for Network-Constrained Trajectories

To a certain extent, comparing symbolically-represented trajectories bears a striking resemblance to comparing documents (or texts) for information retrieval. In document classification, each document is regarded as a bag-of-words and is represented as a collection of words while totally disregarding word order and phrase structure, which is imposed by grammar. Two documents are compared based on the words they share: the more words they have in common, the more similar they are. A wide range of similarity measures and distances can be used to achieve this end (such as the Jaccard index, Sørensen–Dice coefficient, cosine similarity, etc.).

By analogy, we consider that a trajectory is a bag-of-segments (i.e. an unordered collection of visited segments) and use this paradigm for measuring the similarity between trajectories. Comparison between two trajectories is conducted on a segment basis where the presence of each segment is checked individually without accounting for the segment's order in the trajectory or

the presence of other segments. This simplification is justified by the following observations:

- Much like grammar imposes a well-defined phrase structure, the road network's graph imposes that trajectories travel along connected segments.
- Even if the order is unaccounted for explicitly, the underlying network model is a directed graph. Consequently, the direction of travel is implicitly respected since the visited edges are not the same for each direction.
- Congestion situations occur first in singular isolated segments and spread afterward to adjacent segments. Considering individual segments as the basis for comparison is the most natural and intuitive choice.

When comparing documents, not all words have the same relevance. Function words (articles, pronouns, particles, etc.) are very common across all the documents and are consequently inefficient to assess the resemblance and form clusters of similar documents. Content words, on the other hand, carry the content and the meaning of a sentence and play a much more important role in identifying similar documents. Weighting techniques, such as the widely adopted tf–idf (term frequency–inverse document frequency), are used in order to assign weights to words based on their frequencies in the analyzed document corpus. Low values are assigned to widespread words, making their contribution to similarity calculations less important, and vice versa.

The same observation holds for the symbolic trajectories as all segments do not have the same discriminative power. Segments that are frequently traveled by the majority of trajectories are not very relevant to cluster formations. On the contrary, segments that are traveled by a small portion of trajectories play a key role in the formation of the cluster containing those trajectories. To account for this observation, we devise a segment weighting strategy by adapting the tf–idf weighting to the case of trajectory data.

We define the spatial segment frequency (ssf) to measure the importance of a road segment s in a trajectory T:

$$\mathrm{ssf}_{s,T} = \frac{n_{s,T} \cdot \mathrm{length}(s)}{\sum_{s' \in T} n_{s',T} \cdot \mathrm{length}(s')} \ . \tag{3.9}$$

$n_{s,T}$ being the number of occurrences of s in T ($n_{s,T} = 1$ most of the time as trajectories rarely visit a segment more than once) and $\mathrm{length}(s)$ its spatial length.

The inverse trajectory frequency (itf) measures the frequency of the segment s in the whole set of trajectories $\mathcal{T}$:

$$\mathrm{itf}_s = \log \frac{|\mathcal{T}|}{|\{T_i : s \in T_i\}|} \ . \tag{3.10}$$

$|\mathcal{T}|$ is the total number of trajectories in the set $\mathcal{T}$, and $|\{T_i : s \in T_i\}|$ the number of trajectories containing the segment s.

While inspecting the trajectory T, the weight attributed to the road segment s is the combination of both its ssf in the trajectory and its itf:

$$\omega_{s,T} = \text{ssf}_{s,T} \cdot \text{itf}_s \ . \tag{3.11}$$

Finally, to compare two trajectories T_i and T_j, we calculate their cosine similarity:

$$\text{similarity}(T_i, T_j) = \frac{\sum_{s \in \mathcal{S}} \omega_{s,T_i} \cdot \omega_{s,T_j}}{\sqrt{\sum_{s \in \mathcal{S}} \omega_{s,T_i}^2} \cdot \sqrt{\sum_{s \in \mathcal{S}} \omega_{s,T_j}^2}} \ . \tag{3.12}$$

The values of this similarity vary from 0 to 1 with 0 indicating totally independent trajectories and values close to 1 indicating very similar trajectories.

3.3.2 Trajectory Similarity Graph

We model the similarity relationships between trajectories using an undirected, weighted graph $\mathcal{G}_\mathcal{T} = (\mathcal{T}, \mathcal{E}_\mathcal{T}, \mathcal{W}_\mathcal{T})$. Each trajectory in $\mathcal{T}$ is mapped to a vertex in $\mathcal{G}_\mathcal{T}$. An edge between a pair of trajectories T_i and T_j exists if and only if similarity$(T_i, T_j) > 0$ (i.e. if there is at least one common road segment that both trajectories crossed), in which case the similarity is assigned as a weight to that edge. This concept of similarity graph is depicted in Figure 3.6, whereas the pseudo-code for generating the graph is given in Algorithm 2.

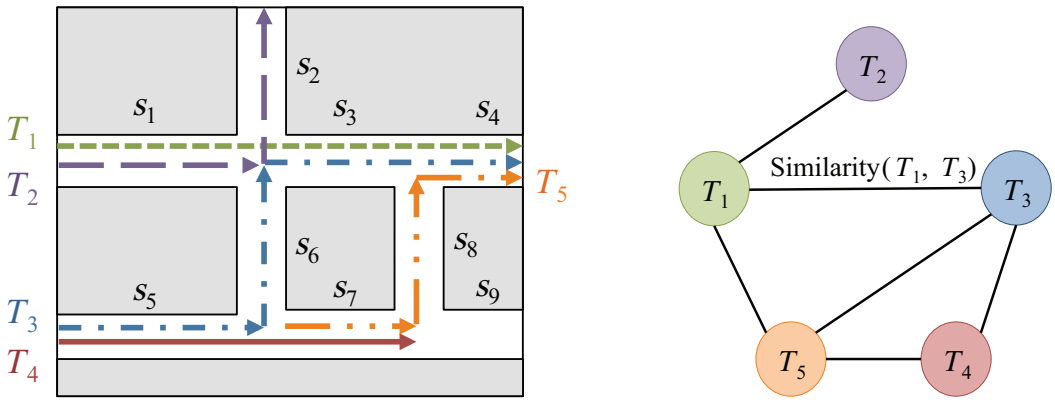


Figure 3.6 – Example of a trajectory similarity graph induced by five trajectories traveling on a road network. Each trajectory is represented by a vertex in the graph. An edge e links two trajectories if they share at least one road segment in common. Each edge is weighted with the similarity between the two trajectories it links together.

The main advantage of using this graph representation, besides being natural and easy to understand, is that it does not define an "artificial" similarity between totally incompatible trajectories. On the contrary, it emphasizes the

Algorithm 2 Generating the similarity graph.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{S})$: road network graph; $\mathcal{T}$: trajectory set.
Output: trajectory similarity graph $\mathcal{G}_\mathcal{T}$

```
1: create an empty undirected weighted graph G_T
2: for all T ∈ T do
3:     insert T as a vertex in G_T
4: end for
5: for all T_i, T_j ∈ T, i ≠ j do   ▷ each pair of trajectories is considered only once
6:     if similarity(T_i, T_j) > 0 then
7:         create an undirected edge e = (T_i, T_j) in G_T
8:         assign weight ω_e = similarity(T_i, T_j) to e
9:     end if
10: end for
```

fact that trajectories which do not share common road segments are independent and should not be "immediately" grouped in the same cluster since there is no similarity edge linking them. Note that this is the same principle used in spectral clustering where similarities between originally non-graph entities of different kinds are represented using graphs.

3.3.3 Clustering the Similarity Graph

To cluster the trajectory similarity graph, we use the implementation of hierarchical modularity-based clustering described in [102] (which is based on the observations and directives in [103]). A simplified pseudo-code of this implementation is given in Algorithm 3. First, the algorithm retrieves a partition of the vertices through modularity optimization[2] (line 1): the *Partition* procedure starts by considering the trivial partition where each vertex is in its own community and merges communities in a greedy fashion (i.e. each time, it merges the two communities that produce the maximum increase of modularity). The merging operation stops when no possible merge can be done without a degradation of the modularity, in which case the *Partition* procedure proceeds to a multi-level refinement step where members of different communities are interchanged in an attempt to further improve the modularity of the partition.

Once the initial partition is retrieved, the algorithm proceeds recursively to construct the hierarchy of communities (lines 4 through 14). For each community at a given level, the subgraph containing only the vertices of the community and the edges connecting them is isolated (line 7). This subgraph is partitioned separately (by invoking *Partition* as shown in line 8). The *TestSig* evaluates the significance of the found partition (by comparing its modularity

[2]Note that exact modularity maximization is NP-hard and is conducted mainly using heuristic algorithms [103].

to the modularity of partitions obtained on similar randomly generated graphs). If the partition is significant indeed, its communities are considered for further partitioning (lines 9-12); otherwise, it's rejected and the original community is retained. The recursion stops when none of the communities at level l yield a significant partition (line 14). The use of recursion to produce a hierarchy of clusters is mainly used to address the resolution issue in modularity-based clustering which tends to favor the formation of big clusters at the expense of smaller ones. However, as we will demonstrate later on, having a hierarchy of nested clusters when analyzing large trajectory datasets can be very helpful (Section 3.3.4.2).

Modularity-based graph clustering approaches are very popular and achieve good results in practice [98]. Nevertheless, we do not exclude the use of other graph clustering alternatives (e.g. spectral clustering [100]) if such techniques could yield better results.

Algorithm 3 Hierarchical modularity-based clustering.

Input: an undirected, weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$
Output: hierarchy of nested clusters of vertices

1: $C_1^{(1)}, ...C_K^{(1)} \leftarrow Partition(\mathcal{G})$ ▷ initial partition
2: $K_T \leftarrow K$ ▷ clusters counter
3: $l \leftarrow 1$ ▷ hierarchy level
4: **repeat**
5: $l \leftarrow l + 1$
6: **for all** cluster $C \in C_1^{(l-1)}, ..., C_K^{(l-1)}$ **do**
7: extract the sub-graph $\mathcal{G}_C$ of vertices belonging to C
8: $C_1^C, ...C_k^C \leftarrow Partition(\mathcal{G}_C)$
9: **if** $TestSig(C_1^C, ...C_k^C)$ **then**
10: $C_{K_T+1}^{(l)}, ..., C_{K_T+k}^{(l)} \leftarrow C_1^C, ...C_k^C$
11: $K_T \leftarrow K_T + k$
12: **end if**
13: **end for**
14: **until** no significant subdivision of level l can be found

3.3.4 Discussion

Here, we discuss various aspects (such as cluster representatives and exploration, etc.) related to our approach.

3.3.4.1 About Cluster Representatives

One common practice in cluster analysis is to appoint a representative for each of the discovered clusters. This helps provide the user with a means to quickly understand the general trend exhibited by the instances of the cluster he is exploring. In most cases, cluster representatives are either (i) elected

amongst the real members of the cluster (e.g. the instance that bears the most average resemblance to the other instances, which is the case in the NNCluster approach [10]), or (ii) artificially created by combining the features of all the members of the cluster (this is the case in the Partition-and-Group framework [58]).

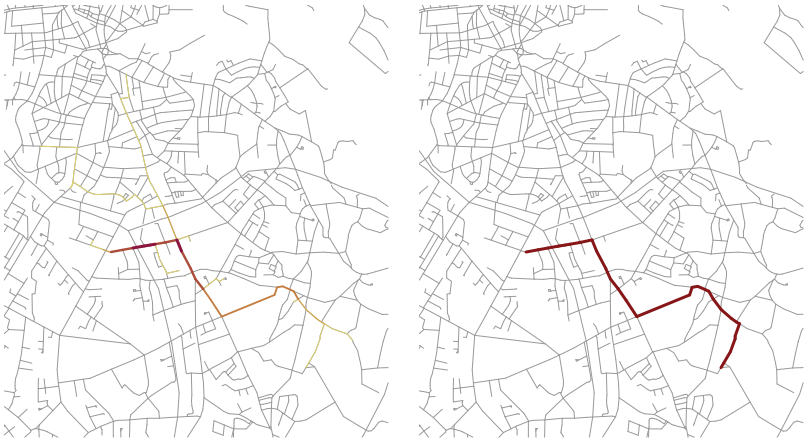
In our work, we do not explore the various aspects of defining and exploiting trajectory cluster representatives in full detail. Nevertheless, it is fairly easy to define representatives of the trajectory clusters discovered using our framework. One way to do so is to opt for an approach similar to the one presented in [10] and choose the trajectory that is the most similar to the other members of its cluster to be the representative.

Definition 10 (Representative Trajectory)**.** Given a cluster C of trajectories, the representative trajectory of C, denoted $RT(C)$, is defined as follows:

$$RT(C) = \underset{T \in C}{\operatorname{argmax}} \frac{1}{|C|} \sum_{T' \in C} \text{similarity}(T, T') \ . \tag{3.13}$$

Where $|C|$ is the number of trajectories in C.

Since we are establishing a hierarchy of nested clusters, the quality of the inspected cluster varies depending on its level in the hierarchy. Clusters in the higher levels tend to be coarse (especially when analyzing large datasets) while those in lower levels tend to be more refined. Thereupon, representative trajectories also tend to behave similarly and give a very coarse approximation of the movement pattern in high-level clusters and become more descriptive of the clusters' behavior at the lower levels. An example of a trajectory cluster and its representative trajectory is depicted in Figure 3.7.



(a) Trajectory cluster (b) Representative trajectory

Figure 3.7 – Example of a cluster (a) and its representative trajectory (b).

3.3.4.2 Exploring the Trajectory Clusters

In order to illustrate how the discovered hierarchy of trajectory clusters can be explored in practice, we present a case study conducted on a synthetic dataset generated using the Brinkhoff generator. The dataset is illustrated in Figure 3.8 and is composed of 10000 trajectories of varying lengths that traveled along the Oldenburg road network (this dataset will also be used in our experimental study presented in Section 3.3.5). It is obvious that visualizing the data isn't sufficient to discover meaningful knowledge about the flow dynamics and drivers' behavior. By regrouping similar individuals, clustering can make such trends and patterns clearly visible.

Figure 3.8 – Case study dataset composed of 10000 trajectories evolving in the Oldenburg road network. The road segments are color-coded in order to indicate their usage, varying from pale yellow (for less used segments) to dark red (for frequently traveled segments).

Applying our approach to this dataset results in a hierarchy of nested clusters that spans over seven levels. The most coarse level contains eight clusters only, whereas the most refined level contains 921 clusters. The hierarchical tree of the first two levels is depicted in Figure 3.9.

The eight trajectory clusters in the highest level of hierarchy are presented in Figure 3.10. Since these are very coarse clusters regrouping a considerable number of trajectories, most of them (e.g. clusters 1, 5, and 7) cover large areas of the road network and define some sort of "geographical clusters." Nevertheless, even at this level of detail, some trends and patterns start to be visible, which is the case in clusters 2 (Figure 3.10(b)), 3 (Figure 3.10(c)),

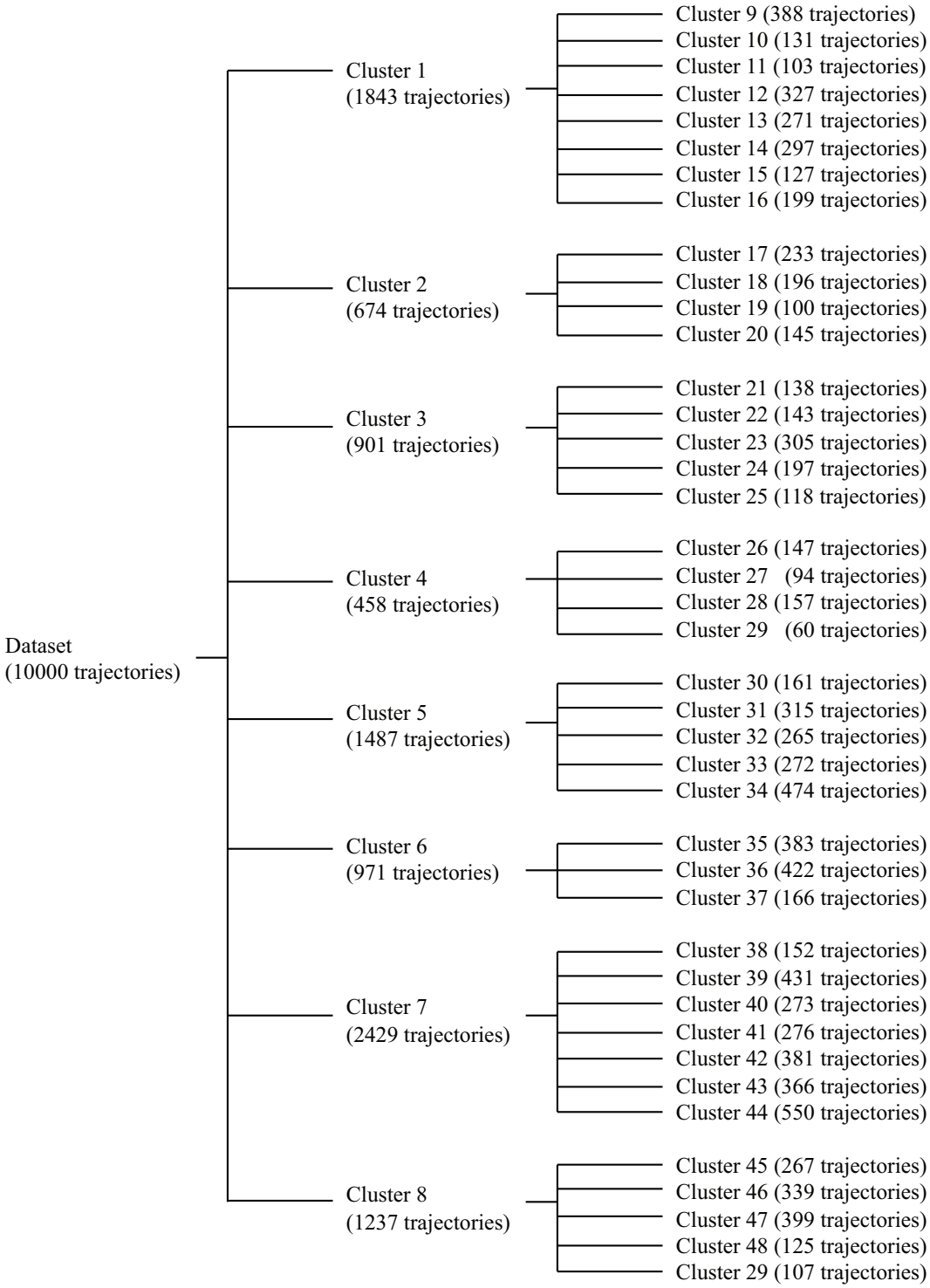


Figure 3.9 – The first two levels of the produced hierarchy of trajectory clusters.

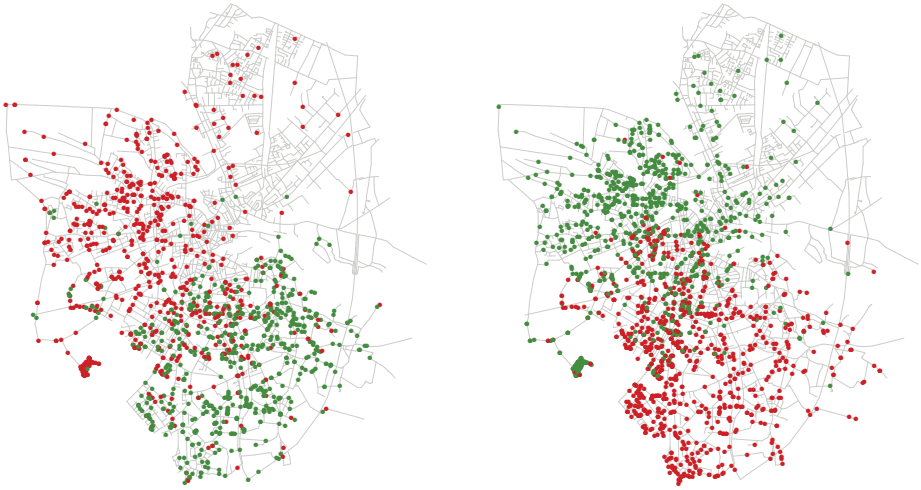
and 4 (Figure 3.10(d)). These clusters seem to be more space-confined and color-coding of the visited road segments reveals heavy usage of particular parts of the road network.

(a) Cluster 1 (1843 trajectories)
(b) Cluster 2 (674 trajectories)
(c) Cluster 3 (901 trajectories)
(d) Cluster 4 (458 trajectories)
(e) Cluster 5 (1487 trajectories)
(f) Cluster 6 (971 trajectories)
(g) Cluster 7 (2429 trajectories)
(h) Cluster 8 (1237 trajectories)

Figure 3.10 – The eight trajectory clusters retrieved in the most coarse level of the hierarchy.

By looking at Figure 3.10, we can notice that some clusters look very similar.

This is the case of clusters 2 (Figure 3.10(b)) and 3 (Figure 3.10(c)) as well as clusters 6 (Figure 3.10(f)) and 8 (Figure 3.10(h)). Let us analyze clusters 2 and 3 to try to understand why these clusters were not merged together and considered as one cluster. Figure 3.11 depicts the departure and arrival points of both clusters and shows that trajectories in cluster 2 moved from south to north, whereas trajectories regrouped in cluster 3 did the exact opposite. Therefore, the approach successfully separated the trajectories based on their travel direction, which supports the claim we made in Section 3.3.1. This is the same reason for which clusters 6 and 8 are also separated.



(a) Cluster 2 (b) Cluster 3

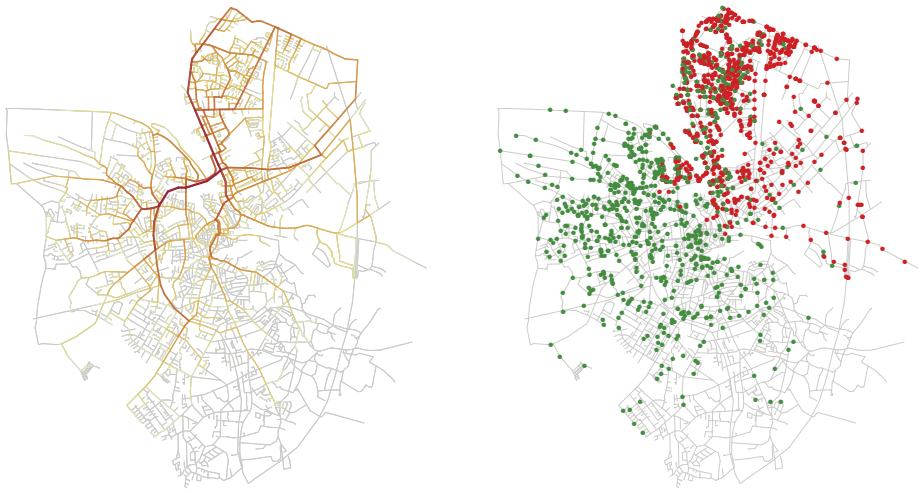
Figure 3.11 – Departure (green) and arrival (red) points of trajectories in clusters 2 and 3.

