

**MINING TRAJECTORY DATABASES FOR
MULTI-OBJECT MOVEMENT PATTERNS**

HTOO HTET AUNG

(B.C.Sc. (Honours), University of Computer Studies, Yangon)

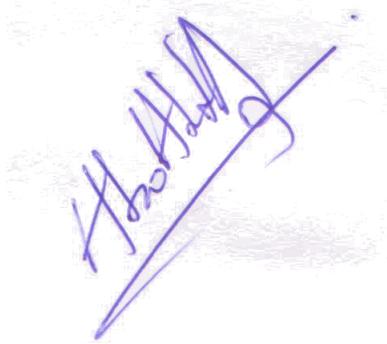
A THESIS SUBMITTED
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2013

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

A handwritten signature in blue ink, appearing to read 'Htoo Htet Aung', written diagonally across the page.

Htoo Htet Aung
May 8th 2013

Acknowledgements

First and foremost, I would like to express a great depth of gratitude to my supervisor, Professor Tan Kian-Lee, a respectable and resourceful scholar, who has provided me with valuable guidance in every stage of my research work including this thesis. Especially when I was weary with worries on the outcomes of my works, be it Qualifying Exam, Graduate Research Proposal, Thesis Proposal or conference paper submissions, his thoughtful reasoning and calm manner had always alleviated my worries and made me achieve a placid state of mind.

I would also like to take this opportunity to thank both members of my thesis advisory committee, namely Professor Wynne Hsu and Professor Lee, Mong Li Janice, who provided insightful comments and suggestions in my Graduate Research Proposal, my Thesis Proposal and this Thesis itself. I would also like to separately mention my thanks to Professor Wynne Hsu, who trusted my abilities and supported my conversion of candidature from a coursework-base programme to a research-base programme for this wonderful opportunity. A special acknowledgement should also be shown to Professor Stéphane Bressan, who provided me with Ships dataset and introduced me with some practical research problems.

Moreover, I must not forget to express my heart-felt thanks to my programming teacher, senior, and friend Zeyar Aung, who helped me with everything in his ability — from trivial matters like application submission to NUS to non-trivial things like occasional discussion, encouragement, and many wonderful meals he provided me with. At the same time, I also would like to say a big “thank you” to Uncle Soe Aung and Auntie Yu Yu Sein for providing me a place-like-home in the weekends.

In addition, I feel strongly thankful to many of my friends both in and out of NUS. I would extend my thanks to my fellow students and researchers (in alphabetical order), Cao Yu, Cao Jianneng, Cao Nan Nan, Chen Ding, Fan Qi, Goh Wei Xiang, Le Thuy Ngoc, Li Luo Cheng, Li Xiaohui, Meduri Venkata Vamsikrishna, Saw Qua Lar, Shen Zhong, Shi Lei, Shwe Aung Zaw, Suraj Pathak, Tran Quoc

Trung, Wang Fangda, Wang Guoping, Wang Zhenkui, Wu Ji, Zeng Zhong, and especially Guo Long, Jonathan Poon, Wu Wei, Xiang Shili, Xiao Qian and Zeng Yong, whom I had a great pleasure to discuss and work with.

Finally, I would like to express my deepest gratitude to my beloved family — my parents, Win Myint Law (Nelson Law) and Phyu Phyu Kyi (Violet Kyi), my younger brother, Khun Thi Ha (William Law), my uncles, Phone Myint (Roland Kyi) and Tin Maung Thein, my aunts, Wah Wah Kyi (Iris Kyi) and Toe Toe Kyi (Pansy Kyi) — for their support and confidence in me and, last but not least, my girlfriend, Ei Thinzar Win.

Table of Contents

Acknowledgements	i
Table of Contents	iii
Summary	vi
List of Tables	viii
List of Figures	x
1 Introduction	1
1.1 Motivation	4
1.1.1 Meetings	4
1.1.2 Frequent Routes	5
1.1.3 Evolving Convoys	7
1.2 Contributions	10
1.2.1 Meetings of Moving Objects	10
1.2.2 Sub-trajectory Cliques and Frequent Routes	10
1.2.3 Dynamic Convoys and Evolving Convoys	11
1.3 Organization	12
2 Overview	14
2.1 Mining Trajectory Databases for Multi-object Movement Patterns	14
2.2 Proposed Mining Problems	19
2.2.1 Finding Closed Meetings of Moving Objects	19
2.2.2 Mining Sub-trajectory Cliques to Extract Frequent Routes	19
2.2.3 Discovery of Evolving Convoys	20
2.3 Platform to Assess the Proposed Algorithms	21
2.3.1 Datasets and Data Cleaning	21
2.3.2 Computational Environment	24
3 Related Works	26
3.1 General Data-mining Techniques	26
3.1.1 Traversing Power-sets	26
3.1.2 Clustering of Data	27
3.2 Multi-object Movement Patterns	29