When dealing with large datasets, "flat" clustering (i.e. clustering that produces only one level of clusters) can result in an overwhelming number of clusters. Imagine for one instant the tedious task of analyzing the 921 clusters produced at the most refined level from the get-go. Our choice of using a hierarchical clustering algorithm was, in part, to counter this and provide a more convenient way of analyzing the data. The user can start with the highest level of hierarchy (containing a small number of clusters) to quickly understand the macro organization and the global movement patterns then focus on particular clusters he deems interesting and gradually reveal more details as he goes down the hierarchy.

To demonstrate this point, let us suppose that we are interested in exploring cluster 6 since we noticed that it presents an interesting pattern where trajectories seem to migrate from the west part to the north-eastern part of the road network (Figure 3.12). Consequently, we proceed to expanding this cluster into its children. The expansion results in three more refined clusters (Figure 3.13). In cluster 35 (Figure 3.13(a)), the start points of the trajectories

seem to be more concentrated in the central part of the road network and the arrival points more concentrated on the north-most part (Figure 3.13(b)). The original trend that we noticed in cluster 6 seems to come mainly from cluster 36 (Figure 3.13(c)) since departures in the latter are more located to the west and arrivals are scattered over the north-east (Figure 3.13(d)). The last child (cluster 37) seems to be less relevant than the other two (Figure 3.13(e)). However, it does regroup trajectories that are mostly concentrated in the north w.r.t. both their start and end points (Figure 3.13(f)). Each of these three clusters can be further refined and so forth. For conciseness's sake, we make do with what we already presented and do not proceed to further refinements.

Notice that in classical hierarchical clustering approaches that produce the whole dendrogram of the cluster merging operations, the choice of where to cut the dendrogram is left up to the user. However, contrary to these approaches, the algorithm used in our clustering step determines those cuts automatically, thus providing the user with a hierarchy containing a limited and reasonable number of levels to be explored. While this makes the exploration of clusters easier, it also limits the possible refinements to those considered as significant w.r.t. modularity.



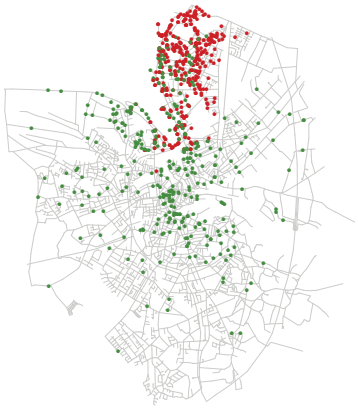
(a) Road usage (b) Departure and arrival points

Figure 3.12 – Trajectory cluster 6, considered for further refinement.

3.3.4.3 Time Complexity

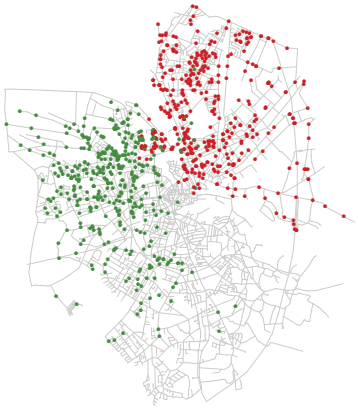
Let n be the number of trajectories in the dataset $\mathcal{T}$ and m the number of segments in the road network. In order to calculate the road segments' weights, a sequential, single pass over the trajectories is sufficient. Therefore, the complexity of the weight calculation step is, in the worst case scenario, $O(mn)$.

(a) Cluster 35 (road usage)



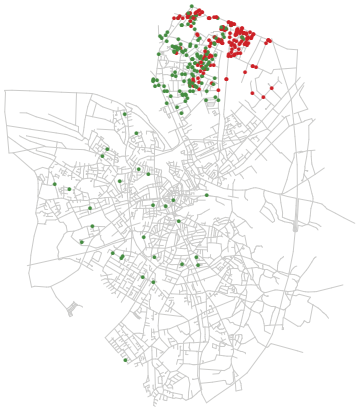
(b) Cluster 35 (departure and arrival points)

(c) Cluster 36 (road usage)



(d) Cluster 36 (departure and arrival points)

(e) Cluster 37 (road usage)



(f) Cluster 37 (departure and arrival points)

Figure 3.13 – Child clusters obtained through refinement of cluster 6.

Trajectories can be represented using the Vector Space Model [104] (i.e. each trajectory is a vector of size m representing the number of crossings on each road segment). Calculating the cosine similarity between two trajectories T_i and T_j is achieved, theoretically, in $O(m)$ time complexity. This complexity, however, is attained in the very improbable case where one of the two trajectories visited every single road segment in the network. In practice, calculating the similarity between the two trajectories T_i and T_j is achieved in $O(\max(|T_i|, |T_j|))$, $|T|$ being the number of segments in T.

Building the trajectory similarity graph requires $\frac{n(n-1)}{2}$ similarity calculations and is consequently done in $O(mn^2)$ time complexity. The resulting graph contains n vertices and at most $\frac{n(n-1)}{2}$ undirected edges. Therefore, the theoretical maximal complexity of the community detection algorithm used in our clustering phase is $O(n^3)$ (cf. [98]). Again, this is rarely observed in practice where the complexity is somewhere near $O(n^2 \log n)$.

3.3.4.4 Positioning With Respect to Existing Approaches

Similarly to [22], the similarity measure we proposed is founded on concepts borrowed from information retrieval and document clustering. However, instead of using a spatial adaptation of the Jaccard Index where all road segments contribute equally to the similarity calculation, we adopt a weighting strategy where each segment's relevance varies depending on its frequency in the analyzed trajectory dataset. Earlier experiments we conducted showed that the weighting technique enhanced clustering results.

We intentionally refrained from adopting distances that use cost functions in the road network (like in [71, 21, 10]) for the following reasons: (i) network-based distances can incur severe computational overheads especially when comparing trajectories on a segment per segment basis, and, more importantly, (ii) we wanted to emphasize on the fact that trajectories that do not share at least one road segment should be considered as completely irrelevant. Cost-based measures still assign a distance value to such trajectories.

The work on trajectory clustering that resembles our approach the most is the one presented in [59]. Although our approach and [59] both use graph-based clustering and propose a hierarchy of nested clusters for multi-level exploration, there are some key differences between the two approaches:

- The work in [59] is applied to free moving trajectories and considers the latter as sets of GPS points. The authors do not rely on an underlying network as a basis of similarity calculations (although their first filtering step can be seen as an attempt at discovering the structure of this network). By contrast, in our work we suppose that the underlying road network's structure is known in advance and we use it as a basis for trajectory representation and comparison.

- In [59], the clustering is applied to a set of characteristic GPS points in order to discover clusters representing regions of interest to which

trajectories are mapped in a later step. In our approach, the clustering is applied directly to the network-constrained trajectories in order to discover clusters of trajectories with similar behavior.

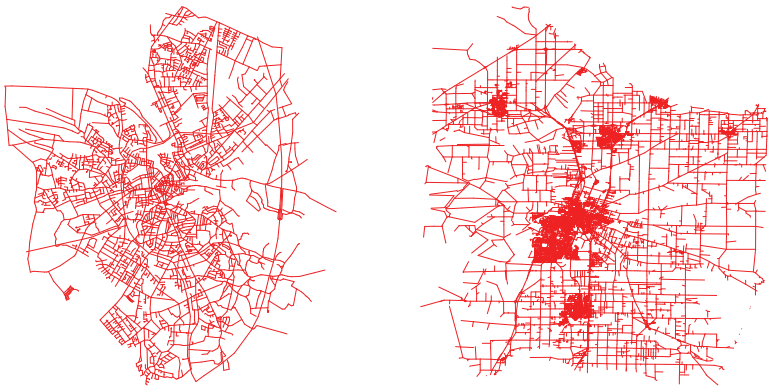
Finally, the framework we presented is based on a purely spatial similarity. Therefore, trajectories that move on the same road segments at different times are regrouped together in the same cluster. Earlier attempts at proposing spatiotemporal similarities for network-constrained trajectories [71, 21] felt counter-intuitive. This is mainly due to the fact that combining a spatial and a temporal similarity by means of addition or multiplication is somewhat unnatural. We believe that a more attractive approach to including time is the one proposed in [11]: the time epoch covered by the trajectories dataset can be divided into sub-epochs (in a fashion that portrays periods of interest such as morning departures and evening returns for example) and clustering can be conducted separately on these epochs.

3.3.5 Experimental Results

We present the experiments that we conducted on our trajectory clustering approach in this section.

3.3.5.1 Datasets

In order to test our approach and compare it to other algorithms, we used various synthetic datasets that we generated using the Oldenburg and San Joaquin road networks. The road network of Oldenburg is composed of 6105 vertices and 7035 undirected edges. The road network of San Joaquin is larger as it contains 18496 vertices and 24123 undirected edges. Both networks are depicted in Figure 3.14.



(a) Oldenburg (b) San Joaquin

Figure 3.14 – Road networks of (a) Oldenburg and (b) San Joaquin.

We use two types of datasets: (i) a large dataset, containing 10000 unlabeled trajectories moving along the Oldenburg road network, generated using the Binkhoff generator [6]; and (ii) nine labeled datasets containing randomly generated clusters of trajectories. As a reminder, the strategy (described in detail in Section 1.2.2.2) that we used to generate the latter works as follows. For each cluster of trajectories to be generated, a vertex is chosen randomly in the road network's graph. vertices that are reachable from the selected vertex within a given maximal distance are chosen to play the role of source vertices of the cluster. The same principle is applied to randomly choose target vertices. For each trajectory to be generated in the cluster, a source (respectively target) vertex is chosen among the source (respectively target) vertices. The trajectory is then generated following a near-shortest path linking the source to the target. The number of trajectories in each cluster is fixed randomly (between an upper and a lower bounds). We generate clusters to depict many scenarios of interactions such as clusters converging to a common destination, diverging from a common source, or inverted clusters (i.e. one cluster's departure is the arrival of the second cluster and vice versa). Such movement patterns and interactions among clusters may also appear randomly and naturally. Table 3.2 summarizes the characteristics of our nine labeled datasets. Each cluster of trajectories contains from 10 to 25 trajectories. We favor the use of the Oldenburg road network (used in 6 out of the 9 datasets) since its small size favors interactions and overlaps between clusters, which makes these clusters less trivial to discover.

Table 3.2 – Characteristics of the labeled datasets.

Dataset	Clusters	Trajectories	Road network
1	9	158	Oldenburg
2	10	163	Oldenburg
3	11	141	Oldenburg
4	6	86	Oldenburg
5	6	91	Oldenburg
6	6	110	Oldenburg
7	12	205	San Joaquin
8	11	190	San Joaquin
9	12	203	San Joaquin

3.3.5.2 Performance Criteria

When experimenting on the dataset generated using the Brinkhoff generator (the one containing 10000 trajectories moving along the Oldenburg road network), no ground-truth clustering for the trajectories is known beforehand (i.e. they are not labeled). Therefore, we try to compare the tested approaches by assessing the resemblance of the trajectories they regrouped into each cluster. To this end, we define the notion of intra-cluster overlaps. Given a set of trajectory

clusters $\mathcal{C}_\mathcal{T} = \{C_1, ...C_K\}$, the sum of average intra-cluster overlaps of $\mathcal{C}_\mathcal{T}$ is expressed according to Formula (3.14).

$$\mathcal{Q}(\mathcal{C}_\mathcal{T}) = \sum_{C \in \mathcal{C}_\mathcal{T}} \frac{1}{|C|} \sum_{T_i, T_j \in C} \text{overlap}(T_i, T_j) \ . \tag{3.14}$$

Where:

$$\text{overlap}(T_i, T_j) = \frac{\sum_{s \in T_i, s \in T_j} \text{length}(s)}{\sum_{s \in T_i} \text{length}(s)} \ . \tag{3.15}$$

The higher the value of the sum of intra-cluster overlaps, the better the formed clusters are since this internal criterion (i.e. a criterion based solely on the information provided to the clustering algorithm) indicates that trajectories regrouped together overlap and share more common road segments.

The trajectories in the other datasets are labeled. Consequently, we can evaluate the quality of the results using external criteria (i.e. criteria that use the information about the a priori knowledge of class labels) such as the purity and entropy [105] as well as the Adjusted Rand Index [106] of the clusters produced by the various approaches (a survey of other internal and external metrics that can be used to compare clusterings can be found in [107]).

Let X be a set (of size $|X|$) of observations and $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ and $\mathcal{C}' = \{C'_1, C'_2, ..., C'_l\}$ two partitions of X into sets of non-empty, disjoint clusters (without loss of generality, we suppose that $\mathcal{C}'$ denotes the ground-truth clusters that we will refer to hereafter as classes).

The purity of a given cluster C_i is the number of instances that are assigned to the most frequent class in the cluster, divided by the size of the cluster, formally expressed as in Formula (3.16).

$$P(C_i) = \frac{1}{|C_i|} \max_{1 \leq j \leq l} |C_i \cap C'_j| \ . \tag{3.16}$$

Where $|C_i|$ designates the number of members in the cluster C_i and $|C_i \cap C'_j|$ the number of common members between the cluster C_i and the class C'_j. The purity of the partition $\mathcal{C}$ is the weighted sum of its individual clusters' purities:

$$\text{purity} = \sum_{i=1}^{k} \frac{|C_i|}{|X|} P(C_i) \ . \tag{3.17}$$

Purity can be perceived informally as a measure of the precision of the discovered clusters. In general, a bad clustering results in a low value of purity (close to 0), whereas a perfect clustering results in a purity equal to 1.

Entropy measures the distribution of the original classes across the discovered clusters. The entropy of a cluster C_i is defined as follows:

$$E(C_i) = -\frac{1}{\log l} \sum_{j=1}^{l} \frac{|C_i \cap C'_j|}{|C_i|} \log \frac{|C_i \cap C'_j|}{|C_i|} \ . \tag{3.18}$$

The entropy of $\mathcal{C}$ is the sum of the weighted entropies of its clusters:

$$\text{entropy} = \sum_{i=1}^{k} \frac{|C_i|}{|X|} E(C_i) \ . \tag{3.19}$$

It is desirable to have clusters containing members belonging to a single class only (in which case the resulting entropy is equal to zero). A good clustering will produce low values of entropy and, vice versa, high values of entropy are generally an indicator of bad clustering.

The classic Rand Index [108] is also a commonly used measure to calculate the fraction of correctly classified (or misclassified) elements among all the elements with the exception that instead of counting single elements, it counts pairs of correctly classified ones. Hubert and Arabie [106] made an adjustment to this index by assuming a generalized hypergeometric distribution as a null hypothesis. The Adjusted Rand Index is the normalized difference between the original Rand Index and its expected value under the null hypothesis:

$$\mathcal{R}_{\text{adj}}(\mathcal{C}, \mathcal{C}') = \frac{\sum_{i=1}^{k} \sum_{j=1}^{l} \binom{|C_i \cap C'_j|}{2} - t_3}{\frac{1}{2}(t_1 + t_2) - t_3} \ . \tag{3.20}$$

Where $t_1 = \sum_{i=1}^{k} \binom{|C_i|}{2}$, $t_2 = \sum_{j=1}^{l} \binom{|C'_j|}{2}$, and $t_3 = \frac{2t_1 t_2}{|X|(|X|-1)}$.

The Adjusted Rand Index has a value of 1 in case of identical partitions and 0 in presence of two totally independent partitions.

3.3.5.3 Characteristics of the Generated Trajectory Similarity Graphs

Applying our approach to the Brinkhoff dataset results in a trajectory similarity graph composed of 10000 vertices and 3272543 edges. The graph has a density of 0.065 which makes it quite sparse, yet it has a diameter of 4 which suggests the presence of a small world effect (similarly to what was observed in other types of trajectory networks in the study presented in [85]). The least connected trajectory (i.e. vertex) in this graph has only two edges linking it to other trajectories while the most connected has a degree of 2524. The degree distribution in this graph is depicted in Figure 3.15 which shows a complex and non-standard distribution that does not seem to abide by a particular law especially those (such as the power law) that favor the use of modularity-based clustering over other graph clustering approaches.

The characteristics of the trajectory similarity graphs generated over the labeled datasets are regrouped in Table 3.3. Here the graphs are more dense since they are composed solely of well-defined (ground-truth) clusters and since, additionally, some of the clusters overlap. All the graphs have a rather small diameter except the graph obtained on the third dataset in whose case the clusters are mostly separated and do not overlap as frequently as in other datasets.

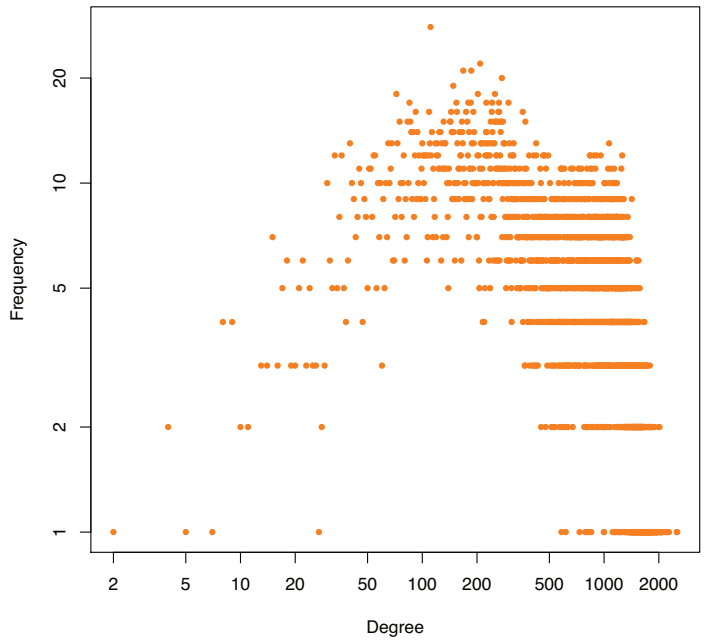


Figure 3.15 – Degree distribution in the trajectory similarity graph generated from the Brinkhoff dataset: each point represents the number of vertices having a given degree. The distribution does not seem to follow any particular law.

3.3.5.4 Comparison with the NNCluster Baseline Algorithm

We now compare our approach to the baseline algorithm proposed in [10], which consists in using agglomerative hierarchical clustering in combination with a network-based distance measure to cluster trajectories (for simplicity, we refer to this approach as NNCluster baseline from now on). However, our own implementation of the NNCluster baseline algorithm differs from the original one described in [10] on the following points:

- In [10], the authors use a custom linkage strategy: a representative trajectory is elected for each cluster (the representative trajectory is the one minimizing the sum of distances to the other members) then the two clusters with the most similar representatives are merged. Instead of this custom linkage, we use the "more standard" average linkage as the cluster-merging criterion (partially in order to avoid implementation errors) which should, we believe, produce fairly similar results.
- After generating the dendrogram of clusters, the algorithm in [10] uses the Davies-Bouldin (DB) Index [91] to report only a single clustering result (the level minimizing the DB index). However, in our experimental study we keep the entire dendrogram in order to be able to compare results

Table 3.3 – Characteristics of the generated trajectory similarity graphs resulting from the labeled datasets.

Dataset	Number of vertices	Number of edges	Vertex degrees min.	avg.	max.	Density	Diameter
1	158	3334	23	42.2	65	0.27	4
2	163	2714	15	33.3	62	0.21	3
3	141	1767	12	25.06	47	0.18	10
4	86	1309	13	30.44	49	0.36	3
5	91	1134	4	24.92	36	0.28	3
6	110	2096	16	38.11	73	0.35	2
7	205	6053	17	59.05	111	0.29	5
8	190	5622	18	59.18	89	0.31	5
9	203	7963	42	78.45	151	0.39	3

reached by both our approach and the NNCluster baseline algorithm for the same number of clusters.

We point out that the reason for which we do not compare our approach directly with the NNCluster algorithm is that the latter introduces a parameter k which requires fine tuning upon which clustering quality depends.

We start by comparing the performances of both approaches on the Brinkhoff dataset. On this dataset, our approach discovered a hierarchy of clusters that spans over seven levels. The highest level contains only eight clusters while the down-most level contains up to 921 clusters. This further confirms our claim in Section 3.3.4 about exploring large datasets of trajectories where flat clustering can still produce an important number of clusters that can be hard to explore.

We compared the sums of intra-cluster overlaps scored by the two approaches for each of those levels, the results are shown in Table 3.4. As expected, the overlap increases, for both approaches, as we go down the hierarchy of clusters since the trajectories initially regrouped in fewer, more coarse clusters get separated in more precise clusters. The results also show that our algorithm outperforms the NNCluster baseline algorithm and produces more compact clusters where the trajectories are more similar and overlapping.

We next evaluate both approaches on our labeled datasets. First, we compare performances achieved at the highest level of aggregation: we split the NNCluster's dendrogram at the level that contains the same number of clusters as the first level of hierarchy discovered by our approach (which is considered as the optimal level w.r.t modularity).

As can be seen from the results reported in Table 3.5 and Table 3.6, in almost all cases our approach resulted in a number of clusters that is inferior to the number of ground-truth clusters. The main reason behind this is the presence of significant cluster interactions (e.g. in case of clusters converging to a common destination through the same parts of the network) that the approach was capable of detecting. This behavior can also be due to one of the

Table 3.4 – Modularity-based clustering vs. NNCluster baseline : Sum of average intra-cluster overlaps.

Hierarchical level	Number of clusters	Intra-cluster overlaps	
		NNCluster Baseline	Modularity
1	8	144	**665**
2	41	446	**1843**
3	143	847	**3239**
4	413	1678	**4543**
5	759	2307	**5359**
6	903	2529	**5576**
7	921	2554	**5596**

limitations of modularity-based clustering algorithms which tend to favor the discovery of big communities at the expense of smaller ones [109]. Nevertheless, we do overcome this limitation since we are using a hierarchical algorithm, and ground-truth clusters are often retrieved in the subsequent levels of the hierarchy. The obtained results also show that our approach outperformed the NNCluster baseline algorithm in most of the datasets, suggesting that in case of "over-merging" clusters, it produces better, more relevant clusters.

We also compare both approaches for the best match possible: in the hierarchy of clusters produced by our modularity-based approach, we chose the level that resembles the most the original clusters in the dataset and we compare both approaches for this same number of clusters. As shown in Table 3.7 and Table 3.8, our approach succeeds in retrieving the same original clusters in five out of the nine datasets while the NNCluster baseline retrieves only three. Our approach also scores better on the other datasets where the exact clusters are not retrieved.

3.3.5.5 Comparison with Spectral Clustering

We now study the effect of varying the graph clustering algorithm. To do so, we compare the modularity-based algorithm we used in our clustering step to spectral clustering [100]. In the spectral clustering implementation we used, eigenvectors are extracted from the graph's Laplacian and are used to conduct a k-means clustering in order to partition the graph's vertices.

We compare the values of external indicators (i.e. purity, entropy, and adjusted Rand index) achieved by both algorithms on the 9 labeled datasets described earlier. The comparison is based on the first level of hierarchy as well as the best-match level (w.r.t. the original classes) reported by modularity-based clustering. The number of clusters discovered by the modularity-based algorithm is used as the k parameter in spectral clustering. Since the spectral clustering implementation we used is based on k-means (whose results vary

Table 3.5 – Modularity-based clustering vs. NNCluster baseline : Adjusted Rand Index for the top-most level of the hierarchy of clusters.

	Discovered	Adjusted Rand Index	
Dataset	clusters	NNCluster Baseline	Modularity
1	6 (9)	0.697	**0.732**
2	8 (10)	0.818	**0.832**
3	8 (11)	0.681	**0.682**
4	5 (6)	**0.725**	**0.725**
5	5 (6)	**0.764**	**0.764**
6	6 (6)	**1**	**1**
7	6 (12)	**0.588**	0.505
8	8 (11)	0.758	**0.763**
9	5 (12)	0.370	**0.502**

Table 3.6 – Modularity-based clustering vs. NNCluster baseline : Entropy and Purity for the top-most level of the hierarchy of clusters.

	Discovered	Purity		Entropy	
Dataset	clusters	NNCluster Baseline	Modularity	NNCluster Baseline	Modularity
1	6 (9)	**0.753**	0.722	0.208	**0.193**
2	8 (10)	0.816	**0.840**	0.116	**0.109**
3	8 (11)	0.677	**0.738**	0.206	**0.186**
4	5 (6)	**0.802**	**0.802**	**0.166**	**0.166**
5	5 (6)	**0.835**	**0.835**	**0.150**	**0.150**
6	6 (6)	**1**	**1**	**0**	**0**
7	6 (12)	0.585	**0.600**	**0.288**	0.321
8	8 (11)	0.763	**0.768**	**0.153**	**0.153**
9	5 (12)	0.483	**0.493**	0.424	**0.362**

Table 3.7 – Modularity-based clustering vs. NNCluster baseline : Adjusted Rand Index for the best-match level of the hierarchy of clusters.

	Discovered	Adjusted Rand Index	
Dataset	clusters	NNCluster Baseline	Modularity
1	9 (9)	0.902	**1**
2	10 (10)	0.881	**1**
3	11 (11)	0.764	**0.873**
4	6 (6)	**1**	**1**
5	6 (6)	**1**	**1**
6	6 (6)	**1**	**1**
7	14 (12)	0.618	**0.961**
8	12 (11)	0.921	**0.971**
9	10 (12)	0.752	**0.889**

Table 3.8 – Modularity-based clustering vs. NNCluster baseline : Purity and entropy for the best-match level of the hierarchy of clusters.

	Discovered	Purity		Entropy	
Dataset	clusters	NNCluster Baseline	Modularity	NNCluster Baseline	Modularity
1	9 (9)	0.924	**1**	0.062	**0**
2	10 (10)	0.902	**1**	0.059	**0**
3	11 (11)	0.823	**0.915**	0.113	**0.064**
4	6 (6)	**1**	**1**	**0**	**0**
5	6 (6)	**1**	**1**	**0**	**0**
6	6 (6)	**1**	**1**	**0**	**0**
7	14 (12)	0.712	**1**	0.185	**0**
8	12 (11)	0.942	**1**	0.038	**0**
9	10 (12)	0.778	**0.872**	0.136	**0.075**

depending on the initialization of cluster centers), we configure it to report the best execution among 100 random starts.

Results for the top-level of hierarchy are reported in Table 3.9 and Table 3.10 whereas those for the best-match level are reported in Table 3.11 and Table 3.12. As can be seen from those results, both spectral clustering and modularity-based clustering performed equally on the majority of the datasets with identical clustering results on six out of the nine datasets in both levels. However, on the remaining datasets, modularity-based clustering had a slight edge over spectral clustering and yielded better results.

Contrary to the modularity-based algorithm we use in our clustering step, spectral clustering produces a one-level flat clustering where the number of clusters is fixed manually. Consequently, it is up to the user to define the suitable number of clusters (e.g. using classic approaches such as the elbow method). In our case, the number of clusters in each level of the hierarchy of nested clusters is automatically determined based on the modularity criterion.

Table 3.9 – Modularity-based clustering vs. spectral clustering : Adjusted Rand Index for the top-most level of the hierarchy of clusters.

	Discovered	Adjusted Rand Index	
Dataset	clusters	Spectral	Modularity
1	6 (9)	**0.732**	**0.732**
2	8 (10)	0.714	**0.832**
3	8 (11)	**0.682**	**0.682**
4	5 (6)	**0.725**	**0.725**
5	5 (6)	0.730	**0.764**
6	6 (6)	**1**	**1**
7	6 (12)	**0.505**	**0.505**
8	8 (11)	**0.763**	**0.763**
9	5 (12)	0.464	**0.502**

Table 3.10 – Modularity-based clustering vs. spectral clustering : Purity and entropy for the top-most level of the hierarchy of clusters.

	Discovered	Purity		Entropy	
Dataset	clusters	Spectral	Modularity	Spectral	Modularity
1	6 (9)	**0.722**	**0.722**	**0.193**	**0.193**
2	8 (10)	0.810	**0.840**	0.148	**0.109**
3	8 (11)	**0.738**	**0.738**	**0.186**	**0.186**
4	5 (6)	0.**802**	**0.802**	**0.166**	**0.166**
5	5 (6)	0.824	**0.835**	0.178	**0.150**
6	6 (6)	**1**	**1**	**0**	**0**
7	6 (12)	**0.600**	**0.600**	**0.321**	**0.321**
8	8 (11)	**0.768**	**0.768**	**0.153**	**0.153**
9	5 (12)	0.473	**0.493**	0.378	**0.362**

Table 3.11 – Modularity-based clustering vs. spectral clustering : Adjusted Rand Index for the best-match level of the hierarchy of clusters.

	Discovered	Adjusted Rand Index	
Dataset	clusters	Spectral	Modularity
1	9 (9)	**1**	**1**
2	10 (10)	**1**	**1**
3	11 (11)	0.802	**0.873**
4	6 (6)	**1**	**1**
5	6 (6)	0.974	**1**
6	6 (6)	**1**	**1**
7	14 (12)	**0.961**	**0.961**
8	12 (11)	0.942	**0.971**
9	10 (12)	**0.889**	**0.889**

Table 3.12 – Modularity-based clustering vs. spectral clustering : Entropy and purity for the best-match level of the hierarchy of clusters.

	Discovered	Purity		Entropy	
Dataset	clusters	Spectral	Modularity	Spectral	Modularity
1	9 (9)	**1**	**1**	**0**	**0**
2	10 (10)	**1**	**1**	**0**	**0**
3	11 (11)	0.837	**0.915**	0.106	**0.064**
4	6 (6)	**1**	**1**	**0**	**0**
5	6 (6)	0.989	**1**	0.0233	**0**
6	6 (6)	**1**	**1**	**0**	**0**
7	14 (12)	**1**	**1**	**0**	**0**
8	12 (11)	0.963	**1**	0.021	**0**
9	10 (12)	**0.872**	**0.872**	**0.075**	**0.075**

3.4 Clustering Road Segments

In this section, we move on to study the road segment clustering problem that we introduced in Section 3.1. To do so, we extend our trajectory clustering approach to the case of segments: we define a similarity measure to evaluate the resemblance between road segments (Section 3.4.1) and, using the calculated similarities, we build a segment similarity graph that we cluster using the same clustering algorithm that we used earlier (Section 3.4.2).

The approach's time complexity as well as some pointers on how to explore road segment clusters are discussed in Section 3.4.3 whereas our experiments on segment clustering are reported in Section 3.4.4.

3.4.1 Road Segment Similarity

This time, we consider each road segment as the bag-of-trajectories that visited it (i.e. $\forall s \in \mathcal{S}, s \equiv \{T \in \mathcal{T} : s \in T\}$). In order to compare two road segments s_i and s_j, one can simply observe how often they co-appear in trajectories (i.e. calculate $|\{T \in \mathcal{T} : s_i \in T \wedge s_j \in T\}|$). The larger the number of concomitant appearances of both segments is, the more they are considered similar. However, different trajectories do not hold the same discriminative power when it comes to characterizing the similarity between road segments they visit: a lengthy trajectory that travels along a considerable number of road segments is not very informative when measuring the similarity between two segments in particular and, vice versa, short trajectories are highly relevant to the formation of the cluster that contains the segments they visit.

We account for this observation by devising a weighting strategy (inspired by tf-idf) where the contribution of each trajectory is proportional to its length. The weight $\omega_{T,s}$ assigned to trajectory T while inspecting a road segment s is expressed in Formula (3.21):

$$\omega_{T,s} = \frac{n_{s,T}}{\sum_{T' \in \mathcal{T}} n_{s,T'}} \cdot \log \frac{|\mathcal{S}|}{|s' \in \mathcal{S} : s' \in T|} \ . \tag{3.21}$$

The first part in this weight calculates the contribution of T to the segment s by calculating the ratio between the number of appearances $n_{s,T}$ of s in T and the total number of appearances of s in the whole dataset of trajectories $\mathcal{T}$. Since multiple visits of a same road segment are very rare, this part is often equal to $\frac{1}{|\{T \in \mathcal{T} : s \in T\}|}$. The second part evaluates the importance of the trajectory across the whole set of road segments: the more segments a trajectory visits, the less important it becomes and vice versa.

We use a cosine similarity to measure the similarity between two road segments s_i and s_j as expressed in Formula (3.22):

$$\text{similarity}(s_i, s_j) = \frac{\sum_{T \in \mathcal{T}} \omega_{T,s_i} \cdot \omega_{T,s_j}}{\sqrt{\sum_{T \in \mathcal{T}} \omega_{T,s_i}^2} \cdot \sqrt{\sum_{T \in \mathcal{T}} \omega_{T,s_j}^2}} \ . \tag{3.22}$$

3.4.2 Building and Clustering the Road Segment Similarity Graph

Similarly to what we did with trajectories, we model the similarity relationships between road segments using an undirected, weighted graph $\mathcal{G}_{\mathcal{S}} = (\mathcal{S}, \mathcal{E}_{\mathcal{S}}, \mathcal{W}_{\mathcal{S}})$. Each road segment in $\mathcal{S}$ is mapped to a vertex in $\mathcal{G}_{\mathcal{S}}$. An edge between a pair of segments s_i and s_j exists if and only if similarity$(s_i, s_j) > 0$ (i.e. if there is at least one common trajectory that crossed both segments). In this case, the similarity is assigned as a weight to that edge. An example of a segment similarity graph is depicted in Figure 3.16.

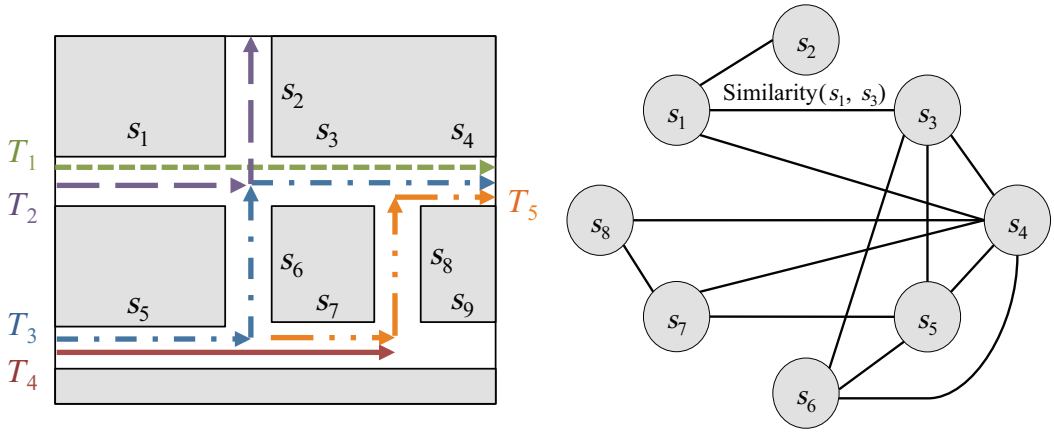


Figure 3.16 – Example of a segment similarity graph resulting from eight road segments being visited by five trajectories. Vertices represent the studied road segments while weighted edges indicate the presence and strength of the similarity between pairs of segments.

To cluster the segment similarity graph and produce a hierarchy of nested segment clusters, we use the exact same algorithm that we described in Section 3.3.3.

3.4.3 Discussion

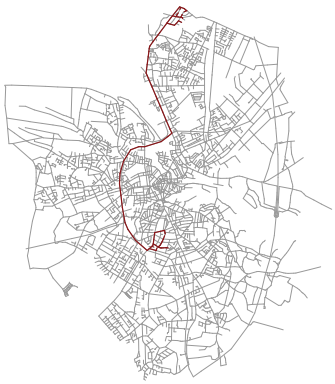
We discuss the exploration of road segment clusters in Section 3.4.3.1 then we give the algorithmic complexity of our approach in Section 3.4.3.2.

3.4.3.1 Exploring Road Segment Clusters with the Help of Trajectory Clusters

Segment clusters are not as easy to grasp and understand as trajectory clusters. Even though it is possible to try and explore these clusters as standalone clusters, we recommend involving the trajectory clusters in the process. Cross-comparing both types of clusters can reveal interesting information about flow dynamics and yield a better interpretation of the clusters. To show how this

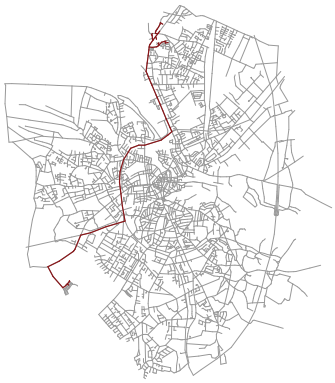
can be done, we use a small synthetic dataset containing 85 trajectories that moved along the Oldenburg road network and visited a total of 485 distinct road segments. We manually partitioned the trajectories into five clusters (depicted in Figure 3.17) that we consider hereafter as the ground-truth clusters.

(a) Cluster 1 (14 trajectories)



(b) Cluster 2 (19 trajectories)

(c) Cluster 3 (20 trajectories)



(d) Cluster 4 (20 trajectories)

(e) Cluster 5 (12 trajectories)

Figure 3.17 – Ground-truth trajectory clusters in the dataset.

We applied both trajectory and road segment clustering to this dataset. The resulting trajectory clusters' hierarchy contains only three trajectory clusters (ground-truth clusters 2 and 3 were considered as part of a same cluster because of their strong resemblance; the same occurs with clusters 4 and 5). Nevertheless, all the ground-truth clusters are retrieved correctly (some of them are even refined) in the following levels. Road segment clustering, on the other hand, resulted in a hierarchy of six levels with only 4 segment clusters in the highest level and 41 clusters in the most detailed level.

In order to give segment clusters more context, a segment cluster can be interpreted based on the trajectory groups that interacted with it. Figure 3.18 shows the crossed matrix of the second level trajectory clusters (reported on the rows) and the second level road segment clusters (on the columns) and gives an idea about the sizes of the clusters and how clusters of one type interact with those of the other type.

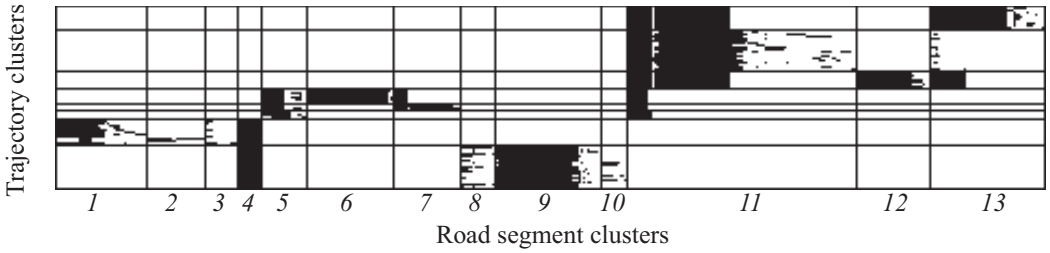


Figure 3.18 – Crossed matrix of the trajectory clusters (rows) and road segment clusters (columns). Each cell gives an idea about the interaction between the corresponding trajectory and segment clusters: the more black dots the cell contains, the more trajectories in the trajectory cluster cross segments belonging to the segment cluster.

The crossed matrix does indeed reveal some interesting patterns and interactions. For instance, the fourth segment cluster is explored exclusively by two trajectory clusters. Visualizing both this segment cluster and its visiting trajectory clusters (Figure 3.19) shows that the segment cluster plays the role of a hub for these two groups of trajectories that converge to it from two different areas in order to travel to two different destinations.

Crossing trajectory clusters and segment clusters is flexible and can be done at various levels of the hierarchies of both cluster types. However, it is totally up to the user to decide the relevance of the crossed clusters. The case of the eleventh segment cluster (cf. Figure 3.18) illustrates this point: this segment cluster is very interesting since it interacts with six trajectory clusters. However, it is obvious that the segment cluster contains a lot of "noise" segments which is expressed by the considerable amount of white space in the six first rows (representing the trajectory clusters) in the column representing the cluster in the crossed matrix. Consequently, drawn conclusions about the interactions between the clusters will not be very reliable. A wiser alternative would be to

(a) Hub segment cluster (b) Trajectory cluster 7 (c) Trajectory cluster 8

Figure 3.19 – A segment cluster (a) playing the role of a hub for two different trajectory clusters ((b) and (c)) that borrow it to travel to two separate destinations.

study the interactions between the (more refined) subclusters of this segment cluster with the six trajectory clusters it interacts with. This whole idea of studying interactions between trajectory and road segment clusters is the main motivation behind the work we present in Chapter 4.

3.4.3.2 Time Complexity

The complexity of our road segment clustering approach can be deduced by analogy to our trajectory clustering. With n being the number of trajectories in $\mathcal{T}$ and m the number of road segments in $\mathcal{S}$, the complexity of building the similarity graph is $O(nm^2)$ and clustering it is theoretically done in $O(m^3)$ ($O(m^2 \log m)$ in practice). The complexity is therefore mainly dependent on the road network's size, which can be prohibitive when dealing with very large road networks.

3.4.4 Experimental Results

In this experimental study, we compare modularity-based clustering and spectral clustering in the case of road segments.

3.4.4.1 Experimental Setting

We compare the performances of both algorithms on five synthetic datasets (cf. Table 3.13) produced with the Brinkhoff generator [6] using the Oldenburg road network. Each dataset contains 100 trajectories visiting a varying number of road segments.

Since no ground-truth clustering can be determined for the road segments in the used datasets, external metrics such as purity and entropy cannot be used to evaluate the results. Therefore, we evaluate the performance of each algorithm by measuring the sum of average intra-cluster overlaps of the road

Table 3.13 – Characteristics of the five synthetic datasets.

Dataset	Number of segments	Number of edges in the similarity graph
1	2562	79811
2	2394	100270
3	2587	110095
4	2477	87023
5	2348	80659

segment partition $\mathcal{C}_\mathcal{S}$ it produces. In the case of road segment clustering, the sum of average intra-cluster overlaps is expressed according to Formula (3.23):

$$\mathcal{Q}(\mathcal{C}_\mathcal{S}) = \sum_{C \in \mathcal{C}_\mathcal{S}} \frac{1}{|C|} \sum_{s_i, s_j \in C} \frac{|\{T \in \mathcal{T} : s_i \in T \wedge s_j \in T\}|}{|\{T \in \mathcal{T} : s_i \in T \vee s_j \in T\}|} \,. \tag{3.23}$$

$|C|$ is the number of segments in cluster C, $|\{T \in \mathcal{T} : s_i \in T \wedge s_j \in T\}|$ is the number of trajectories that visited both road segments s_i and s_j while $|\{T \in \mathcal{T} : s_i \in T \vee s_j \in T\}|$ is the number of trajectories that traveled along at least one of them.

3.4.4.2 Characteristics of the Generated Road Segment Similarity Graphs

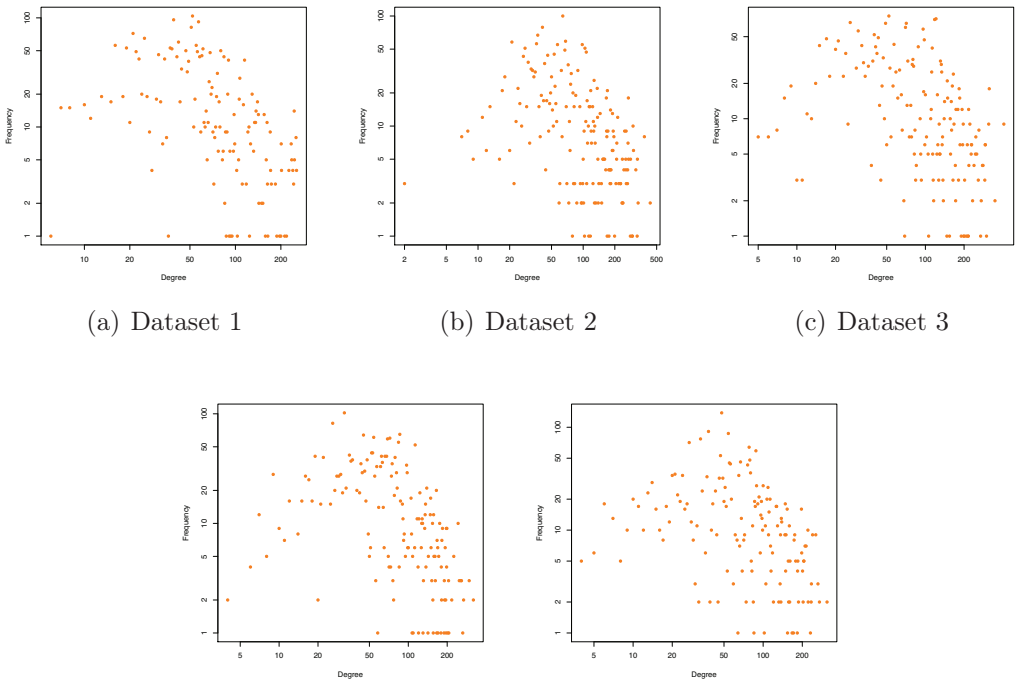
The main characteristics of the road segment similarity graphs generated from our five datasets are charted in Table 3.14. All the graphs are very sparse yet have rather small diameters. The degree distributions of these graphs are represented in Figure 3.20. Here again, the distribution is quite complex and no particular law is apparent.