3.2.1	Meetings	29
3.2.2	Flocks	31
3.2.3	Moving Groups	32
3.2.4	Convoys	32
3.2.5	Moving Clusters	34
3.2.6	Swarm	35
3.2.7	Sub-trajectory Clusters	36
3.2.8	Other Movement Patterns	39
4	Finding Closed MEMOs	40
4.1	Introduction	40
4.2	Finding Closed MEMOs	43
4.3	Algorithms for Finding Closed MEMOs	45
4.3.1	An Apriori-based Closed MEMO Miner	46
4.3.2	An ECLAT-based Closed MEMO Miner	53
4.3.3	A Filter-And-Refinement Closed MEMO Miner	56
4.4	Experimental Evaluations	58
4.4.1	Experiment Setup	58
4.4.2	Results and Analysis	59
4.5	Summary	66
5	Mining Sub-trajectory Cliques to Find Frequent Routes	68
5.1	Introduction	68
5.2	Sub-trajectory Cliques and Frequent Routes	71
5.3	Methods to Mine Sub-trajectory Cliques to Extract Frequent Routes	78
5.3.1	Hardness of Mining Sub-trajectory Cliques from a Trajectory Database	78
5.3.2	Apriori-based Frequent Route Miner	80
5.3.3	Approximation of Sub-trajectory Cliques for Frequent Route Mining	87
5.3.4	A Divide and Conquer Scheme for Scalable Approximation of Sub-trajectory Cliques	90
5.4	Experimental Evaluations	94
5.4.1	Experiment Setup	94
5.4.2	Results and Analysis	94
5.5	Summary	103
6	Discovery of Evolving Convoys	104
6.1	Introduction	104
6.2	Dynamic Convoys and Evolving Convoys	110
6.3	Algorithms to Discover of Evolving Convoys	115
6.3.1	Simple Slice-by-slice Algorithm	115
6.3.2	Interleaved DEC Algorithms	118
6.4	Experimental Evaluations	124
6.4.1	Preliminary Experiments	124
6.4.2	Experiment Setup	125

6.4.3	Results and Analysis	127
6.5	Summary	131
7	Conclusion	132
7.1	Contributions	133
7.1.1	Finding Closed MEMOs	133
7.1.2	Mining Sub-trajectory Cliques to Extract Frequent Routes .	133
7.1.3	Discovery of Evolving Convoys	134
7.2	Future Works	135
7.2.1	Unified Framework for MOMO Patterns	135
7.2.2	Check-in and Social-network Data	136
A	Preliminary Experiments on Convoy Discovery	145
A.1	Experiment Setup	145
A.2	Results and Analysis	147

Summary

In this thesis, we present our studies on “Mining Trajectory Databases for Multi-object Movement Patterns”. A multi-object movement pattern describes the characteristics of a collective-movement performed by multiple objects. Knowledge of these patterns has numerous applications in epidemiology, ecology, preservation of wild-life, traffic monitoring and control, Location-Based Services, marketing, social-studies, and even on-line game development.

We present the research we had conducted to find meeting patterns. Meeting pattern, which is defined as a set of moving objects confined in a fixed spatial area for a period of time, has many applications including traffic control and social studies. However, current literature lacks a thorough study on the discovery of meeting patterns in Trajectory Databases. We (a) introduce MEMO pattern, a new definition of meeting pattern, (b) propose three new algorithms based on a novel data-driven approach to extract closed MEMOs from moving object datasets and (c) implement and evaluated them along with the polynomial-time algorithm previously reported in [23], whose performance has never been evaluated. Experiments using real-world datasets revealed that our filter-and-refinement algorithm outperforms the others in many realistic settings.

We report the research we had performed on finding frequent routes by mining Sub-trajectory cliques (TRAJCLIQS). We had studied techniques to find frequent routes in Trajectory Databases without any prior knowledge of the underlying spatial space. Since mining all TRAJCLIQS is an NP-Complete problem and exact algorithms even from data-driven approach are not feasible, we proposed two approximate algorithms based on the *Apriori* algorithm. Empirical results showed that our proposed algorithms can run faster than the existing polynomial time approximation algorithm appeared in [12] and provide a tighter results. Our experiments also showed that the frequent routes reported by our algorithms are intuitive.