Table 3.14 – Characteristics of the generated road segment similarity graphs.

	Number	Number	Vertex degrees				
Dataset	of vertices	of edges	min.	avg.	max.	Density	Diameter
1	2562	79811	6	62.3	254	0.02	11
2	2394	100270	2	83.77	434	0.04	8
3	2587	110095	5	85.11	408	0.03	8
4	2477	87023	4	70.26	319	0.03	10
5	2348	80659	4	68.7	310	0.03	9

3.4.4.3 Results of the Comparison Between Modularity-Based Clustering and Spectral Clustering

Since the highest level of hierarchy produced by modularity-based clustering is considered as the optimal clustering (w.r.t. modularity), we compare



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

(d) Dataset 4 (e) Dataset 5

Figure 3.20 – Distribution of vertex degrees in the road segment similarity graphs.

both algorithms based on the same number of clusters contained at this level. Table 3.15 shows that spectral clustering outperformed modularity-based clustering. We suspect that this is mainly due to the complex nature of the degree distributions in the road segment similarity graphs which results in situations where modularity-based is not advantaged compared to other approaches. The results consequently suggest that, contrary to the case of trajectory clustering, it might be preferable to use spectral clustering to cluster the graph of road segment similarity. However, as discussed earlier in Section 3.4.3.1, the biggest drawback when it comes to road segment clustering is the lack of context of the produced clusters. This is especially true since, contrary to approaches such as [8], segments regrouped together are not necessarily contiguous and can belong to different parts of the road network. Therefore, the relevance of relying solely on interior indexes (such as the sum of average intra-cluster overlaps used here) in order to evaluate the produced partitions' quality is yet to be proven.

Since the results that we obtained on road segment clustering are not as promising as those in the case of trajectories and since they do not benefit from the same ease of interpretation, we refrain from further experimentations and proceed, in the following chapter, to the presentation of an alternative approach based on simultaneous co-clustering of trajectories and road segments in order to retrieve automatically-correlated clusters of both types.

Table 3.15 – Results achieved by modularity-based clustering and spectral clustering on the road segment datasets.

	Number of	Intra-cluster overlaps	
Dataset	discovered clusters	Spectral	Modularity
1	23	**685.82**	657.20
2	21	**556.22**	524.46
3	20	**623.21**	561.09
4	22	**647.56**	594.76
5	26	**684.81**	666.24

3.5 Conclusions

In this chapter, we explored the problem of clustering network-constrained trajectory data. Clustering trajectories of moving objects which are freely moving in an Euclidean space was studied comprehensively with different proposals and formulations. However, in most real applications, moving objects (such as vehicles, airplanes, etc.) are restricted by an underlying network (road network, aerial corridors, etc.). The constraints imposed by the presence of such networks play an essential and key role in determining the resemblance between moving objects. This case of clustering trajectory data in the presence of network constraints attracted attention only recently and was understudied in comparison with the free movement case.

We formalized two clustering problems based on trajectory data in road network environments: (i) the network-constrained trajectories clustering problem where we aim to retrieve groups of trajectories with similar motion patterns, and (ii) the road segment clustering problem where we desire to retrieve clusters of segments that are frequently visited by the same moving objects.

We then studied the constrained trajectories clustering problems. To do so, we started by defining a new similarity measure to compare trajectories based on co-occurrences of road segments. We then modeled interactions between trajectories in the form of a similarity graph that we partitioned using modularity-based community detection in order to discover clusters of similar trajectories.

This graph-based approach presents a set of attractive features. First, the approach is non-parametric and, consequently, does not require fine tuning contrary to existing approaches that use density-based clustering (e.g. [79, 58, 55, 8]) where the quality of the resulting partitions varies greatly depending on parameter values. Secondly, instead of reporting one flat level of clusters, our approach reports a hierarchy of nested trajectory clusters. This hierarchy can be very useful when exploring clusters identified in large trajectory datasets: the user can start with a limited set of coarse clusters to rapidly understand the general movement trends then proceed to inspect more refined clusters by

means of successive zooms and cluster expansions. Our proposal, however, is not flawless. One of its drawbacks is sensitivity to noise since the clustering algorithm we used does not filter outlier trajectories. The sensitivity to noise was not studied in the scope of this work and can be considered in future work.

Results achieved in our experimental study were promising and encouraged us to extend our approach to the road segment clustering problem. Surprisingly, the achieved results in this second case were less convincing than the trajectory case. The retrieved road segment clusters lacked context and were hard to interpret on their own. One potential solution to this issue is to also include trajectory clusters and use them to interpret segment clusters they interact with. However, this cross-analysis is left up to the user who should manually decide on the relevance of the crossed clusters and the obtained results.

This last drawback is the main motivation for the work we will present in the next chapter where we will study the problem of co-clustering network-constrained trajectory data simultaneously based on trajectories and road segments.

Chapter 4

Co-Clustering Network-Constrained Trajectory Data

In the previous chapter, we made a first attempt at clustering network-constrained trajectory data by adopting a graph-based approach. We started by defining two separate problems. First we defined network-constrained trajectory clustering as the problem of regrouping trajectories based on their movement pattern: the more two trajectories visit the same road segments, the more they are considered similar. The logic at work here is that detecting mobility profiles can help grasp a better knowledge of how the road network is put to use. Secondly, we tried to study the road segment clustering problem in which we are rather interested in discovering clusters of road segments that are often visited together. The motivation behind this is that doing so can help predict the propagation of traffic jams: if a congestion is detected for some members of a given segment cluster, then one can rationally presume that it may potentially spread to the other members of the same group. We approached both problems using the same logic. We used a graph representation to model the interactions between entities of interest (i.e. trajectories or road segments) and we used a modularity-based hierarchical community detection algorithm to retrieve hierarchies of nested clusters.

Our experiments with the trajectory clustering approach yielded promising results. We showed how trajectory clusters can be explored with various levels of detail, how cluster representatives can be potentially extracted if they are needed, etc. Road segment clustering, on the other hand, was more problematic. Applying the same techniques to solve this problem resulted in segment clusters that are hard to interpret due primarily to their lack of context. One way to overcome this is to analyze road segment clusters in the light of trajectory clusters. However, it is up to the user to manually do so by choosing the appropriate levels in the hierarchies of trajectory clusters and segment clusters, and interpreting the interactions between pairs of clusters accordingly. Moreover, there certainly is a duality between the two problems

we defined: clustering trajectories somewhat yields a segment partition and vice versa, so why not merge the two problems together and try to extract both partitions simultaneously?

To address these observations and shortcomings, we propose in this chapter an alternative approach to partitioning trajectory data which is based on the use of co-clustering. Co-clustering is a technique that aims to partition observations in a given dataset based on all of their descriptive variables simultaneously. For instance, if we consider that our observations are visits to road segments where each visit is described using two variables (the trajectory's id and the visited road segment's id), co-clustering will try to find groups of visits emanating from the same trajectories to the same road segments. One of the attractive features of co-clustering is the fact that it takes advantage of interactions and correlations between variables in order to retrieve more complex patterns and trends (in comparison with those discovered by simpler clustering techniques). The contribution we present here is a joint work with Romain Guigourès and Marc Boullé from Orange Labs with whom we collaborated in order to apply their algorithms to trajectory data we provided.

The chapter is structured as follows. Since the work presented here is based on co-clustering, we give a brief insight regarding this aspect in Section 4.1. Our data model and methodology are explained in Section 4.2. In Section 4.3, we present a case study where we apply the new approach to a dataset (the one we already used in Section 3.4.3.1) in order to showcase its use to characterize traffic in the road network and make the comparison with the techniques we presented in the previous chapter. Experimental results are presented in Section 4.4. Finally, concluding remarks are exposed in Section 4.5.

4.1 Related Work

Co-clustering (or biclustering) can be defined as the "simultaneous clustering of both row and column sets in a data matrix" [110]. Since this technique aims to extract features based on interrelations between rows and columns simultaneously, it can be used to reveal more complex patterns than those retrieved by more conventional partitioning techniques. Co-clustering (like normal clustering) works on conventional data that are expressed as a set of unlabeled observations described through a number of variables.

The first attempt at co-clustering is often accredited to Hartigan [111]. Although the term "biclustering" (introduced later on in [110]) was not used explicitly, Hartigan proposed to extract sub-matrices (dubbed clusters) from a given data matrix. The rows of this data matrix correspond to the observations (or cases), whereas the columns correspond to the variables. Each case has a response to each variable. In order to build the clusters, a suitable model needs to be defined. Hartigan explores the equal-response model where it is desirable to have clusters in which the cases had the same response w.r.t. the variables. Consequently, a matrix A^* is used to model the average interactions

between case clusters and variable clusters in the data matrix A. A^* is none other than the ideal data matrix closest to A under the equal-response model. The deviation of the original data matrix A w.r.t. the ideal case A^* is measured using the sum of squared deviations $SSQ = \sum_{i,j}(A_{ij} - A^*_{ij})^2$ and is used as a quality criterion to guide the clustering process which is conducted using a heuristic hierarchical divisive algorithm. Initially, all the cells of A^* are equal to the average of the values in A. At each step, the algorithm tries to retrieve the best split of the matrix into two sub-matrices (which is the partition maximizing the SSQ reduction). The process is reiterated[1] until the SSQ reduction is less than the SSQ reduction expected by chance (i.e. obtained by random splits). This results not only in a direct partition of cases and variables but also in two hierarchies of nested case clusters and nested variable clusters. A similar technique that uses cluster inertia as the quality criterion is presented in [112], whereas approaches to optimally partition a data matrix into sub-matrices are described in [113, 114].

Stochastic co-clustering techniques make the hypothesis that the analyzed data stem from a mixture of underlying distributions and well-defined statistical models. Approaches to estimate the parameters of these models include Expectation Maximization (EM) based techniques [115, 116], bayesian techniques such as the Latent Dirichlet Allocation generative model [117], etc. The latter was applied to clustering documents and words concurrently.

Dhillon proposed a spectral approach to co-clustering words and documents in [118]. In its philosophy, the approach is very close to what we are trying to achieve since a bipartite graph model is presented to render the interactions between words and documents. In [118], a data matrix A is used to count the occurrences of words (columns) in the documents (rows). A is first normalized into $A_n = D_1^{-\frac{1}{2}} A D_2^{-\frac{1}{2}}$ (D_1 and D_2 are diagonal matrices representing the frequencies of words and documents: $D_1(i,i) = \sum_j A_{ij}$ and $D_2(j,j) = \sum_i A_{ij}$). Then, instead of using the eigenvectors like in normal spectral clustering [100], the second singular vectors of A_n are calculated and used in combination with k-means to retrieve the co-clusters of words and documents. An extension of this approach for the co-clustering of gene expression data can be retrieved in [119].

Other approaches are inspired by information retrieval theory and use mutual information [120] as the quality criterion for clustering. For example, if we are interested in the two-dimension case where a dataset $\mathcal{D}$ is described through two variables X_1 and X_2, such approaches would search for the optimal biclustering model $\mathcal{M}$ (defined by the partitioning π of X_1 and X_2 into two partitioned variables X_1^π and X_2^π) that minimizes the loss of mutual information, which is expressed as follows (4.1):

$$\mathcal{M}^* = \underset{\mathcal{M}}{\operatorname{argmin}}(I(X_1, X_2) - I(X_1^\pi, X_2^\pi)) \ . \tag{4.1}$$

[1] Obviously, if the algorithm degenerates, it ends up in the situation where each cluster contains only one response (i.e. one case and one observation) in which case $A^* = A$.

Dhillon et al. [121] use this criterion in combination with a monotonically-decreasing algorithm in order to retrieve a locally-optimal biclustering of words and documents. A generalization of the mutual information biclustering model can be found in [122].

Blockmodeling approaches [123, 124] may also be considered as co-clustering techniques. These approaches operate by rearranging the rows and columns of the data matrix in order to extract uniform and similar blocks (by employing a χ^2 test, a density-based criterion, etc.). One blockmodeling technique of particular interest in the context of this work is the MODL [125] approach. MODL is based on Bayesian model selection and was originally proposed for supervised discretization and value grouping, then generalized later on to supervised multivariate classification and unsupervised co-clustering. Since what we are trying to achieve in this chapter is a clustering of categorical bivariate data, we briefly present MODL only under this specific perspective.

In order to describe the joint distribution of the data, MODL investigates a family of unsupervised bivariate value grouping models. A model $\mathcal{M}$ in this family can be described through the following parameters [125]:

- The number of groups J_1 and J_2 for each of the input variables Y_1 and Y_2.
- The repartition of the values, for each variable, into groups of values.
- The distribution of the N instances of the data sample $D = \{D_1, D_2, ..., D_N\}$ among the G cells of the resulting data grid ($G = J_1 J_2$).
- For each variable and each group, the distribution of the instances of the group on the values of the group.

These parameters are either (i) parameters describing the partition of values into groups of values (4.2), (ii) parameters of the multinomial distribution of the instances on the data grid cells (4.3), or (iii) parameters of the multinomial distribution of the instances of each group on the values of the group (4.4).

$$J_1, J_2, \{j_1(v_1)\}_{1 \leq v_1 \leq V_1}, \{j_2(v_2)\}_{1 \leq v_2 \leq V_2} \ . \tag{4.2}$$

$$\{N_{j_1 j_2}\}_{1 \leq j_1 \leq J_1, 1 \leq j_2 \leq J_2} \ . \tag{4.3}$$

$$\{n_{v_1}\}_{1 \leq v_1 \leq V_1}, \{n_{v_2}\}_{1 \leq v_2 \leq V_2} \ . \tag{4.4}$$

V_1 and V_2 are the number of values for each variable. $j_1(v_1)$ (resp. $j_2(v_2)$) stands for the index of the group containing value v_1 (resp. v_2), n_{v_1} (resp. n_{v_2}) is the number of instances for value v_1 (resp. v_2), and $N_{j_1 j_2}$ is the number of instances in the cell (j_1, j_2) of the data grid.

In order to evaluate each model, a Bayesian prior is defined. According to this prior, parameters are chosen hierarchically and uniformly at each level

with the hypothesis of totally-independent and equiprobable distributions (cf. Figure 4.1). Then, the Bayes optimal model w.r.t. the prior is selected. This model is the one minimizing the following criterion (4.5):

$$
\begin{aligned}
& \log V_1 + \log V_2 + \log B(V_1, J_1) + \log B(V_2, J_2) \\
+ \quad & \log \binom{N+G-1}{G-1} + \sum_{j_1=1}^{J_1} \log \binom{N_{j_1}+m_{j_1}-1}{m_{j_1}-1} + \sum_{j_2=1}^{J_2} \log \binom{N_{j_2}+m_{j_2}-1}{m_{j_2}-1} \\
+ \quad & \log N! - \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \log N_{j_1 j_2}! \\
+ \quad & \sum_{j_1=1}^{J_1} \log N_{j_1}! + \sum_{j_2=1}^{J_2} \log N_{j_2}! - \sum_{v_1=1}^{V_1} \log n_{v_1}! - \sum_{v_2=1}^{V_2} \log n_{v_2}! \ .
\end{aligned}
\tag{4.5}
$$

$B(V, J)$ is the number of divisions of V values into J groups, m_{j_1} (resp. m_{j_2}) is the number of values in group j_1 (resp. j_2), and N_{j_1} (resp. N_{j_2}) is the number of instances of group j_1 (resp. j_2) of variable Y_1 (resp. Y_2). The first line in the formula depends on the prior distribution of group numbers and the partitioning of values into these groups. The second line is composed of the specification of the parameters of the multinomial distribution of instances on the cells of the data grid and the specification of the multinomial distribution of instances of each group on the values of said group. The likelihood of the distribution of instances on the grid cells is expressed in the third line, whereas the likelihoods of local distributions for each variable are described in the fourth line.

In the particular case of co-clustering of bivariate categorical data, the MODL criterion converges asymptotically towards the mutual information criterion as defined in [121] (the demonstration can be found in [126]). A general overview of the algorithmic aspect of co-clustering using the MODL criterion will be exposed in Section 4.2.2.

Finally, to the best of our knowledge, there were little to no attempts to co-cluster moving object trajectory data (especially in the network-constrained case). Among the few attempts that we are aware of in this respect is the one presented in [127] where a dual hierarchical Dirichlet process (Dual-HDP) is used for trajectory analysis based on semantic region modeling in video surveillance settings.

For a more in-depth survey on co-clustering techniques, we refer the interested reader to the one presented in [128].

4.2 Data Model and Methodology

We now present our proposition for co-clustering network-constrained trajectory data. As far as the road network and trajectories are concerned, we use the

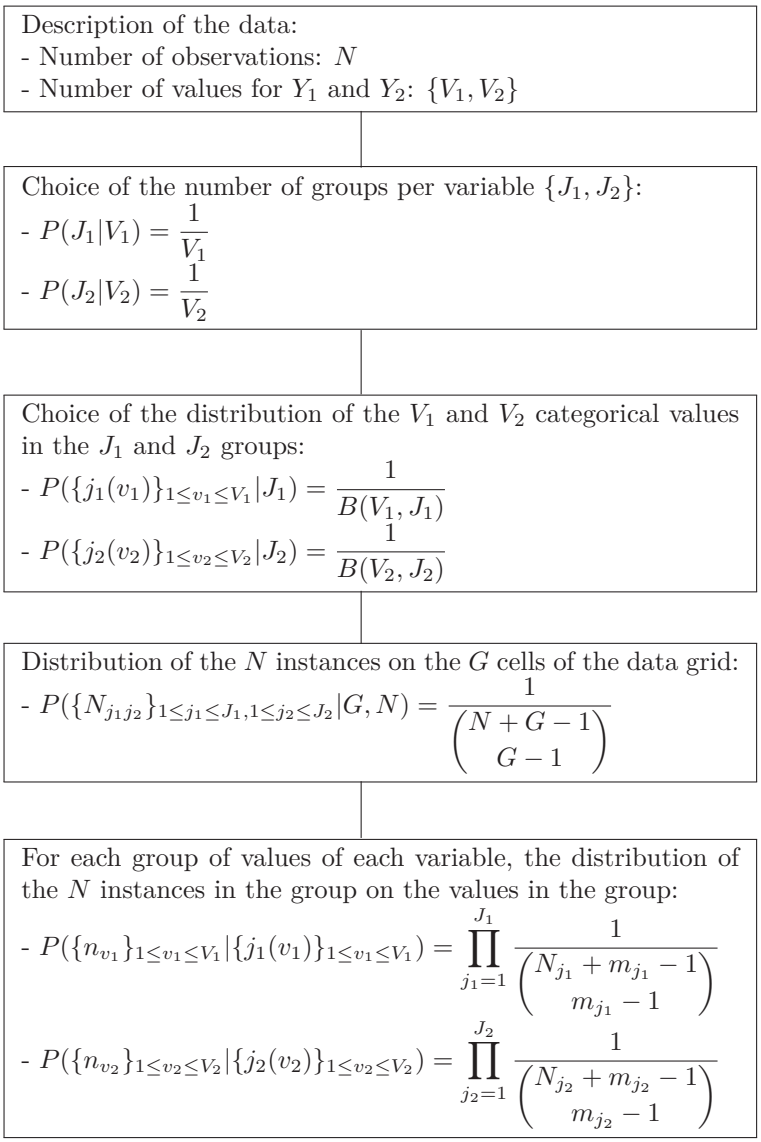


Figure 4.1 – Hierarchy of parameters established by the prior.

same symbolic data model that we used in Chapter 3, in which the road network is represented as an oriented graph depicting its intersections and road segments, and where trajectories are expressed as sequences of traveled road segments (cf. Section 1.1.2 for the full details of the symbolic model). The problem at hand can be defined as follows:

Definition 11 (Network-Constrained Trajectory Co-Clustering Problem)**.** Given a set of trajectories $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ and the set of all the road segments $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ they visited, we aim to simultaneously partition both the trajectory set $\mathcal{T}$ and the segment set $\mathcal{S}$ in order to discover meaningful non-overlapping co-cluster structures. Each co-cluster involves a trajectory cluster and a segment cluster that are regrouped based on the uniformity of

the visits the road segments receive from the trajectories. Desirably, in a given co-cluster, each road segment is visited by most of the trajectories in the same co-cluster and, vice versa, each trajectory visits most of the segments in the co-cluster.

We present our data model for trajectory/segment interactions in Section 4.2.1 then we explain our approach to partitioning the data in Section 4.2.2

4.2.1 Modeling Trajectory/Segment Interactions Using Bipartite Graphs

In the context of co-clustering network-constrained trajectory data, we are interested in modeling mutual interactions between trajectories and road segments (instead of modeling interactions solely between trajectories or between segments). We do so by adopting a bipartite graph structure. A bipartite graph is a graph whose vertices can be divided into two separate and independent sets. An edge can only link a vertex from the first set to a vertex from the second (i.e. no edges are tolerated between vertices of the same type).

We model interactions between trajectories and road segments as a bipartite graph $\mathcal{G} = (\mathcal{T}, \mathcal{S}, \mathcal{E})$. $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ is the set of trajectories, $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ is the set of all the segments composing the road network that registered at least one visit, and $\mathcal{E}$ is the set of edges modeling visits (i.e. interactions) from trajectories to road segments. An edge e exists between a trajectory T and a road segment s if and only if T visited s at least once. This representation is illustrated through the example in Figure 4.2.

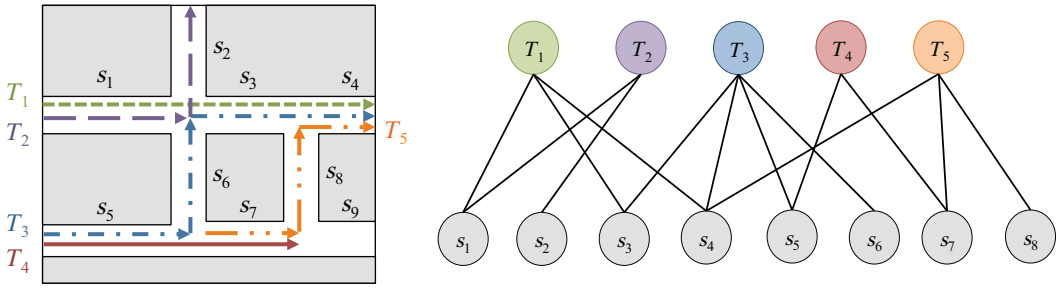


Figure 4.2 – Example of five trajectories visiting eight road segments in the network and the equivalent trajectory/segment interactions bipartite graph.

Notice that this representation can be considered as a natural extension to the graph models we presented in the previous chapter. In fact, projecting the bipartite graph on its set of trajectory vertices $\mathcal{T}$ will yield a simple graph that is very similar to the trajectory similarity graph described in Section 3.3.2 whereas projecting it on its segment vertices $\mathcal{S}$ will produce a graph similar to the segment similarity graph in Section 3.4.2 (the only difference is the weights assigned to the edges).

4.2.2 Co-Clustering the Bipartite Graph

In order to partition the bipartite graph $\mathcal{G}$, we apply a co-clustering approach to its adjacency matrix. In the adjacency matrix, trajectories are represented in the rows while road segments are represented on the columns. The intersection of row i with column j indicates the number of times trajectory T_i visited the road segment s_j (which is often equal to 1 in our context since each trajectory does not normally visit the same segment more than once). The co-clustering structure that we aim to achieve (referred to as $\mathcal{M}$ hereafter) is defined through the set of modeling parameters described in Table 4.1.

Table 4.1 – Notations.

Bipartite graph $\mathcal{G}$	**Co-clustering model $\mathcal{M}$**
$\mathcal{T}$: set of trajectories	$C_{\mathcal{T}}$: set of trajectory clusters
$\mathcal{S}$: set of road segments	$C_{\mathcal{S}}$: set of road segment clusters
$\mathcal{E}$: set of visits from trajectories in $\mathcal{T}$ to road segments in $\mathcal{S}$	$C_{\mathcal{E}}$: co-clusters of trajectory and road segments

The main advantage of co-clustering techniques is that they do not require preprocessing nor do they require the definition of an "artificial" similarity between trajectories or between segments. Nonetheless, they do present the drawback of being computationally expensive since interesting formulations of the problem are known to be NP-complete [128].

As we hinted before, we opt for the MODL [125] approach to conduct the co-clustering of $\mathcal{G}$. We made this choice because this approach (i) is non-parametric and does not require user intervention or fine-tuning, (ii) is easily scalable and can consequently be used to analyze large datasets, and (iii) was already and successfully applied to geo-tagged data [129]. MODL co-clustering is conducted using an agglomerative greedy heuristic. Initially, the trivial, most refined model is considered. This model contains only one trajectory and one road segment per cluster. Then, all cluster merging operations are evaluated and the best merge (w.r.t. the quality criterion) is applied. Once no more merging operations are possible, the result of the heuristic is refined using a post-optimization step in which some elements swap their cluster memberships. The whole process is encapsulated within a Variable Neighborhood Search (VNS) [130] metaheuristic that restarts the algorithm several times with different random cluster initializations. The algorithm has a complexity of $O(|\mathcal{E}|\sqrt{|\mathcal{E}|}\log(|\mathcal{E}|))$ [125] where $|\mathcal{E}|$ indicates the total number of edges in the bipartite graph $\mathcal{G}$ (which, in our case, translates to the overall number of road segment traversals).

In practice, we use the Khiops [131] tool developed by Orange Labs to conduct the MODL co-clustering. Khiops provides the best co-clustering model (w.r.t. the MODL criterion) as well as two separate hierarchies of nested clusters, one for the trajectories and one for the road segments. These hierarchies are the result of a post-processing step in which an agglomerative hierarchical

clustering is conducted using the degradation of the MODL criterion as a dissimilarity measure. Starting from the optimal co-clustering, at each step a decision is made about whether to merge two trajectory clusters or two segment clusters based on which one produces the smallest degradation. The process stops when all the data is merged within a single co-cluster containing one trajectory cluster and one segment cluster. The levels in the hierarchy of trajectory (resp. segment) clusters trace all the merging operations going from the trajectory (resp. segment) clusters in the optimal level up to one single cluster containing all the trajectories (resp. road segments). Consequently, the same ability of multi-level exploration in the case of the approaches in Chapter 3 is also possible here: if the optimal co-clustering contains an overwhelming number of clusters, the user can decide to trade-off some quality and analyze fewer clusters of lower resolution. Khiops also offers the possibility to visualize useful information (such as mutual information and frequencies in the retrieved co-clusters) that help guide the analysis and interpretation of the results. These aspects will be showcased throughout Section 4.3 and Section 4.4.2.

4.3 Case Study

In this section, we present a case study through which we showcase our proposition. We also point out some key differences between the present approach and the ones we already proposed in Chapter 3.

4.3.1 Test Case Data

For our case study, we will reuse the dataset that we already introduced in Section 3.4.3.1. We remind that the dataset is synthetic and is composed of 85 trajectories divided into five ground-truth clusters. The trajectories visited a total of 485 road segments in the road network of Oldenburg. In order to distinguish between the original clusters and those discovered by means of clustering, the former will be called "classes" from now on. As explained in Section 3.4.3.1, these classes interact among themselves and are consequently not easy to separate. For the convenience of the reader, the illustration of the five trajectory classes (already presented in Figure 3.17) is replicated here in Figure 4.3.

4.3.2 Analysis of the Trajectory Clusters

Recall that when we applied our modularity-based trajectory clustering to the dataset, the approach started by discovering three trajectory clusters only. The reason behind this is that the approach picked up on the interactions between trajectory classes and partitioned the data accordingly. Further exploration of the three discovered clusters (using the second level of the produced cluster hierarchy) yielded eight more refined clusters, three among

(a) Class 1 (14 trajectories)

(b) Class 2 (19 trajectories)

(c) Class 3 (20 trajectories)

(d) Class 4 (20 trajectories)

(e) Class 5 (12 trajectories)

Figure 4.3 – Classes (ground-truth trajectory clusters) in the dataset.

which are perfect matches of ground-truth clusters. The remaining five clusters were pure clusters (as they contained only trajectories from one class) resulting from over-partitioning some classes where the algorithm detected a considerable variability. The confusion matrix between original classes and clusters of the second level of hierarchy is shown in Table 4.2.

Table 4.2 – Confusion matrix of the original classes present in the data (rows) and the clusters discovered by applying modularity-based community detection (columns).

	1	2	3	4	5	6	7	8
1	0	0	0	7	3	4	0	0
2	0	19	0	0	0	0	0	0
3	12	0	8	0	0	0	0	0
4	0	0	0	0	0	0	0	20
5	0	0	0	0	0	0	12	0

MODL co-clustering, on the other hand, was not "fooled" by the interactions in-between classes and directly retrieved a partition that is more faithful to the original data (cf. Table 4.3). Here again, two original classes were over-partitioned and three classes were retrieved correctly.

Table 4.3 – Confusion matrix of the original classes present in the data (rows) and the clusters discovered by applying MODL co-clustering to the bipartite graph's adjacency matrix (columns).

	1	2	3	4	5	6	7
1	0	0	7	7	0	0	0
2	0	0	0	0	19	0	0
3	0	0	0	0	0	12	8
4	20	0	0	0	0	0	0
5	0	12	0	0	0	0	0

The confusion matrix between trajectory clusters discovered by both approaches is depicted in Table 4.4 and shows that the correctly-discovered classes are the same in both cases and indicates that the two approaches only disagreed on the clustering of the remaining two "ambiguous" classes.

Unlike our modularity-based techniques, MODL does not require a preprocessing step and the concoction of a measure of similarity between observations. Instead, MODL is applied directly to the raw data. Moreover, MODL works on both variables describing the data (trajectories and road segments) simultaneously, whereas in modularity-based clustering only one variable is considered at a time. Therefore, and in view of the similar results achieved by both

Table 4.4 – Confusion matrix between trajectory clusters discovered by MODL (rows) and trajectory clusters discovered by modularity optimization (columns).

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	20
2	0	0	0	0	0	0	12	0
3	0	0	0	7	0	0	0	0
4	0	0	0	0	3	4	0	0
5	0	19	0	0	0	0	0	0
6	12	0	0	0	0	0	0	0
7	0	0	8	0	0	0	0	0

approaches w.r.t. the produced trajectory clusters, it is more tempting to apply MODL even if the user is only interested in clustering trajectories.

4.3.3 Mutual Analysis of Trajectory and Road Segment Clusters

We now include the road segment clusters in our case study. The co-clusters discovered using MODL involve twelve segment clusters. On the other hand, modularity-based clustering results in a hierarchy of six levels (with four segment clusters in the most coarse level and 41 clusters in the most detailed one as we mentioned before in Section 3.4.3.1). The second level of this hierarchy contains 13 segment clusters and is the one that resembles the most the clusters discovered using MODL. Therefore, we will mainly consider this level for our study.

4.3.3.1 Differences Between Co-Clustering and Separate Trajectory and Segment Clustering

Table 4.5 depicts the confusion matrix between road segment clusters discovered by both approaches and indicates that they handle segment clustering in drastically different fashions. In order to understand the reasons underlying the differences between the two approaches, we propose studying the case of the 11^{th} column of Table 4.5 which corresponds to a segment cluster (Figure 4.4) discovered by the modularity-based approach that the co-clustering approach decided to split into three different clusters (Figure 4.5).

MODL co-clustering decided that the segments belong to three different clusters for the following reasons:

- Traffic on the segment cluster depicted in Figure 4.5(c) comes uniquely from one trajectory cluster (corresponding to the ground-truth class depicted earlier in Figure 4.3(b)).

Table 4.5 – Confusion matrix between road segment clusters discovered using MODL co-clustering (rows) and those discovered using modularity-based clustering (columns).

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	0	0	0	0	0	0	0	0	0	0	34	0
2	0	0	0	0	0	0	0	0	0	0	0	0	37
3	0	0	0	11	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	11	0	0
5	0	0	0	0	0	40	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	35	0	2
7	0	0	0	0	0	0	0	16	49	12	0	0	0
8	0	0	0	0	15	0	6	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	14
10	0	0	0	0	0	0	0	0	0	0	60	0	0
11	43	27	15	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	6	0	25	0	0	0	0	0	0

- Traffic on the segment cluster in Figure 4.5(b) comes not from one but two groups of trajectories (the ones in Figure 4.3(b) and Figure 4.3(c)).

- Traffic on the cluster in Figure 4.5(a) emanates from three trajectory classes (Figures 4.3(a), 4.3(b) and 4.3(c)).

This last cluster is very important as it reveals the presence of an interesting hub structure (as discussed later in Section 4.3.3.2) that multiple groups of moving objects coming from different areas borrow to go to different destinations. This valuable information was detected and revealed immediately by the co-clustering approach since it took into account the provenance (trajectories) of the visits on the road segments.

In order to analyze the situation from the modularity-based approach's perspective, we plot the adjacency matrix of the segment similarity sub-graph (illustrated in Figure 4.6) corresponding to the cluster in Figure 4.4. We re-ordered the rows and columns of the matrix in order to group together the segments based on their membership to the three clusters discovered by co-clustering (Figure 4.5). The segments in the first two tiers of the matrix (which correspond to the clusters in Figure 4.5(a) and Figure 4.5(b)) are considered very similar w.r.t. the similarity measure we defined in Section 3.4.1. Consequently, the fact that modularity-based clustering regrouped them in the same cluster is very logical. Segments in the remaining tier (which obviously are the ones in Figure 4.5(c)) are less similar to the two others and should have been isolated. However, the approach decided to keep it in the same cluster at this level of hierarchy. Thereupon, it is up to the user to "guess" that

Figure 4.4 – A road segment cluster (regrouping 106 segments) discovered by modularity-based clustering.

(a) Cluster 4 (11 segments) (b) Cluster 6 (37 segments) (c) Cluster 10 (60 segments)

Figure 4.5 – The three road segment clusters discovered by co-clustering instead of the unique cluster depicted in Figure 4.4.

the concerned segment cluster needs further refinement in order to potentially retrieve that same information that co-clustering retrieved.

For the segment cluster at hand, our modularity-based approach failed to detect the hub structure and overlooked it completely even at the finest level of detail (i.e. the bottommost level of hierarchy). In fact, analyzing this level reveals the tendency of the modularity-based approach to over-cluster the data as can be witnessed in Figure 4.7. This behavior can handicap the analysis of large datasets as it can result in an overwhelming number of segment clusters (cf. Section 4.4.3.1).

Curiously, the 11^{th} row of Table 4.5 corresponds to the inverse situation of the one we just studied: a single segment cluster detected by co-clustering was split into three clusters by modularity-based clustering (these clusters are not illustrated here for conciseness's sake). The traffic on the concerned road segments originates uniquely from a single trajectory class (Figure 4.3(e)) that

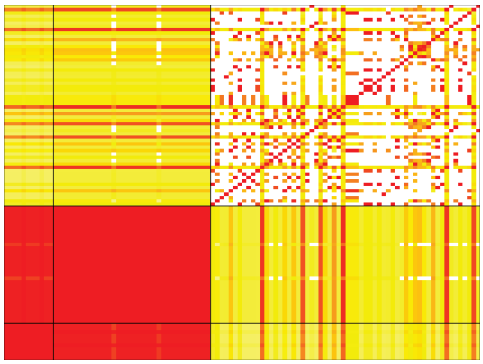


Figure 4.6 – Adjacency matrix of the segment similarity sub-graph of the cluster depicted in Figure 4.4. Color coding indicates the intensity of the similarity between the segments and varies from white (no similarity) to red (as the similarity approaches to 1).

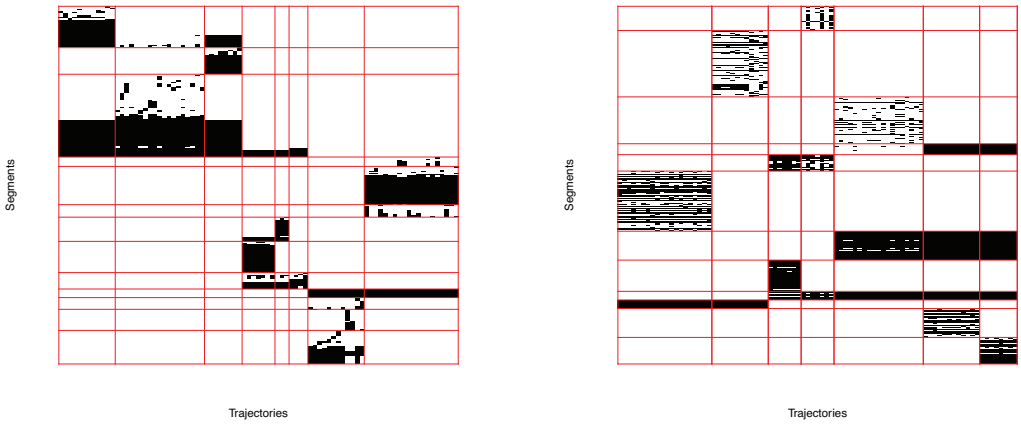
(a) 51 segments (b) 9 segments (c) 6 segments

(d) 5 segments (e) 9 segments (f) 3 segments

Figure 4.7 – Some of the most refined child clusters of the segment cluster depicted in Figure 4.4. Notice how even at this most detailed level, the hub structure in Figure 4.5(a) is not detected. Instead, over-fitting is observed since the core of the cluster in Figure 4.4 remains intact (a) but the approach starts separating peripheral segments into small clusters of no particular relevance ((b) through (f)).

both approaches retrieved correctly. While co-clustering successfully regrouped all these segments together, modularity-based clustering decided to spread them on multiple clusters. This is due to the same over-clustering problem we just mentioned: since the cluster was obvious to detect, the modularity-based approach started over-clustering it immediately and isolated the first segments and the last segments of the involved trajectories from the main segment cluster and into two separate clusters.

The difference in handling road segment clustering is also visible when studying the adjacency matrix of the original bipartite graph $\mathcal{G}$. In Figure 4.8, we re-ordered the rows and columns of the matrix in order to bring together trajectories and segments belonging to the same clusters. We observe in the case of the modularity-based approaches (Figure 4.8(a)) that road segments are regrouped together based on common trajectories without accounting for the traffic's volume. Therefore, road segments that are rarely visited can be attached to segments that are visited frequently. This translates, when looking at the adjacency matrix, into the presence of blocks with heterogeneous distributions in which some segments are traveled by all the trajectories in the cluster, whereas others are only visited by a limited subset of trajectories. In co-clustering, on the other hand, segments are correlated based on usage, which results in blocks of homogeneous densities (Figure 4.8(b)).



(a) Modularity (b) Co-clustering

Figure 4.8 – Crossed matrices of trajectory clusters (columns) and road segment clusters (rows) retrieved through (a) modularity-based clustering and (b) co-clustering.

4.3.3.2 Using Trajectory/Segment Co-Clusters for Traffic Characterization

By inspecting trajectory clusters and road segment clusters simultaneously, it is possible to characterize road segments based on the roles they play in traffic. This makes it possible to identify hubs that are frequently traveled by multiple groups of vehicles driving to different regions (Figure 4.9 and Figure 4.10), secondary roads (Figure 4.11), and even rarely frequented alleys. Therefore, our methodology makes it possible to characterize the topological structure of the underlying road network based on trajectories contributing their usage information.

(a) Segment cluster 3 (11 segments)

(b) Trajectory cluster 1 (20 trajectories)

(c) Trajectory cluster 2 (12 trajectories)

Figure 4.9 – A hub road segment traveled by two different trajectory clusters with different departures and destinations.

Mutual information is frequently used in co-clustering to quantify the correlations between partitions of the studied variables. These are, in our case, trajectories and road segments. Mutual information is always positive. High values of this index usually indicate that trajectory clusters visit rather exclusive and unique segment clusters. We use mutual information in our study to quantify the relationship between pairs of trajectory and segment clusters as well as their contribution to the model's mutual information. Given a cluster of trajectories $c_T \in C_\mathcal{T}$ and a cluster of road segments $c_S \in C_\mathcal{S}$, the contribution of the pair to mutual information, denoted $mi(c_S, c_T)$, is calculated as follows (4.6):

$$mi(c_S, c_T) = P(c_S, c_T) \log \frac{P(c_S, c_T)}{P(c_S)P(c_T)} \ . \tag{4.6}$$

Where $P(c_S, c_T)$ is the probability of a segment traversal to belong to a trajectory in c_T and concern a road segment that belongs to c_S, $P(c_S)$ is the probability of visiting a segment belonging to c_S, and $P(c_T)$ is the probability of having a trajectory belonging to c_T.

(a) Segment cluster 4 (11 segments)
(b) Trajectory cluster 3 (7 trajectories)
(c) Trajectory cluster 4 (7 trajectories)

(d) Trajectory cluster 5 (19 trajectories)
(e) Trajectory cluster 6 (12 trajectories)
(f) Trajectory cluster 7 (8 trajectories)

Figure 4.10 – A second hub road segment traveled by five different trajectory clusters with different departures and destinations.

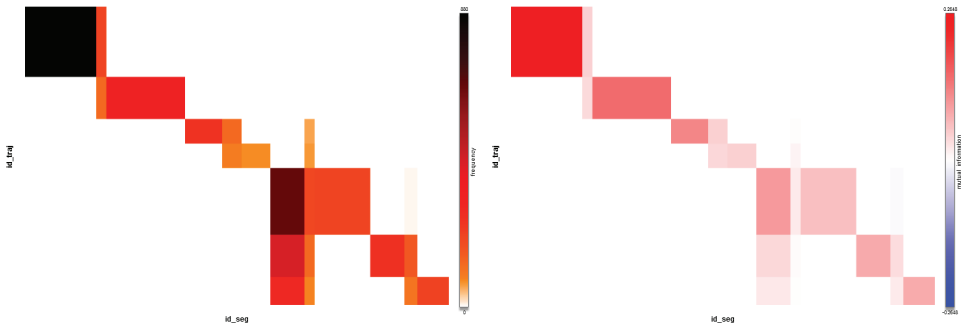
(a) Cluster 1 (34 segments)
(b) Cluster 5 (40 segments)
(c) Cluster 7 (77 segments)

Figure 4.11 – Examples of "secondary" road segment clusters leading to peripheral areas of the road network and visited exclusively by single groups of trajectories.

A positive contribution to mutual information indicates that the number of visits of trajectories in c_T to road segments in c_S is higher than what is expected

in case the two clusters were completely independent from one another. Vice versa, a negative contribution is an indicator that quantity of traffic is inferior to normal. Finally, a null contribution to mutual information indicates that traffic either conforms to what is expected or is very low.

Figure 4.12(b) presents the contribution to mutual information for each pair of co-clusters discovered in the dataset at hand. For instance, if we take the left, topmost co-cluster, we can notice that the segments (already depicted in Figure 4.11(c)) are exclusively traveled by members of a single trajectory cluster (already depicted in Figure 4.9(b)). In this case, the trajectory cluster comprises 21.6% of the studied trajectories and the road segments cluster 17.3% of segments in the dataset. If we suppose that both clusters are independent, then we can expect no more than observing $21,6\% \times 17.3\% = 3.7\%$ of the total road segment traversals to be originating from both clusters. Here, however, we observe that no less than 17.3% from the total traversals belong to this co-cluster, which largely exceeds the expected traffic in case of unrelated and independent clusters.



(a) Frequency (b) Mutual information

Figure 4.12 – Frequency and mutual information of the retrieved co-clusters.