We also had conducted research in finding convoy patterns. Traditionally, a convoy is defined as a set of moving objects that are close to each other for a period of time. Existing techniques, following this traditional definition, cannot find evolving convoys with dynamic members and do not have any monitoring aspect in their design. We propose new concepts of dynamic convoys and evolving convoys, which reflect real-life scenarios, and develop algorithms to discover evolving convoys in an incremental manner.

List of Tables

2.1	Example Predicates and Collective Movements.	17
2.2	Datasets Used to Assess the Proposed Algorithms.	22
3.1	A Comparison of the Traditional Convoy Models.	35
4.1	A Trace of A-miner.	49
4.2	A Partial Trace of E-miner.	55
4.3	The Size of the Datasets after Pre-processing.	59
4.4	Run-time Statistics of FAR-miner in the Experiments.	62
5.1	Records of the Ship Trajectory.	72
5.2	Two Pairs of Re-parametrizations of the Two Sub-trajectories.	75
5.3	A Trace of A-0.	84
5.4	A Trace of A-1.	88
5.5	Parameters and Performance of the Frequent Route Mining Algorithms.	96
5.6	Memory Footprint of Algorithms A-1 and A-2.	99
5.7	Results and Performance of Algorithms A-1 and A-1 (FP).	100
6.1	Maximal Convoys Formed by Five Commuters.	107
6.2	A Partial Trace of the Simple Slice-by-Slice (S^3) Algorithm.	118
6.3	Parameters Used to Assess Convoy Discovery Algorithms.	126
6.4	Datasets and Index Settings Used by the Convoy Discovery Algorithms.	127
6.5	Running Time and Results of Convoy Discovery Algorithms.	128
A.1	Datasets and Experiment Settings Used to Assess Convoy Discovery Algorithms in Preliminary Experiments.	146
A.2	Running Time Comparison of Convoy Discovery Algorithms for Different Datasets in Preliminary Experiments.	147

List of Figures

1.1	Some Movements of Ships Captured by an AIS receiver.	2
1.2	How Convoy Information Improves Players' Experience.	9
2.1	An Example Trajectory Database Containing Four Time-stamps. . .	16
2.2	Mining Multi-object Movement Patterns.	18
2.3	Mining Closed Meetings of Moving Objects.	19
2.4	Mining Sub-trajectory Cliques to Extract Frequent Routes.	20
2.5	Mining Evolving Convoys.	21
2.6	Distances between Two Locations Consecutively Reported.	23
2.7	Comparison of the Taxi Dataset before and after Cleaning.	24
3.1	An Example of Density-based Clustering.	28
3.2	An Instance of Two Overlapping Meetings.	30
3.3	An Example Contrasting a Flock and a Meeting.	31
3.4	An Example Scenario, Where Algorithm CuTS Has False-negatives. .	34
3.5	Comparison of Existing Moving Group Models.	36
4.1	Examples of a MEMO, a Meeting Place, and Two Closed MEMOs. . . .	44
4.2	Movements of Four Objects and the Corresponding Lattice.	49
4.3	The Performance of the Closed MEMO Mining Algorithms.	60
4.4	Impact of the Dataset Size on the Closed MEMO Mining Algorithms. .	61
4.5	Impact of the Parameter r on the Closed MEMO Mining Algorithms. .	63
4.6	Impact of the Parameter m on the Closed MEMO Mining Algorithms. .	64
4.7	Impact of the Parameter w on the Closed MEMO Mining Algorithms. .	64
4.8	MEMOs Discovered by Closed MEMO Mining Algorithms and Density- connected Clusters of Three-dimension GPS Points.	67
5.1	Trajectory of a Ship.	72
5.2	Two Polygonal Curves and Two Pairs of Possible Re-parametrizations. .	74