Notice that the mutual information (Figure 4.12(b)) contributes with an information that is different from the frequency matrix (Figure 4.12(a)). We can observe that some road segment clusters are significantly traversed by members belonging to multiple trajectory clusters. This behavior is quite characteristic of hubs that vehicles coming from different regions cross in order to attend different destinations. Some of these clusters have very small contrast w.r.t. mutual information, which indicates that traffic on the hub is rather balanced.

4.4 Experimental Results

We now proceed to the evaluation of our co-clustering approach on a larger scale. To do so, we compare it to the modularity-based trajectory clustering and segment clustering approaches we presented in the previous chapter. We

first describe our experimental setting in Section 4.4.1. We present some of the results co-clustering achieved in Section 4.4.2. The comparison with our previous techniques is reported in Section 4.4.3.

4.4.1 Experimental Setting

For this experimental study, we use five synthetic datasets generated with the Brinkhoff generator [6] using the Oldenburg road network. Each dataset is composed of 1000 moving object trajectories that visited over 8000 distinct road segments. The characteristics of each dataset are detailed in Table 4.6 along with the characteristics of the different graphs induced by our approaches.

Table 4.6 – Characteristics of the used datasets and the different similarity graphs they induce.

Dataset	Number of trajectories	Number of visited segments	Edges in the trajectory similarity graph	Edges in the segment similarity graph	Edges in the bipartite graph
1	1000	8336	30447	667556	41849
2	1000	8441	32123	662191	43182
3	1000	8477	30265	682737	42241
4	1000	8345	30961	652122	41771
5	1000	8269	29342	639676	41006

One straightforward observation that we can already make here concerns the size of the graphs produced by the different approaches. As it is made clear by Table 4.6, trajectory clustering produces similarity graphs that contain around 30000 edges. Co-clustering stays reasonably close to this with bipartite graphs that contain a little more than 40000 edges. Road segment clustering, on the other hand, produces far bigger graphs with no less than 600000 edges. This can be quite problematic when handling larger datasets that potentially involve larger road networks, especially considering the elevated time complexity of the approach (cf. Section 3.4.3).

We are mainly interested in studying the behavioral differences between clustering trajectories and road segments separately, and co-clustering both variables simultaneously. Therefore, we will compare both approaches under this angle. To do so, we use the following indices:

- The sum of average trajectory intra-cluster overlaps (cf. Section 3.3.5) achieved by both simple trajectory clustering and co-clustering.
- The sum of average road segment intra-cluster overlaps (cf. Section 3.4.4) achieved by both simple road segment clustering and co-clustering.
- The Adjusted Rand Index (ARI) between trajectory partitions and between road segment partitions discovered by both approaches (cf. Section 3.3.5).

- The Normalized Mutual Information (NMI) between trajectory partitions and between road segment partitions retrieved by both approaches.

Given two partitions $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ and $\mathcal{C}' = \{C'_1, C'_2, ..., C'_{k'}\}$ of a dataset containing N observations, the normalized mutual information between these partitions is calculated using Formula (4.7).

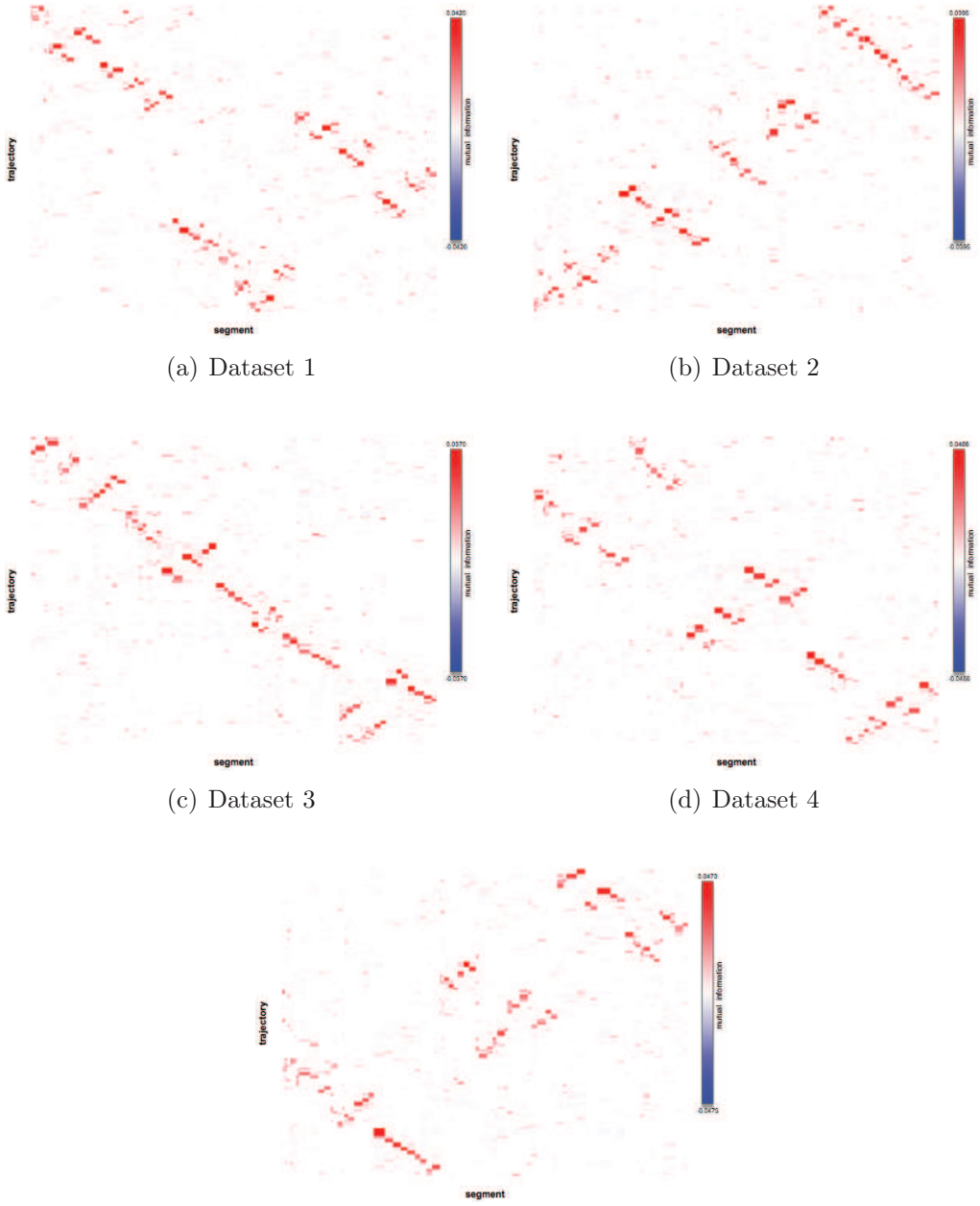
$$\mathrm{NMI}(\mathcal{C}, \mathcal{C}') = \frac{-2 \sum_{i=1}^{k} \sum_{j=1}^{k'} |C_i \cap C'_j| \log \frac{|C_i \cap C'_j| N}{|C_i||C'_j|}}{\sum_{i=1}^{k} |C_i| \log \frac{|C_i|}{N} + \sum_{j=1}^{k'} |C'_j| \log \frac{|C'_j|}{N}} \ . \tag{4.7}$$

Where $|C_i \cap C'_j|$ stands for the number of common observations between C_i and C'_j, and $|C_i|$ and $|C'_j|$ the number of observations in C_i and C'_j respectively.

4.4.2 Examples of Results Produced by Co-Clustering

Figure 4.13 depicts the matrices of mutual information between trajectory and segment clusters obtained by application of MODL co-clustering on the five datasets (each block in the matrix corresponds to a co-cluster composed of a trajectory cluster and a road segment cluster). In all cases, the resulting matrices are very sparse as, expectedly, each trajectory cluster interacts with only a limited number of segment clusters (and vice versa). This kind of visual aid can be used to quickly spot meaningful co-clusters (and simple clusters) and prune those that are not very relevant. For example, a trajectory cluster that does not have a strong interaction with any of the produced road segment clusters can either be a trajectory cluster that travels exclusively on hub road segments or an irrelevant, malformed cluster; a considerable interaction between a road segment cluster and multiple trajectory clusters is generally an indicator of a hub that is used by multiple groups of moving objects traveling in-between different areas (cf. Figure 4.14); etc.

Like we mentioned in Section 4.2.2, it is possible to study the clustering results at lower resolutions with the help of the hierarchies of trajectory clusters and road segment clusters generated by Khiops. The tool also offers the possibility of visualizing the various indicators, such as mutual information, for these resolutions as depicted in Figure 4.15. As is the case in our modularity-based approaches, this can be very convenient when analyzing large datasets as the user can choose to trade-off some quality to reduce the number of co-clusters to analyze to a reasonable level. The user can then expand, refine, and explore co-clusters of interest. This can also be used to explore just the hierarchy of trajectory clusters or the hierarchy of segment clusters. An example of cluster refinement, in the case of trajectory clusters, is illustrated in Figure 4.16: we started with the cluster of 66 trajectories shown in Figure 4.16(a). By expanding this cluster using the hierarchy provided by Khiops, we obtained two more specialized clusters with more apparent trends as the first (Figure 4.16(b)) moves in the central part of the network, whereas the second (Figure 4.16(c)) moves more on the east peripheral side. Further refinement helps make the



(a) Dataset 1

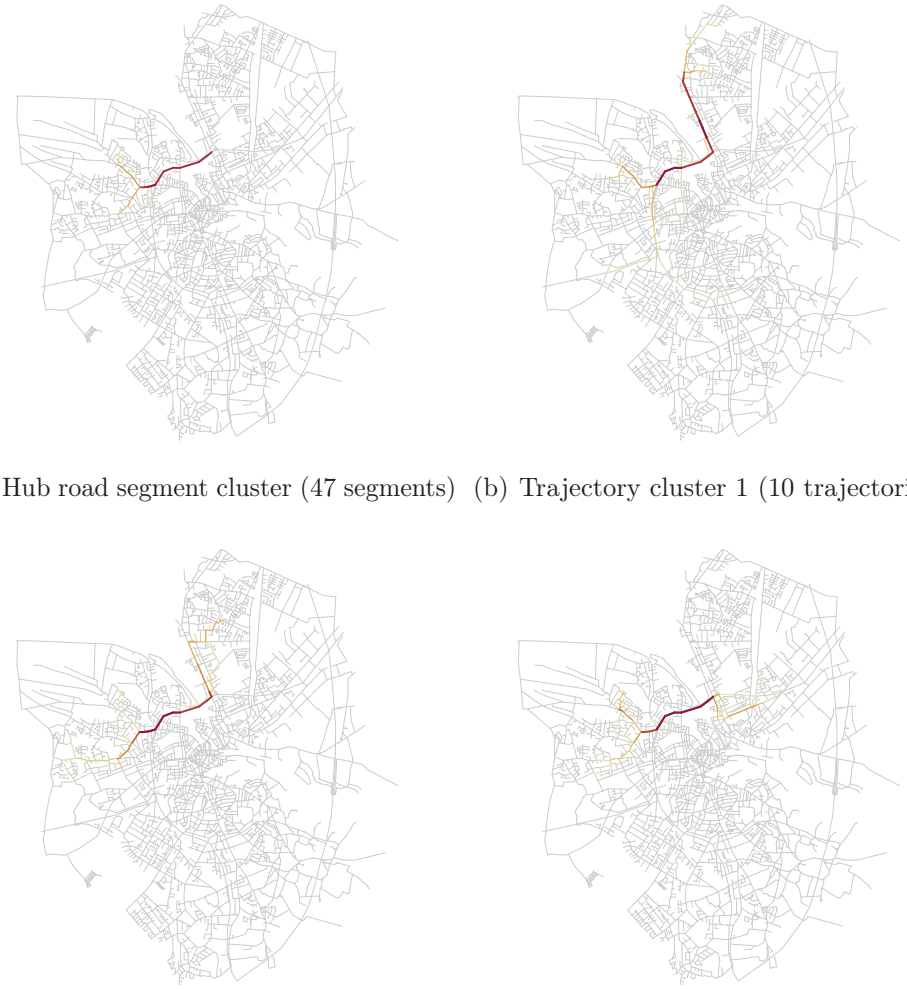
(b) Dataset 2

(c) Dataset 3

(d) Dataset 4

(e) Dataset 5

Figure 4.13 – Mutual information matrices of the co-clusters discovered in the datasets. Trajectory clusters are depicted in the rows and road segment clusters on the columns.



(a) Hub road segment cluster (47 segments) (b) Trajectory cluster 1 (10 trajectories)

(c) Trajectory cluster 2 (7 trajectories) (d) Trajectory cluster 3 (8 trajectories)

Figure 4.14 – Example of a hub road segment cluster identified in dataset 1 and the three trajectory clusters it interacts with.

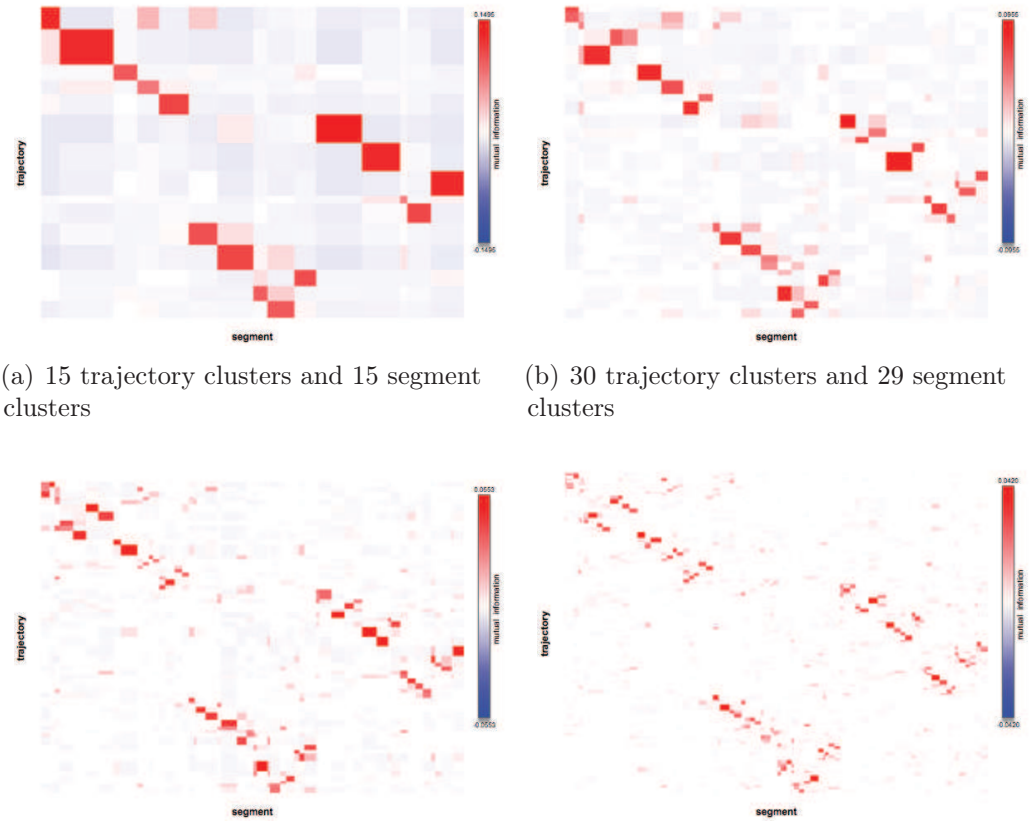
trend apparent in Figure 4.16(c) even more apparent (cf. Figure 4.16(d)) and isolate yet another trend in a separate cluster (Figure 4.16(e)). The hierarchy of the aforementioned clusters is depicted in Figure 4.17.

4.4.3 Comparison with Modularity-Based Clustering

We now compare the characteristics and quality of the clusters produced by our co-clustering approach and the approaches presented in Chapter 3.

4.4.3.1 Characteristics of the Discovered Clusters

Table 4.7 depicts the number of clusters obtained with co-clustering and those obtained with modularity-based clustering on both ends of the hierarchy it



(a) 15 trajectory clusters and 15 segment clusters

(b) 30 trajectory clusters and 29 segment clusters

(c) 60 trajectory clusters and 52 segment clusters

(d) Optimal level (113 trajectory clusters and 80 segment clusters)

Figure 4.15 – Mutual information matrices of the co-clusters discovered in dataset 1 at various resolutions.

produces. The evolution of the number of discovered clusters w.r.t. the level in the hierarchy in the case of modularity-based clustering is further detailed in Figure 4.18 (for the trajectory case) and Figure 4.19 (for the road segment case). The behavior of both approaches w.r.t. trajectory clustering is coherent as far as the number of discovered clusters is concerned. The two approaches behave differently when it comes to segment clusters. MODL co-clustering generates a low number of segment clusters. Modularity-based clustering, however, quickly degenerates and produces a much higher number of clusters as can be witnessed from Figure 4.19. This is due to the behavior we described in Section 4.3.3.1: once significant segment clusters are identified, the approach starts refining them by ejecting their peripheral segments and isolating them into very small clusters.

(a) Trajectory cluster 1
(66 trajectories)

(b) Trajectory cluster 2
(42 trajectories)

(c) Trajectory cluster 3
(24 trajectories)

(d) Trajectory cluster 4
(11 trajectories)

(e) Trajectory cluster 5
(13 trajectories)

Figure 4.16 – Example of trajectory cluster refinement by successive zooms and expansions of the cluster hierarchy.

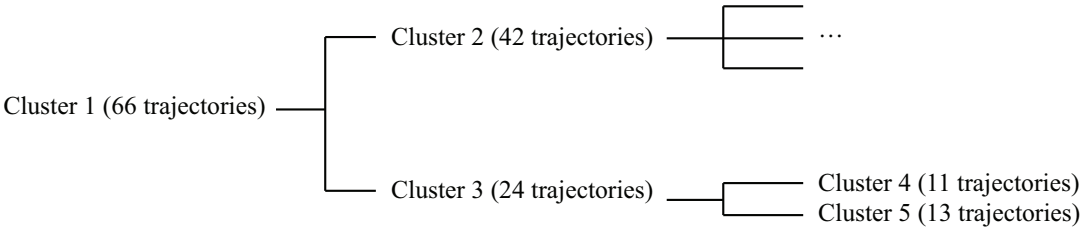


Figure 4.17 – Hierarchy of the trajectory clusters depicted in Figure 4.16.

Table 4.7 – Number of clusters retrieved by the different approaches on the trajectory datasets.

			Modularity-based clustering			
	Co-Clustering		Traj. clusters		Seg. clusters	
Dataset	Traj. clusters	Seg. clusters	High.level	Low. level	High. level	Low. level
1	113	80	10	105	15	1142
2	122	79	10	106	18	1196
3	128	76	11	129	15	1154
4	119	74	9	99	14	1154
5	122	72	10	98	19	1153

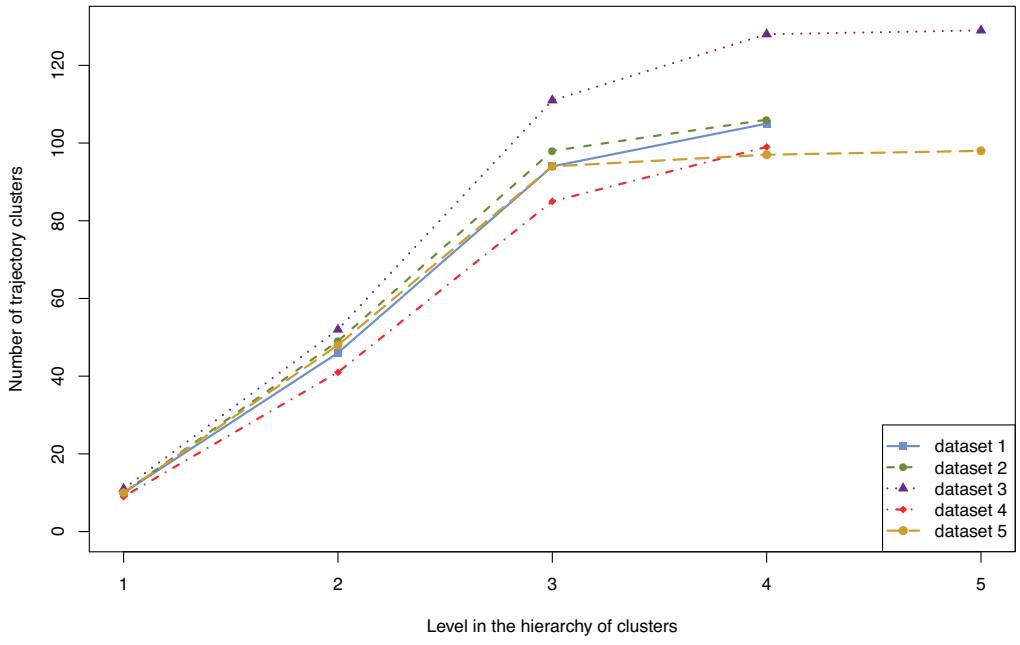


Figure 4.18 – Number of trajectory clusters vs. the level in the hierarchy discovered by modularity-based clustering.

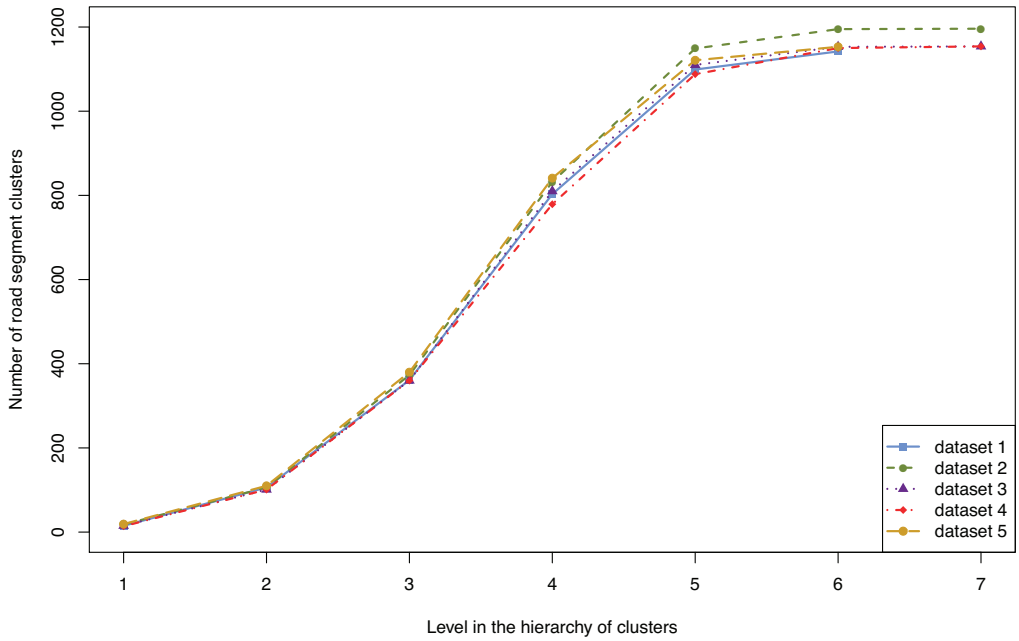


Figure 4.19 – Number of road segment clusters vs. the level in the hierarchy discovered by modularity-based clustering.