5.3	A Visualization of a Trajectory Database Containing Four Trajectories.	77
5.4	Conversion of a MAXCLIQUE problem into TRAJCLIQ problem. . . .	81
5.5	How Algorithm A-0 Finds Frequent Routes.	84
5.6	Two Trajectory Segments and Their Corresponding Free-space Cell.	85
5.7	How Algorithm A-2 Divides a Trajectory Database.	91
5.8	Frequent Routes of the Ships Discovered.	97
5.9	Trajectory Clusters and Frequent Routes in the Same Area.	98
5.10	A Trajectory Cluster and Frequent Routes in the Same Area.	99
5.11	All Frequent Routes of the Trucks.	100
5.12	Impact of the Parameter m on Frequent Route Mining Algorithms.	101
5.13	Impact of the Parameter l on the Frequent Route Mining Algorithms.	102
5.14	Impact of the Parameter r on the Frequent Route Mining Algorithms.	102
6.1	Trajectory Database Containing Five Commuters' Movements. . . .	106
6.2	Detailed Movements of the Five Commuters.	107
6.3	The Concept of Convoy Evolution.	112
6.4	Transition between Membership in an Evolving Convoy.	114
6.5	A Visualization of the Example of Five Soldiers' Movements.	115
6.6	A Trajectory Database of Eight Objects' Movements.	118
6.7	Impact of Parameter w on Convoy Discovery Algorithms.	128
6.8	Impact of Parameter k on Convoy Discovery Algorithms.	129
6.9	Impact of Parameter min_pts on Convoy Discovery Algorithms. . .	130
6.10	Impact of Parameter ε on Convoy Discovery Algorithms.	130
A.1	Effect of Parameters w and k on Performance of Convoy Discovery Algorithms during Preliminary Experiments.	148
A.2	Effect of DBSCAN Parameters ε and min_pts on Performance of Convoy Discovery Algorithms during Preliminary Experiments. . .	148
A.3	Effect of Parameter λ on Performance of Convoy Discovery Algorithms during Preliminary Experiments.	149
A.4	Effect of the Nature of the Dataset on the Convoy Discovery Algorithms during Preliminary Experiments.	150

Introduction

A Global Positioning System (GPS) receiver, or a GPS client device, is a location-sensing device that allows its users to access time-stamped locations of the device. Advances in GPS technology enable the user of a GPS client to maintain a highly accurate (up to a few metres) record of the locations he (or the tracked object — such as a naval vessel, a vehicle, or a wild-life, which is tagged with the GPS device) visited in high temporal resolutions and, hence, his detailed movement data.

Since the GPS service was open for civilian use, GPS receivers have been installed in naval vessels (ships) to assist in navigation. The Automatic Identification System (AIS) transmits the time-stamped location data (movement data) obtained from the vessels' on-board GPS receivers to nearby vessels and maritime authorities. The movement data received from the AIS is used to assist the vessels' watch-standing officers and the maritime authorities to track and monitor the movement of the nearby vessels. The maritime authorities often archive the movement data (trajectories) of the ships near their ports in trajectory databases for record-keeping purposes and for further studies of the ships' trajectories to optimize their ports' operations. Figure 1.1 shows one such dataset captured from an AIS receiver in Singapore on September 5, 2011 during 0800 - 0900 hrs.

Similarly, businesses in the public transportation industry (taxi and bus operators) and those in the logistics industry equip their fleets with GPS receivers for management, control, and security purposes. These businesses record and archive

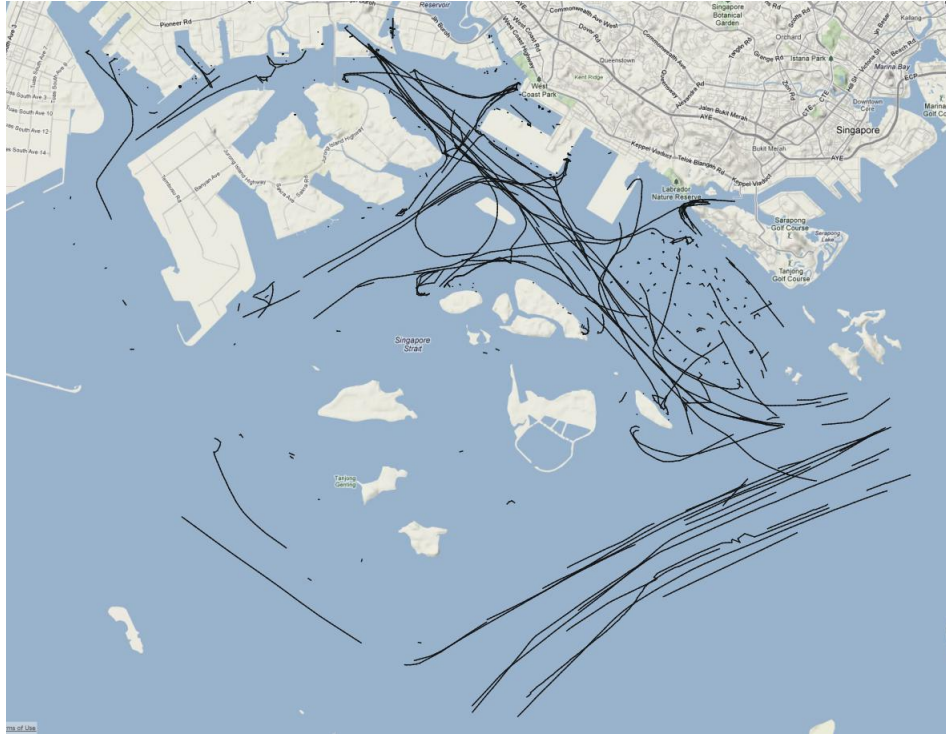


Figure 1.1: Some Movements of Ships Captured by an AIS Receiver in Singapore on September 5, 2011, 08:00 – 09:00.

the movement data of their fleets in trajectory databases for analysis aiming to improve the quality of their services.

Along with high mobile-penetration, the amount of civilians' movement data (trajectories) obtained from GPS devices is also growing larger. Almost all mobile devices (phones and tablets) available on the market include a GPS receiver. On-line GPS track sharing services like Endomondo¹, My Fitness Pal², Every Trail³, and WikiLoc⁴ allow their users to record and publish their own trajectories. These tracking data can be used for recommending travel routes for general public and/or sightseeing routes for tourists by Location-Based Services (LBS).

Moreover, ecologists and marine biologists are looking forward to track the animals they are studying by attaching GPS receivers (and data transmitters) to the

¹<http://www.endomondo.com/>

²<http://www.myfitnesspal.com/>

³<http://www.everytrail.com/>

⁴<http://www.wikiloc.com/>

animals in question [52]. In fact, tracking a small sets of land and sea animals using GPS devices has been successfully demonstrated [2, 50, 51]. Along with more advances in GPS technology and reduction in costs, we expect the scientific community to eventually collect and archive substantial amount of animal movement data in trajectory databases in the near future.

In addition to the GPS data, multi-player on-line games, like Quake 2, are a substantial source of movement data as they allow their users (players) to record their in-game trajectories (as well as other status and action data) and publish the data on the internet for analysis and behaviour studies. There has been some recent efforts [15, 44] in the Artificial Intelligence (A.I) research community to study the in-game trajectory data to distinguish human players and computer-controlled (bot) players. Moreover, following an incidence of a virtual outbreak, epidemiologists noticed the similarity between players' behaviour during the virtual outbreak and humans' behaviours during an actual epidemic outbreak. They went on to suggest the feasibility of using on-line games as test-beds for studying human behaviours — actions, communications, and movement data — to assess the effectiveness of methods to control communicable diseases [9, 41].

In this thesis, we will study the problem of *Mining Trajectory Databases for Multi-object Movement Patterns* (formally defined in Chapter 2). Knowledge of the instances of Multi-object Movement Patterns, which are embedded in the **T**rajectory **D**atabases (TJDB), such as (a) multiple objects travelling to and meeting in a specific spatial area — a meeting, (b) multiple objects travelling in the same route at different time — a frequent route, and (c) multiple objects forming and moving in a group — a convoy — will be interesting for various applications in epidemiology, ecology, preservation of wild-life, traffic monitoring and control, Location-Based Services (LBS), marketing, social-studies, and even on-line game development.

However, there are many limitations in the existing data mining and knowledge

discovery techniques to discover instances of Multi-object Movement Patterns. Current literature lacks an experimental studies on algorithms to discover the meeting patterns. Moreover, for each meeting pattern formed, the associated meeting place is not well defined yet. There are still challenges in discovering frequent routes without prior knowledge of the underlying spacial region as spatial-space is continuous. Existing works on finding convoy patterns cannot handle real-life convoys as, in reality, convoys members occasionally dispatches themselves from their parent groups as well as new members join and/or existing members leave the convoy in different stages of the convoys' life-spans. In addition, a Trajectory Database (TJDB) contains movement data of several thousands of objects over an extended period of time. Therefore, efficient and effective mining of TJDBs for the instances of the Multi-object Movement Patterns becomes a new and interesting challenge.

1.1 Motivation

In this section, we will briefly introduce the **Multi-object Movement Patterns** (MOMO Patterns), which we will explore in details in the following chapters, and motivate the study of extraction their instances (MOMO Instances) from Trajectory Databases (TJDBs).

1.1.1 Meetings

Informally, a meeting is formed when a group of objects comes to a *fixed* (circular) area and stays in the area for a while. Discovery of the meeting and related information — its member objects, place, time, and duration — from Trajectory Databases can have many applications. For instance, tracking the meeting place and group size of the tracked animals across time enables the ecologists to better understand grouping behaviours (interactions) of the animals they are tracking for

their researches as well as know the animals' habitats and grouping time.

For some applications, the information of the meeting places and time can be more important than their member objects. For example, meetings of commuters in a particular restaurant at lunch time show the restaurant is popular for lunch among commuters. Location-based Services can use this information to recommend popular restaurant to other users, who is looking for a good place to have lunch. In this example, the place and time the meeting instances appeared are more important than who participated in the meetings for the purpose of making recommendations.

However, the existing literature lacks a thorough experimental study on the discovery of meeting patterns from Trajectory Databases (TJDBs). To accurately report all meeting patterns from a TJDB, the only existing algorithm reported in [23] requires $O(n^4\tau^2)$ time (n is the number of objects and τ is the number of time-stamps in the TJDB) in order to report all longest duration meetings. It will not be scalable for TJDBs containing hundreds of objects that spans a long time-span. Therefore, the need to develop practical algorithms for extracting the information (members, place, time, *etc*) of the meetings in TJDBs is still open.

1.1.2 Frequent Routes

A frequent route is a path, which many of the tracked objects take frequently. The knowledge of frequent routes and their characteristics (for example, time-of-day) can be useful in many applications including traffic navigations and route suggestions for sight-seeing or travelling. Current traffic navigation systems (marketed as GPS devices with built-in navigations) use the shortest-paths in the road network to navigate their users to reach their destinations. This approach has several limitations since the shortest route is not necessarily the best route (in terms of time taken to travel if there is usually some traffic jams on that route). Moreover, the shortest path may not be suitable for the tourists (the recommended path may not

pass many sight-seeing locations) or even not safe to walk (the recommended path passes the areas having high crime rates). Knowledge of how to select the best route is often embedded in locals' trajectory data as frequent routes since the locals (cab drivers etc) learn which routes are the best routes from their experiences and take them frequently.

Mining frequent routes from a Trajectory Database (TJDB) is not trivial for many reasons. Firstly, in many applications, underlying road network (or semantic and properties of spatial-regions) is not available. For instance, pedestrians are not confined to road networks and will walk arbitrarily. Therefore, without a concrete information of all the underlying routes, it is not possible to count the number of time each route is used. Secondly, two vehicles travelling the same road or the same vehicle travelling the same road twice will rarely have two identical sequences of locations reported in the trajectory databases because the spatial space is continuous. Even if the movement is made on the exact same path (by two vehicles or same vehicle at different time), it is still not possible to directly match the sub-trajectories as the movements made may be at different speeds and, in the case of two vehicles, they may have different GPS sampling rates. Therefore, matching two sub-trajectories if they are taking the same route is not trivial and needs a complicate similarity metric. Lastly, a TJDB contains movement data of a large number of tracked objects over a lengthy period of time, resulting in a huge number of sub-trajectories to check. Given that the number of sub-trajectory routes in a given TJDB tends to be exponential in nature, an efficient traversing of the TJDB in order to discover frequent routes becomes an essential. Hence, efficient and accurate discovery of frequent routes in Trajectory Databases become a research area worth exploring.

1.1.3 Evolving Convoys

The existing works [10, 23, 30, 32] model a convoy — i.e. a group of tracked objects, which travel together — as a fixed set of member objects, which are found together throughout the life-span of the group. In reality, we notice that some real-life convoys have some members, which move away from the other members of the convoy (parent convoy) from time to time. For example, some animals may temporarily move away from their herds. It is also possible that a commuter from a convoy may leave behind due to the traffic congestion (due to the existence of pedestrians on zebra crossing, traffic lights *etc*) or the need for petrol (driving away from the convoy to a petrol station) and catch up the convoy shortly after. When a car-pooling recommendation system makes suggestions for suitable car-pooling groups using convoy information, it is not desirable for the recommendation system to leave him out just because he was temporarily away (left behind) from other commuters, who were travelling in the same route at the same time as he was. In on-line games, some players belonging to a group may move away from their peers to complete some tasks (quests). There is a need to model the real-life convoys in a more natural and flexible way, which allows some members of the convoy to temporarily move away from the convoy.

Moreover, in reality, some members may join (leave) the convoy later (earlier) than the convoy's starting (ending) time. For car-pooling recommendation systems, it is more practical and desirable to include a commuter in the car-pooling group suggested for the members of convoy that he had always joined although he was never present when that convoy started to form. Results obtained from mining Trajectory Databases using the current convoy models contain several convoys, whose member objects and life-spans overlap, when there is a tracked object joining (leaving) the convoy. From usability point-of-view, reporting all such overlapping

convoys may be confusing and have limited applications. For monitoring wild-life, a complete list of overlapping convoys is hard to comprehend for the human scientist (and may be subjected to more processing in order to establish links between related convoys). Selecting a single representative from a set of overlapping convoys is also an application-dependent task. For example, some scientists may be interested in longer-duration convoys (with fewer members) while others may be more interested in larger convoys (with shorter life-spans). A more realistic approach that reports each related set of overlapping convoys as a more comprehensible single evolving entity is needed.

An interesting new application of near real-time convoy information is in the development of Mass Multi-player On-line Games (MMOs). MMOs are on-line games which allow players, whose characters are in close proximity of each other in the game world, to interact with (communicate and help) each other. Since this feature distinguish MMOs from traditional single-player computer games, the application providers (game developers) allow and even encourage the players to form groups.

Since the players reside in (and share) the same virtual world and kill the same set of enemies (called “monsters”), the game needs to constantly replenish the virtual world with new monsters for the players to kill. Replenishing the virtual world with monsters is termed as “spawning”. Currently, the monsters are spawned based on the region of the virtual world using a static script created by the developers. Since monsters are spawned in a region regardless of the characteristics of the player groups in it, for larger groups, the game will be easy while for smaller groups, the same game will be difficult. The top two panels of the Fig. 1.2 shows a demonstration of the limitation of spawning monsters using a static script. The application server (game server) created five hard monsters regardless of the size of the group of players. For a group of three players (top-left panel), the game will be difficult

but for a group of eight players (top-right panel), it will be easy.

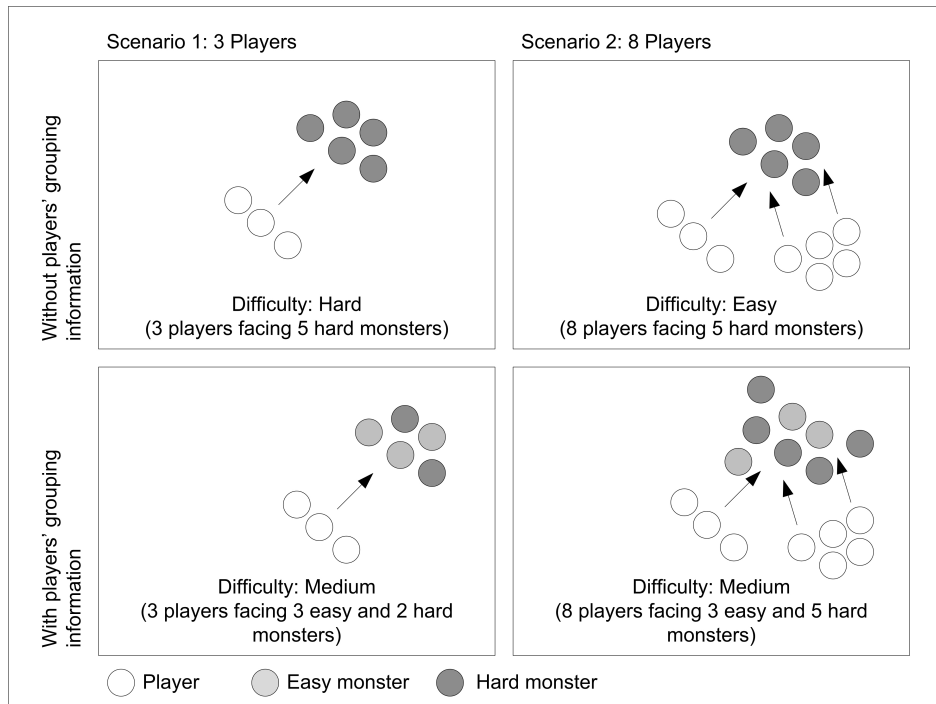


Figure 1.2: How Convoy Information Improves Players' Experience.

Ad hoc creation of monsters and puzzles based on the players' statuses by an Artificial Intelligence agent has been explored for single-player games [36] and demonstrated for a limited (up to 4 players) multi-player game [11]. To extend the ad hoc monster creation into MMOs, the application server needs the near real-time convoy information (group size and skills of the members) of the players. With convoy information of the players extracted from the movement data-streams of the players, the application server (game server) can uniquely spawn monsters and puzzles for each user group. The bottom two panels of the Fig. 1.2 demonstrates how the game server can create suitable monsters based on the grouping information of the players. For fewer players, fewer monsters are spawned (bottom-left panel) while more monsters are spawned for a larger set of players (bottom-right panel).

1.2 Contributions

The contributions of this thesis can be divided into three parts. The first two parts deal with reporting the Multi-object Movement Patterns (MOMO Patterns) from off-line Trajectory Databases (TJDBs) while the last part deals with finding MOMO Patterns in both off-line and streaming settings.

1.2.1 Meetings of Moving Objects

This thesis presents the problem of mining Trajectory Databases (TJDBs) for meeting patterns along with a new definition of meetings, called **Meeting of Moving Objects** (MEMO), which defines the information associated with each instance of the meeting pattern such as the meeting place, duration, and members. We also designed effective and efficient algorithms to find meeting patterns in a TJDB and report the associated meeting places, durations, and members.

We implemented (a discrete version of) the existing algorithm proposed in [23] to discover meetings and compared it with our solutions. According to the experimental evaluations we conducted, our methods to find MEMOs are more accurate and efficient than the existing solution.

1.2.2 Sub-trajectory Cliques and Frequent Routes

This thesis contains our studies on finding frequent routes from a Trajectory Database (TJDB). Since a road network or semantic of the regions of the spatial space the moving objects are traversing is often not available — for example, ecologists studying some wild animals do not have a complete roadmap of the routes the animals are using, we developed methods to discover frequent routes from a given TJDB without the need of prior knowledge of the underlying spatial space. We explored the option of grouping similar sub-trajectories together and extracting a frequent route

from each group as this two-step method does not require to have the underlying road networks that the moving entities in question take.

In order to group similar sub-trajectories, i.e. sub-trajectories taking the same route, together in the same group regardless of the speed they travelled, minor differences in sequence of locations they reported in the TJDB, and differences in GPS sampling rate, we used Fréchet distance as the similarity measure in grouping sub-trajectories.

However, since mining **Sub-trajectory Cliques** (TRAJCLIQS) using Fréchet distance — also known as sub-trajectory clustering — is a known NP-Complete problem [12], we designed novel data-driven approximation algorithms, which are able to efficiently discover approximate TRAJCLIQS and frequent routes from real-life datasets.

1.2.3 Dynamic Convoys and Evolving Convoys

As the final contribution, this thesis reports our exploration in the area of convoy discovery. Since we realize the traditional notion of convoys cannot accurately model the real-life convoys, which has dynamic members — or the members of a convoy moving away from the convoy temporarily, we introduced a new concept of convoys called **Dynamic Convoys** (DYCO). A DYCO allows dynamic members under constraints imposed by user-defined parameters.

Since real-life convoys may also have new members joining the convoy and existing members leaving the convoy (they may not return at all), we continued to study the new concept of convoy evolution by defining how DYCOs (of fixed duration) evolves into one another. An **Evolving Convoy** (EVOCO) captures the relationships between different stages of convoys such that a convoy in a stage has more (fewer) members than its previous stage.

We explored new algorithms that can be used to incrementally discover evolving

convoys. The proposed algorithms are designed to be incremental in nature so that we can use them for Trajectory Databases, which are streaming into the mining process in real-time.

1.3 Organization

This thesis is organized in the following manner. The current chapter introduces the subject of the thesis. We will give an overview of the thesis and the related works in the next two chapters, which will be followed by three more chapters, each devoted to our contributions to the mining a specific Multi-object Movement Pattern. Then, we will conclude the thesis.

In Chapter 2, we will formally introduces the concept of *Mining Trajectory Databases for Multi-object Movement Patterns* and provide an overview of the specific mining problems we are going to present in this thesis. We will also introduce the platform (data and computation settings) we used for the experiments we conducted.

In Chapter 3, we will discuss the related works to this thesis. We will present and discuss in details of the existing literature on general data-mining techniques and finding different types of multi-object movement patterns in a Trajectory Database.

We devote Chapter 4, 5, and 6 for mining Multi-object Movement Patterns from Trajectory Databases. We will describe our research on algorithms to find instances of the Meeting of Moving Objects (MEMO) in Chapter 4. In Chapter 5, we will propose Sub-trajectory Cliques (TRAJCLIQS), from each of which we will extract a Frequent Route. We will discuss approximation algorithms to mine TRAJCLIQS in a Trajectory Database to find frequent routes. In Chapter 6, we will present new concepts concerning convoys, namely the concept of dynamic convoy (DYCO) and the concept of how a sequence of DYCOs evolving into one another

to form an evolving convoy (EVOCO), and discuss algorithms to extract EVOCOs incrementally from (both streaming and off-line) Trajectory Databases.

We will conclude this thesis in Chapter 7.

Some of the research works described in this thesis have been published. The works in Chapter 4 and Chapter 6 are published as research papers [6, 7] in the Proceedings of the 23rd and 22nd Scientific and Statistical Database Management Conferences (SSDBM 