4.4.3.2 Comparison of the Clustering Quality

Since the number of trajectory clusters in the most refined level of the hierarchy of clusters produced by the modularity-based approach is close to the one produced by co-clustering, we choose to compare both approaches w.r.t. this level. Table 4.8 charts the values of the sum of average intra-cluster overlaps, Normalized Mutual Information (NMI), and Adjusted Rand Index (ARI) achieved by both approaches on each of the datasets.

Modularity-based trajectory clustering outperformed MODL co-clustering for the sum of average intra-cluster overlaps which indicates that, even while producing a slightly lower number of trajectory clusters, the latter are more compact and present more overlapping trajectories within the same clusters. We believe that this is mainly caused by the weighting technique we adopted while conducting simple trajectory clustering: in the modularity-based approach, the segments were weighted depending on their contribution to each trajectory and their global relevance w.r.t. the entirety of the dataset, which helped guide the clustering process. In MODL co-clustering, all road segments are treated equally without taking these factors in consideration.

For all the datasets, cross-comparing trajectory partitions produced by both approaches resulted in high values for the normalized mutual information (with an average of 0.74). This indicates that both partitions are highly correlated and

overlap considerably. The Adjusted Rand Index is a more precise indicator since it penalizes the differences in cluster memberships, cluster sizes, and number of clusters and since it's adjusted-for-chance. The results show low Adjusted Rand Index values (averaging only 0.27), which highlights a strong disagreement in cluster composition on both sides and indicates that the clustering is conducted very differently from one approach to the other.

Table 4.8 – Trajectory clusters' quality for modularity-based clustering and co-clustering with MODL. The number of trajectory clusters considered for each approach is specified in parentheses under their respective intra-cluster overlaps columns.

	Intra-cluster overlaps			
Dataset	Modularity	Co-clustering	NMI	ARI
1	371.75 (105)	352.35 (113)	0.74	0.29
2	368.74 (106)	359.99 (122)	0.73	0.25
3	412.41 (129)	368.13 (128)	0.76	0.27
4	361.38 (99)	354.65 (119)	0.73	0.25
5	349.31 (98)	354.24 (122)	0.74	0.29

We do the same comparison for the road segment partitions. However, since the number of clusters discovered in the case of modularity-based clustering increases considerably after its second level of hierarchy we select this level as the basis of comparison. This decision is further backed up by the fact that the number of clusters in this level is close to the number of segment clusters produced by the co-clustering technique.

The results (average sum of intra-cluster overlaps, NMI, and ARI) are shown in Table 4.9. The same observations that we made in the case of trajectory clusters still hold true here. Modularity-based clustering of road segments yields clusters that are more compact than their co-clustering counterparts. The high values of NMI and low values of ARI across all datasets allege that while the partitions are correlated, they show a considerable difference in cluster memberships.

The results we presented are somewhat expected. When conducting trajectory clustering separately, the modularity-based approach (Section 3.3) focuses only on the similarities between trajectories. The approach is further guided by the weighting technique (Section 3.3.1) that indicates irrelevant segments to neglect while assessing trajectory similarities. Analogically, our modularity-based segment clustering (Section 3.4) concentrates solely on the similarity relationships between road segments. In other words, these approaches try, by design, to maximize the overlaps within the produced clusters. In contrast, the co-clustering approach tries to characterize the interactions between trajectories and road segments. Consequently, it tries to optimize a totally different criterion that does not necessarily lead to compact clusters on each side.

Table 4.9 – Road segment clusters' quality for modularity-based clustering and co-clustering with MODL.

	Intra-cluster overlaps			
Dataset	Modularity	Co-clustering	NMI	ARI
1	646.23 (106)	474.75 (80)	0.71	0.36
2	645.83 (106)	460.93 (79)	0.71	0.35
3	621.18 (101)	452.59 (76)	0.70	0.35
4	605.23 (101)	424.59 (74)	0.69	0.33
5	662.82 (110)	424.44 (72)	0.71	0.37

The work we described here is a work in progress and more experiments (probably involving varying the co-clustering algorithm) are needed to draw better-grounded conclusions.

4.5 Conclusions

In this chapter, we presented our last contribution which consisted in sketching an approach to co-clustering network-constrained trajectory data. By extending our graph modeling paradigm, we represented interactions between trajectories and the road segments they visited as a bipartite graph. We then used a blockmodeling technique to co-cluster the adjacency matrix of this bipartite graph and simultaneously extract clusters of trajectories and clusters of segments that highly interact. Through a case study, we illustrated how this approach can be used to characterize traffic in road networks and through our experimental study, we compared results with the approaches we already defined in Chapter 3.

We emphasize that the work we presented here is still in progress. In future work, we intend to conduct more extensive experimentations in order to study effects of changing the co-clustering algorithm and consider the application of weighting techniques to the edges of our bipartite graph structure.

We conclude this thesis in the following chapter where we will summarize the contributions of this work and give a brief glimpse of perspectives that we deem interesting w.r.t. our approaches and to the domain in general.

Chapter 5

General Conclusions

In this final chapter, we revisit the main contributions of the work presented in this thesis and discuss some prospects on possible extensions and on moving object trajectory analysis in general.

5.1 Contributions and Main Results

In this thesis, we addressed two main problems: (i) sampling moving object trajectory data streams, and (ii) clustering trajectory data in road network environments.

Sampling trajectories must be handled intelligently in order to reduce the size of the data while still preserving the maximum of their features and spatiotemporal characteristics. This problem comes with extra challenges in streaming environments since the proposed techniques must be capable of handling the data on-the-fly and have a sufficiently low complexity in order to handle the data load efficiently. If conducted efficiently and effectively, sampling can help alleviate storage and calculation time requirements.

Clustering is an unsupervised learning task often used to discover the presence of groups of observations that behaved similarly within a given population. Application of clustering on trajectory data in road network environments can lead to the extraction of useful knowledge about mobility trends and the characterization of traffic in the network. One of the main challenges of this problem is to incorporate the constraints of the underlying road network in the similarity assessment and the clustering process.

Our contributions and main results w.r.t. these problems are summarized in what follows.

STSS, a Lightweight Algorithm for Sampling Trajectory Streams with Deterministic Error Bounds

To address the first problem, we proposed the SpatioTemporal Stream Sampling (STSS) algorithm, a lightweight trajectory sampling algorithm based on linear

prediction (Chapter 2). The main idea behind STSS is to try to capture the moving object's current behavior and use it for predicting its forthcoming positions. As long as these positions adhere to the registered trend, they can be discarded without considerably altering the features of the original data (Section 2.3.1).

Besides from benefiting from a low time complexity, the algorithm also guarantees an upper bound for the errors resulting from the lossy compression process (Section 2.3.2). Moreover, this error bound is directly controllable by the user through the only parameter required by the algorithm. All of the above qualities make STSS suitable for use in streaming environments where it can serve not only to summarize and keep a compressed history of incoming trajectories but also as a load shedding mechanism to help the system deal with the high arrival rate of the data. Alternatively, STSS can be deployed directly on the moving objects' side in order to alleviate the charge of the central system.

Through experimental results, we showed that STSS achieves decent results but is slightly outperformed by higher complexity trajectory sampling algorithms like TD-TR and OPW-TR (Section 2.4).

Clustering Network-Constrained Trajectories Using a Graph-Based Approach

For the second problem, we started by formalizing the network-constrained trajectory clustering problem in which we are interested in partitioning trajectories evolving in a road network environment into disjoint groups that behaved similarly (Section 3.1).

The first challenge was to define a suitable similarity measure for this particular context. Noticing the similarity between symbolically-represented trajectories and the model used for representing documents as bags-of-words in information retrieval, we defined our similarity based on the use of a cosine similarity in combination with a modified version of tf-idf weighting. Our measure compares trajectories based not only on their shared road segments but also based on the relevance of the latter w.r.t. the analyzed dataset in its entirety (Section 3.3.1).

Representing interactions and similarities between entities using graphs is a very interesting idea adopted in approaches such as spectral clustering [100] and successfully tested in the case of trajectories with no motion restrictions in [59]. We borrowed this paradigm to define a graph model for depicting the similarities of network-constrained trajectories (Section 3.3.2). By doing so, we transposed the network-constrained trajectory clustering problem into a graph clustering one. We used a non-parametric modularity-based community detection algorithm to conduct the partitioning of the trajectory similarity graph (Section 3.3.3). The result of the clustering process is a hierarchy of nested trajectory clusters that offer the possibility of exploring the data with

successive layers of refinement. The user of the approach can start with a limited set of coarse trajectory clusters to grasp a quick and general knowledge of the detected trends in the data then proceed to expanding clusters of interest in order to reveal more polished and precise patterns.

We also demonstrated some of the attractive features of this approach through a comprehensive case study (Section 3.3.4.2), evaluated its performance, and compared it with a similar trajectory clustering approach (Section 3.3.5).

Extending the Graph-Based Approach to Road Segment Clustering

By studying the network-constrained trajectory clustering problem, it became clear that road segments are of equal importance as trajectories. Therefore, we also defined the road segment clustering problem in which we desire to detect groups of road segments that are frequently visited together by the same trajectories (Section 3.1). Our intuition behind this is that such clusters can help predict the spreading of traffic jams, detect the presence of hub structures in the road network, etc.

In order to solve this problem, we proceeded by analogy to our trajectory clustering approach (Section 3.4). We used a graph to depict interactions between road segments and used the same clustering algorithm to discover the road segment clusters (Section 3.4.2). We also observed that such clusters are hard to interpret directly and proposed to overcome this shortcoming by analyzing the segment clusters in the light of trajectory clusters (Section 3.4.3.1).

Interestingly, our experimental results showed that modularity-based clustering is probably not the best choice to cluster the road segment similarity graph as it was outperformed by spectral clustering (Section 3.4.4).

Our main objective in the previous two contributions was to prove the interestingness of adopting a graph-based approach to clustering network-constrained trajectory data. Consequently, we consider that the modularity-based algorithm we used in our clustering phase can potentially be interchanged with other graph clustering approaches (we re-discuss this point in the Future Work and Open Issues section of this chapter).

Simultaneous Co-Clustering of Trajectories and Road Segments

The final contribution reported in this thesis is the result of a collaboration with Romain Guigourès and Marc Boullé from Orange Labs. It consists in an approach to simultaneously cluster trajectories and road segments to produce co-clusters involving both entities (Chapter 4). This work was triggered by the observation of the duality between the original trajectory clustering and segment clustering problems we defined earlier. In fact, partitioning trajectories

implicitly induces a road segment partition and, vice versa, partitioning road segments induces a trajectory partition.

Instead of using two separate similarity graphs, we proposed a bipartite graph representation of interactions between trajectories and road segments (Section 4.2). We used a blockmodeling method to co-cluster the adjacency matrix of this bipartite graph (Section 4.2.2).

We also illustrated how the approach can be used concretely to study interactions between trajectories and road segments, and how the traffic can be characterized accordingly (Section 4.3). Through our experimental study (Section 4.4.2), we compared clusters produced by co-clustering to those obtained by the univariate approaches we introduced in Chapter 3. This experimental study also revealed that the cluster structures discovered by both approaches are quite different from one another which is mainly due to the fundamental difference between their functioning.

5.2 Future Work and Open Issues

We now discuss possible extensions and open issues that we would like to address in future work.

Multi-Trajectory Compression Techniques

In Chapter 2 we focused on sampling trajectories on an individual basis (each trajectory is sampled separately). It would be interesting to investigate multi-trajectory compression in which the incoming trajectories are cross-compared in order to detect redundancies or nearby moving objects and take advantage of this information to reduce the size of the data (for example, when several moving objects are close at a given time, only one position is kept to summarize their whereabouts). Of course, such techniques should preferably guarantee deterministic bounds for the resulting compression errors and must be efficient if they are conceived with streaming environments in mind.

Experimentation with Synthetic Datasets Generated Using More Sophisticated Models

It is worth noting that experimenting with synthetic data is of particular interest in road planning. In fact, rather than proceeding by trial and error, the planned road network's graph can be used to generate trajectories to which clustering can be applied in order to reveal the presence of potential problems (such as congestion zones). The results can then be used to alter the initial plan and re-test it before construction even begins.

The synthetic datasets we used to test our approaches were generated using either the Brinkhoff generator [6] or our trajectory cluster generation strategy (described in Chapter 1) both of which use rather simple movement models.

In future work, we will consider the use of the SUMO [20] simulator since it seems to implement more realistic motion models.

Experimentation on Real Data

One of the obvious extensions to our work on clustering trajectory data is to test and evaluate the performances of our propositions on real datasets. Although not reported here, an attempt was made in this direction during the course of this work. We retrieved the Beijing taxis dataset[1] proposed as part of the GeoLife project. We then implemented the map-matching algorithm described in [9] and tried to match the data to a graph representation of the Beijing road network that we extracted from OpenStreetMap data. However, this attempt resulted in poorly-matched trajectories and had to be dropped.

Studying the Effects of Varying the Clustering Algorithms

The main contribution of this thesis is on the methodological level since we tried to show the interestingness of studying various network-constrained trajectory data clustering problems under a graph perspective. We chose modularity-based clustering for the approaches presented in Chapter 3 based on its popularity and the good results it yields in practice [98]. We chose the MODL blockmodeling approach in Chapter 4 because it was applied successfully in the context of spatiotemporal data [129] and to take advantage of the visual aid provided by the Khiops tool [131].

In the future, we intend to experiment with other graph clustering and co-clustering algorithms in order to test their effect on the discovered partitions. Potential candidates in the case of co-clustering include the approaches presented in [118, 117] since they were originally proposed in the context of word/document clustering which bears close resemblances to trajectory/segment clustering.

Integration of Time in the Clustering Process

Our clustering approaches are solely based on the spatial features of the analyzed trajectories. Consequently, one possible extension is to include support of the temporal dimension. In the case of approaches presented in Chapter 3, this can be undertaken by redefining the similarity measures in order to include time. This was attempted in [71, 21] where the spatial similarity and temporal similarity between two trajectories are calculated independently then combined together into one spatiotemporal similarity (using a weighted sum or a product).

[1]The dataset can be found here: `http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/default.aspx`.

The key difficulty in such approaches is to find a good accord between the spatial and the temporal part of the similarity measure.

We believe that a better approach to including time in this context is by following on the idea presented in [11] in which the total lifespan of the analyzed trajectory dataset is divided into periods, and clustering is conducted on each period separately. This idea is in fact particularly attractive for traffic analysis since some periods (e.g. early morning when people commute to work and late evening when they commute back home) are more important than others.

In the context of co-clustering, including time can be handled quite differently. In fact, time can be simply integrated as an additional variable (besides trajectories and road segments) and tri-clustering (using MODL or other alternatives) can be applied similarly to the work described in [129].

Dynamic Road Network Models

The graph model used to schematize the road network in most existing work (including ours) is quite simplistic. For instance, it supposes that the road network is static and does not change over the course of time. In reality, this is hardly the case. As a counterexample, reversible direction lanes (which are lanes that can be traveled in one direction or the other depending on certain conditions) are often used to regulate traffic during rush hours. Granted, such lanes can be represented by two separate edges in the road network's graph but it might be interesting to investigate alternative models, probably using dynamically evolving graphs, where the road network's graph evolves in time (as to depict certain roads being temporarily closed or changes in the travel direction on some portions, etc.). It might also be interesting to study the impact of such dynamic models on the clustering process and if they can be directly integrated in it.

Evaluation and Comparison of Trajectory Clustering Algorithms

One of the challenges encountered during this work was to come up with a way to evaluate the clustering quality of our approaches and compare them to existing methods. In fact, in a considerable number of existing approaches, the authors start by presenting a baseline naive approach then present enhancements (consisting mainly in the use of heuristics) to reduce their computational cost. The experimental studies presented in such cases mainly revolved around this aspect and the evaluation of the quality of the produced clusters is rarely addressed. Moreover, existing methods are rarely cross-compared mainly due to the fact that they are based on different formulations of a problem that is fundamentally the same.

The sum of average intra-cluster overlaps we defined for both trajectory clusters and segment clusters (which are inspired by the traditional intra-clusters inertia in the clustering of static data) were an attempt that we made

in this direction. We believe that this area must receive more attention in the future in order to come up with robust criteria that allow objective evaluation of trajectory clustering algorithms.

Predicting Social Relationships Using Trajectory Data Analysis

Novel location-based services and applications are being invented every day. Today, almost every single social network offers its users the possibility to share their position with their friends and acquaintances. Some social networks, such as Foursquare, are even solely based on location-sharing. The mechanism at hand here is quite simple: you have an existing list of friends that you already know and with whom you share information about your whereabouts (if you want to, that is).

Recently, Niantic Labs (an internal startup within Google) launched Ingress [3], a location-based augmented reality massively multiplayer online game. In Ingress, two factions (the Enlightened and the Resistance) compete to capture portals and link them to form control fields (basically, a portal corresponds to a point of interest, generally a statue, a well-known monument, etc.). Players must be within immediate physical proximity of a given portal in order to interact with it which makes the game location-oriented. When the game was launched, people started playing individually by exploring their district and maintaining their own "turf." However, the game's logic quickly pushed players into playing as groups and moving together for extended periods. Over the course of time, perfect strangers socialized and started forming friendships (and even couples) and sharing other social activities that are unrelated to the game. Some players that interacted heavily with others even gained a certain notoriety and became local leaders of their factions. In other words, social relationships in this case were formed based on sharing the same location-based activity and moving together (as clusters) frequently.

To the data analyst, this is an opportunity to define a whole set of research problems at the junction of sociology and trajectory data analysis. For instance, if we suppose that the trajectories of Ingress players can be collected (either as complete GPS logs or simply as interactions with portals), can we predict the formation of social relationships based on mining this data and discovering groups of individuals that interacted with the same portals at very close intervals? Are we able to tell if a given player can potentially become the leader of his faction at the local scale by detecting that he heavily interacts with several groups of other members of the same faction? These are questions among many others that may arise in the light of this new tendency.

Bibliography

[1] "Waze: Free community-based mapping, traffic & navigation app." `http://www.waze.com/` (retrieved June 14, 2013).

[2] "Openstreetmap." `http://www.openstreetmap.org/` (retrieved June 14, 2013).

[3] "Ingress." `http://www.ingress.com/` (retrieved June 17, 2013).

[4] F. Giannotti and D. Pedreschi, eds., *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*. Springer, 2008.

[5] D. Schrank, B. Eisele, and T. Lomax, "Tti's 2012 urban mobility report," tech. rep., Texas A&M Transportation Institute, 2012.

[6] T. Brinkhoff, "A framework for generating network-based moving objects," *Geoinformatica*, vol. 6, pp. 153–180, June 2002.

[7] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pp. 853–864, VLDB Endowment, 2005.

[8] A. Kharrat, I. S. Popa, K. Zeitouni, and S. Faiz, "Clustering algorithm for network constraint trajectories," in *SDH*, Lecture Notes in Geoinformation and Cartography, pp. 631–647, Springer, 2008.

[9] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate gps trajectories," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, (New York, NY, USA), pp. 352–361, ACM, 2009.

[10] G.-P. Roh and S.-w. Hwang, "Nncluster: An efficient clustering algorithm for road network trajectories," in *Database Systems for Advanced Applications*, vol. 5982 of *Lecture Notes in Computer Science*, pp. 47–61, Springer Berlin - Heidelberg, 2010.

[11] A. Kharrat, I. S. Popa, K. Zeitouni, and S. Faiz, "Caractérisation de la densité de trafic et de son évolution à partir de trajectoires d'objets mobiles," in *UbiMob* (D. Menga and F. Sedes, eds.), vol. 394 of *ACM International Conference Proceeding Series*, pp. 33–40, ACM, 2009.

[12] H. Güting, T. de Almeida, and Z. Ding, "Modeling and querying moving objects in networks," *The VLDB Journal*, vol. 15, pp. 165–190, June 2006.

[13] I. Sandu-Popa and K. Zeitouni, "Modeling and querying mobile location sensor data," in *The Fourth International Conference on Advanced Geographic Information Systems, Applications, and Services (GEOProcessing 2012)*, pp. 222–231, Feb. 2012.

[14] "Geographic privacy-aware knowledge discovery and delivery (geopkdd)." `http://www.geopkdd.eu/` (site offline as of June 10, 2013).

[15] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Algorithms for nearest neighbor search on moving object trajectories," *Geoinformatica*, vol. 11, pp. 159–193, June 2007.

[16] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proceedings of the 18th international conference on World wide web*, WWW '09, (New York, NY, USA), pp. 791–800, ACM, 2009.

[17] "Real trajectory data of trips in the cook county and the dupage county of illinois." `http://www.cs.uic.edu/~boxu/mp2p/gps_data.html` (retrieved June 12, 2013).

[18] "Spatial reference epsg projection 4326 - wgs 84." `http://spatialreference.org/ref/epsg/4326/` (retrieved June 6, 2013).

[19] J.-M. Saglio and J. Moreira, "Oporto: A realistic scenario generator for moving objects," *GeoInformatica*, vol. 5, no. 1, pp. 71–93, 2001.

[20] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, (Barcelona, Spain), pp. 63–68, October 2011.

[21] J.-W. Chang, R. Bista, Y.-C. Kim, and Y.-K. Kim, "Spatio-temporal similarity measure algorithm for moving objects on spatial networks," in *Proceedings of the 2007 international conference on Computational science and its applications - Volume Part III*, ICCSA'07, (Berlin, Heidelberg), pp. 1165–1178, Springer-Verlag, 2007.

[22] Y. Xia, G.-Y. Wang, X. Zhang, G.-B. Kim, and H.-Y. Bae, "Research of spatio-temporal similarity measure on network constrained trajectory data," in *Proceedings of the 5th international conference on Rough set and knowledge technology*, RSKT'10, (Berlin, Heidelberg), pp. 491–498, Springer-Verlag, 2010.

[23] N. Meratnia and R. A. de By, "Spatiotemporal compression techniques for moving point objects," in *EDBT* (E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, eds.), vol. 2992 of *Lecture Notes in Computer Science*, pp. 765–782, Springer, 2004.

[24] L. Golab and M. T. Özsu, "Issues in data stream management," *SIGMOD Rec.*, vol. 32, pp. 5–14, June 2003.

[25] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, "Stream: The stanford stream data manager," in *SIGMOD Conference*, p. 665, 2003.

[26] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing," in *SIGMOD Conference*, p. 668, 2003.

[27] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik, "Aurora: a new model and architecture for data stream management," *VLDB J.*, vol. 12, no. 2, pp. 120–139, 2003.

[28] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spatscheck, "Gigascope: high performance network monitoring with an SQL interface," in *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), p. 623, ACM Press, 2002.

[29] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "Niagaracq: A scalable continuous query system for internet databases," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA* (W. Chen, J. F. Naughton, and P. A. Bernstein, eds.), pp. 379–390, ACM, 2000.

[30] L. Liu, C. Pu, and W. Tang, "Continual queries for internet scale event-driven information delivery," *IEEE Trans. on Knowl. and Data Eng.*, vol. 11, pp. 610–628, July 1999.

[31] G. Hébrail, "Data stream management and mining," in *MMDSS*, 2007.

[32] F. Clérot, B. Csernel, and G. Hébrail, "Tutorial gestion et fouille de flux de données," in *EGC Conference*, 2007.

[33] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," Technical Report 2002-19, Stanford InfoLab, 2002.

[34] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, pp. 182–209, Sept. 1985.

[35] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, (London, UK, UK), pp. 693–703, Springer-Verlag, 2002.

[36] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[37] S. Guha, N. Koudas, and K. Shim, "Data-streams and histograms," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, (New York, NY, USA), pp. 471–475, ACM, 2001.

[38] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Fast, small-space algorithms for approximate histogram maintenance," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, (New York, NY, USA), pp. 389–398, ACM, 2002.

[39] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "One-pass wavelet decompositions of data streams," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, pp. 541–554, Mar. 2003.

[40] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[41] B. Csernel, F. Clérot, and G. Hébrail, "Streamsamp: data stream classification over tilted windows through sampling," in *International Workshop on Knowledge Discovery from Data Streams (in conjunction with ECML-PKDD)*, 2006.

[42] N. Meratnia and R. A. de By, "A new perspective on trajectory compression techniques," in *Proceedings of ISPRS DMGIS'2003*, 2003.

[43] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," in *The Canadian Cartographer 10(2)*, pp. 112–122, 1973.

[44] J. Hershberger and J. Snoeyink, "Speeding up the douglas-peucker line-simplification algorithm," in *Proc. 5th Intl. Symp. on Spatial Data Handling*, pp. 134–143, 1992.

[45] M. Potamias, K. Patroumpas, and T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, SSDBM '06, (Washington, DC, USA), pp. 275–284, IEEE Computer Society, 2006.

[46] M. Potamias, K. Patroumpas, and T. Sellis, "Amnesic online synopses for moving objects," in *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, (New York, NY, USA), pp. 784–785, ACM, 2006.

[47] T. Palpanas, T. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel, "Online amnesic approximation of streaming time series," in *In ICDE*, pp. 338–349, 2004.

[48] Y. Theodoridis, "The athens trucks dataset." `http://www.chorochronos.org/?q=node/5` (retrieved February 27, 2013).

[49] "Masternaut's website." `http://www.masternaut.com/` (retrieved February 27, 2013).

[50] P. Laube, M. van Kreveld, and S. Imfeld, "Finding remo - detecting relative motion patterns in geospatial lifelines," in *11th International Symposium on Spatial Data Handling*, pp. 201–214, 2004.

[51] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle, "Reporting flock patterns," in *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, (London, UK), pp. 660–671, Springer-Verlag, 2006.

[52] M. R. Vieira, P. Bakalov, and V. J. Tsotras, "On-line discovery of flock patterns in spatio-temporal data," in *GIS '09: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, (New York, NY, USA), pp. 286–295, ACM, 2009.

[53] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *GIS '06: Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, (New York, NY, USA), pp. 35–42, ACM, 2006.

[54] G. Al-Naymat, S. Chawla, and J. Gudmundsson, "Dimensionality reduction for long duration and complex spatio-temporal queries," in *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, (New York, NY, USA), pp. 393–397, ACM, 2007.

[55] H. Jeung, H. T. Shen, and X. Zhou, "Convoy queries in spatio-temporal databases," in *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, (Washington, DC, USA), pp. 1457–1459, IEEE Computer Society, 2008.

[56] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 1068–1080, 2008.

[57] H. Yoon and C. Shahabi, "Accurate discovery of valid convoys from moving object trajectories," in *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, ICDMW '09, (Washington, DC, USA), pp. 636–643, IEEE Computer Society, 2009.

[58] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 593–604, ACM, 2007.

[59] D. Guo, S. Liu, and H. Jin, "A graph-based approach to vehicle trajectory analysis," *J. Locat. Based Serv.*, vol. 4, pp. 183–199, September 2010.

[60] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 491–502, ACM, 2005.

[61] D. J. Berndt and J. Clifford, "Finding patterns in time series: a dynamic programming approach," in *Advances in knowledge discovery and data mining* (U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds.), pp. 229–248, Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996.

[62] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, (Washington, DC, USA), pp. 201–208, IEEE Computer Society, 1998.

[63] M. Vlachos, D. Gunopoulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, (Washington, DC, USA), pp. 673–684, IEEE Computer Society, 2002.

[64] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pp. 792–803, VLDB Endowment, 2004.

[65] B. Lin and J. Su, "One way distance: For shape based similarity search of moving object trajectories," *GeoInformatica*, vol. 12, no. 2, pp. 117–142, 2008.

[66] M. van Kreveld and J. Luo, "The definition and computation of trajectory and subtrajectory similarity," in *GIS '07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, (New York, NY, USA), pp. 1–4, ACM, 2007.

[67] K. Buchin, M. Buchin, M. van Kreveld, and J. Luo, "Finding long and similar parts of trajectories," in *GIS '09: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, (New York, NY, USA), pp. 296–305, ACM, 2009.

[68] J.-R. Hwang, H.-Y. Kang, and K.-J. Li, "Spatio-temporal similarity analysis between trajectories on road networks," in *ER (Workshops)*, Lecture Notes in Computer Science, pp. 280–289, Springer, 2005.

[69] J.-R. Hwang, H.-Y. Kang, and K.-J. Li, "Searching for similar trajectories on road networks using spatio-temporal similarity," in *Advances in Databases and Information Systems* (Y. Manolopoulos, J. Pokorný, and T. Sellis, eds.), vol. 4152 of *Lecture Notes in Computer Science*, pp. 282–295, Springer Berlin Heidelberg, 2006.

[70] H. Zhao, Q. Han, H. Pan, and G. Yin, "Spatio-temporal similarity measure for trajectories on road networks," in *Proceedings of the 2009 Fourth International Conference on Internet Computing for Science and Engineering*, ICICSE '09, (Washington, DC, USA), pp. 189–193, IEEE Computer Society, 2009.

[71] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic, and S. Djordjevic-Kajan, "Trajectory similarity search in spatial networks," in *Proceedings of the 10th International Database Engineering and Applications Symposium*, (Washington, DC, USA), pp. 185–192, IEEE Computer Society, 2006.

[72] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[73] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 9th ed., Mar. 1990.

[74] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: A new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

[75] M. Ester, H.-P. Kriegel, J. S, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD'96*, pp. 226–231, 1996.

[76] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," *SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, 1999.

[77] Y. Li, J. Han, and J. Yang, "Clustering moving objects," in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 617–622, ACM, 2004.

[78] C. S. Jensen, D. Lin, and B. C. Ooi, "Continuous clustering of moving objects," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 9, pp. 1161–1174, 2007.

[79] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *Proceedings of the 9th international conference on Advances in Spatial and Temporal Databases*, SSTD'05, (Berlin, Heidelberg), pp. 364–381, Springer-Verlag, 2005.

[80] P. D. Grünwald, I. J. Myung, and M. A. Pitt, *Advances in Minimum Description Length: Theory and Applications (Neural Information Processing).* The MIT Press, 2005.

[81] J.-G. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, (Washington, DC, USA), pp. 140–149, IEEE Computer Society, 2008.

[82] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 1081–1094, 2008.

[83] Z. Li, J.-G. Lee, X. Li, and J. Han, "Incremental clustering for trajectories," in *DASFAA (2)* (H. Kitagawa, Y. Ishikawa, Q. Li, and C. Watanabe, eds.), vol. 5982 of *Lecture Notes in Computer Science*, pp. 32–46, Springer, 2010.

[84] D. Guo, "Flow mapping and multivariate visualization of large spatial interaction data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1041–1048, 2009.

[85] I. R. Brilhante, J. A. F. de Macedo, C. Renso, and M. A. Casanova, "Trajectory data analysis using complex networks," in *Proceedings of the 15th Symposium on International Database Engineering & Applications*, IDEAS '11, (New York, NY, USA), pp. 17–25, ACM, 2011.

[86] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares, "A clustering-based approach for discovering interesting places in trajectories," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, (New York, NY, USA), pp. 863–868, ACM, 2008.

[87] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. Sellis, "On-line discovery of hot motion paths," in *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, (New York, NY, USA), pp. 392–403, ACM, 2008.

[88] N. Pelekis, I. Kopanakis, E. Kotsifakos, E. Frentzos, and Y. Theodoridis, "Clustering trajectories of moving objects in an uncertain world," in *ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 417–427, IEEE Computer Society, 2009.

[89] M. Nanni and D. Pedreschi, "Time-focused clustering of trajectories of moving objects," *J. Intell. Inf. Syst.*, vol. 27, no. 3, pp. 267–289, 2006.

[90] S. Gaffney and P. Smyth, "Trajectory clustering with mixtures of regression models," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, (New York, NY, USA), pp. 63–72, ACM, 1999.

[91] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979.

[92] W. Liu, Z. Wang, and J. Feng, "Continuous clustering of moving objects in spatial networks," in *KES '08: Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II*, (Berlin, Heidelberg), pp. 543–550, Springer-Verlag, 2008.

[93] C. Lai, L. Wang, J. Chen, X. Meng, and K. Zeitouni, "Effective density queries for moving objects in road networks," in *APWeb/WAIM* (G. Dong, X. Lin, W. W. 0011, Y. Yang, and J. X. Yu, eds.), vol. 4505 of *Lecture Notes in Computer Science*, pp. 200–211, Springer, 2007.

[94] I. Ntoutsi, N. Mitsou, and G. Marketos, "Traffic mining in a road-network: How does the traffic flow?," *Int. J. Bus. Intell. Data Min.*, vol. 3, no. 1, pp. 82–98, 2008.

[95] J.-I. Won, S.-W. Kim, J.-H. Baek, and J. Lee, "Trajectory clustering in road network environment," in *IEEE Symposium on Computational Intelligence and Data Mining, 2009. CIDM '09*, pp. 299 –305, 30 2009-april 2 2009.

[96] J. Chen, P. Chen, Q. Huo, and X. Xu, "Clustering network-constrained uncertain trajectories," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, vol. 3, pp. 1657 –1662, july 2011.

[97] S. E. Schaeffer, "Survey: Graph clustering," *Comput. Sci. Rev.*, vol. 1, pp. 27–64, Aug. 2007.

[98] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[99] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, Sept. 2007.

[100] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, pp. 395–416, Dec. 2007.

[101] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[102] F. Rossi and N. Villa-Vialaneix, "Représentation d'un grand réseau à partir d'une classification hiérarchique de ses sommets," *Journal de la Société Française de Statistique*, vol. 152, pp. 34–65, 12 2011.

[103] A. Noack and R. Rotta, "Multi-level algorithms for modularity clustering," in *Proceedings of the 8th International Symposium on Experimental Algorithms*, SEA '09, (Berlin, Heidelberg), pp. 257–268, Springer-Verlag, 2009.

[104] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[105] Y. Zhao and G. Karypis, "Criterion functions for document clustering: Experiments and analysis," tech. rep., 2002.

[106] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, pp. 193–218, 1985.

[107] S. Wagner and D. Wagner, "Comparing Clusterings – An Overview," Tech. Rep. 2006-04, Universität Karlsruhe (TH), 2007.

[108] W. M. Rand, "Objective Criteria for the Evaluation of Clustering Methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[109] S. Fortunato and M. Barthélemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, pp. 36–41, Jan. 2007.

[110] B. Mirkin, *Mathematical classification and clustering*. Kluwer Academic Press, 1996.

[111] J. Hartigan, "Direct Clustering of a Data Matrix," *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 123–129, 1972.

[112] G. Govaert, "Algorithme de classification d'un tableau de contingence," in *First international symposium on data analysis and informatics*, (Versailles), pp. 487–500, INRIA, 1977.

[113] T. Eckes and P. Orlik, "An error variance approach to two-mode hierarchical clustering," *Journal of Classification*, vol. 10, no. 1, pp. 51–74, 1993.

[114] B. Mirkin, P. Arabie, and L. Hubert, "Additive two-mode clustering: The error-variance approach revisited," *Journal of Classification*, vol. 12, no. 2, pp. 243–263, 1995.

[115] G. Govaert and M. Nadif, "Clustering with block mixture models," *Pattern Recognition*, vol. 36, pp. 463–473, 2003.

[116] M. Nadif and G. Govaert, "Model-based co-clustering for continuous data," in *ICMLA*, pp. 175–180, 2010.

[117] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[118] I. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 269–274, ACM, 2001.

[119] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, "Spectral biclustering of microarray data: coclustering genes and conditions," *Genome research*, vol. 13, no. 4, pp. 703–716, 2003.

[120] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991.

[121] I. S. Dhillon, S. Mallela, and D. Modha, "Information-theoretic co-clustering," in *KDD '03*, pp. 89–98, 2003.

[122] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *Journal of Machine Learning Research*, vol. 6, pp. 1705–1749, Dec. 2005.

[123] R. Breiger, S. Boorman, and P. Arabie, "An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling," *Journal of Mathematical Psychology*, vol. 12, no. 3, pp. 328–383, 1975.

[124] P. Arabie, S. Schleutermann, J. Daws, and L. Hubert, "Marketing applications of sequencing and partitioning of nonsymmetric and/or two-mode matrices," in *Data, Expert Knowledge and Decisions* (W. Gaul and M. Schader, eds.), pp. 215–224, Springer Berlin Heidelberg, 1988.

[125] M. Boullé, "Data grid models for preparation and modeling in supervised learning," in *Hands-On Pattern Recognition: Challenges in Machine Learning, vol. 1*, pp. 99–130, Microtome, 2011.

[126] M. Boullé, "Sélection bayésienne de modéles avec prior dépendant des données," in *Extraction et gestion des connaissances (EGC'2012)*, pp. 29–34, 2012.

[127] X. Wang, K. T. Ma, G.-W. Ng, and W. Grimson, "Trajectory analysis and semantic region modeling using a nonparametric bayesian model," in *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, pp. 1–8, 2008.

[128] S. C. Madeira and A. L. Olivieira, "Biclustering algorithms for biological data analysis: a survey," *Computational Biology and Bioinformatics*, vol. 1, no. 1, pp. 24–45, 2004.

[129] R. Guigourès, M. Boullé, and F. Rossi, "A triclustering approach for time evolving graphs," in *Co-clustering and Applications, IEEE 12th International Conference on Data Mining Workshops (ICDMW 2012)*, (Brussels, Belgium), pp. 115–122, 12 2012.

[130] P. Hansen and N. Mladenovic, "Variable neighborhood search: Principles and applications," *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

[131] "Khiops: France telecom tool for mining large databases." `http://www.khiops.com/` (retrieved June 3, 2013).

List of Publications

In Proceedings of International Conferences and Workshops

[1] M. K. El Mahrsi, C. Potier, G. Hébrail, and F. Rossi, "Spatiotemporal sampling for trajectory streams," in *SAC'10: Proceedings of the 2010 ACM Symposium on Applied Computing,* (New York, NY, USA), pp. 1627-1628, ACM, 2010. (Poster)

[2] M. K. El Mahrsi and F. Rossi, "Modularity-Based Clustering for Network-Constrained Trajectories," in *Proceedings of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2012)*, (Bruges, Belgium), pp. 471-476, Apr. 2012.

[3] M. K. El Mahrsi and F. Rossi, "Graph-Based Approaches to Clustering Network-Constrained Trajectory Data," in *Proceedings of the Workshop on New Frontiers in Mining Complex Patterns (NFMCP 2012)*, (Bristol, UK), pp. 184-195, Sept. 2012.

In Proceedings of National Conferences

[4] M. K. El Mahrsi and F. Rossi, "Clustering par optimisation de la modularité pour trajectoires d'objets mobiles," in *Actes des 8èmes journées francophones Mobilité et Ubiquité* (N. Roose, Philippe et Rouillon-Couture, ed.), (Anglet, France), pp. 12-22, Cépaduès Éditions, Jun. 2012.

[5] M. K. El Mahrsi, R. Guigourès, F. Rossi, and M. Boullé, "Classifications croisées de données de trajectoires contraintes par un réseau routier," in *Actes de 13ème Conférence Internationale Francophone sur l'Extraction et gestion des connaissances (EGC'2013)* (C. Vrain, A. Péninou, and F. Sedes, eds.), vol. RNTI-E-24, (Toulouse, France), pp. 341-352, Hermann-Éditions, Feb. 2013.

Book Chapters

[6] M. K. El Mahrsi and F. Rossi, "Graph-based approaches to clustering network-constrained trajectory data," in *New Frontiers in Mining Complex Patterns* (A. Appice, M. Ceci, C. Loglisci, G. Manco, E. Masciari, and Z. Ras, eds.), vol. 7765 of *Lecture Notes in Computer Science*, pp. 124-137, Springer Berlin Heidelberg, 2013.

Analyse et fouille de données de trajectoires d'objets mobiles

Mohamed Khalil EL MAHRSI

RÉSUMÉ : Dans cette thèse, nous explorons deux problèmes de recherche liés à la gestion et à la fouille de données de trajectoires d'objets mobiles.

Dans un premier temps, nous étudions l'échantillonnage de flux de trajectoires. Les appareils de géo-localisation modernes sont capables d'enregistrer et de transmettre leurs coordonnées géographiques à un taux très élevé. Garder l'intégralité des trajectoires capturées grâce à ces terminaux peut s'avérer coûteux tant en espace de stockage qu'en temps de calcul. L'élaboration de techniques d'échantillonnage adaptées devient alors primordiale afin de réduire la volumétrie des données en supprimant certaines positions (jugées inutiles ou redondantes) tout en veillant à préserver le maximum des caractéristiques spatiotemporelles des trajectoires originales. Dans le contexte de flux de données, ces techniques doivent en plus être exécutées « à la volée » et s'adapter au caractère à la fois continu et éphémère des données. Afin de répondre à ces besoins, nous proposons l'algorithme STSS (*Spatiotemporal Stream Sampling*). STSS bénéficie d'une faible complexité temporelle et garantit une borne supérieure pour les erreurs commises lors de l'échantillonnage. Nous présentons également une étude expérimentale à travers laquelle nous montrons les performances de notre proposition tout en la comparant à d'autres approches proposées dans la littérature.

La deuxième problématique étudiée dans le cadre de ce travail est celle de la classification non supervisée (ou *clustering*) de trajectoires contraintes par un réseau routier. La majorité des travaux traitant du clustering de trajectoires se sont intéressés au cas où ces dernières évoluent librement dans un espace Euclidien. Ces travaux n'ont donc pas pris en considération l'éventuelle présence d'un réseau sous-jacent au mouvement, dont les contraintes jouent un rôle primordial dans l'évaluation de la similarité entre trajectoires. Nous proposons trois approches pour traiter ce cas. La première approche se focalise sur la découverte de groupes de trajectoires ayant parcouru les mêmes parties du réseau routier. La deuxième approche vise à grouper des segments routiers visités très fréquemment par les mêmes trajectoires. Quant à la troisième approche, elle combine les deux aspects afin d'effectuer un co-clustering simultané des trajectoires et des segments routiers. Nous illustrons nos approches à travers divers cas d'étude afin de démontrer comment elles peuvent servir à caractériser le trafic routier et les dynamiques de mouvement dans le réseau routier. Nous réalisons des études expérimentales afin d'évaluer les performances de nos propositions.

MOTS-CLÉS : objets mobiles, trajectoires, réseau routier, échantillonnage, spatiotemporel, flux de données, similarité, classification non supervisée.

ABSTRACT: In this thesis, we explore two problems related to managing and mining moving object trajectories.

First, we study the problem of sampling trajectory data streams. Modern location-aware devices are capable of capturing and transmitting their position at very high rates. Storing the entirety of the trajectories provided by such devices can entail severe storage and processing overheads. Therefore, adapted sampling techniques are necessary in order to discard unneeded positions and reduce the size of the trajectories while still preserving their key spatiotemporal features. In streaming environments, this process needs to be conducted "on-the-fly" since the data are transient and arrive continuously. To this end, we introduce a new sampling algorithm called Spatiotemporal Stream Sampling (STSS). This algorithm is computationally-efficient and guarantees an upper bound for the approximation error introduced during the sampling process. Experimental results show that STSS achieves good performances and can compete with more sophisticated and costly approaches.

The second problem we study is clustering trajectory data in road network environments. Most of prior work assumed that moving objects can move freely in an Euclidean space and did not consider the presence of an underlying road network and its influence on evaluating the similarity between trajectories. We present three approaches to clustering such data: the first approach discovers clusters of trajectories that traveled along the same parts of the road network; the second approach is segment-oriented and aims to group together road segments based on trajectories that they have in common; the third approach combines both aspects and simultaneously clusters trajectories and road segments. Through extensive case studies, we show how these approaches can be used to reveal useful knowledge about flow dynamics and characterize traffic in road networks. We also provide experimental results where we evaluate the performances of our propositions.

KEY-WORDS: moving objects, trajectories, road network, spatiotemporal, sampling, data streams, similarity, clustering.

TELECOM ParisTech

EDITE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY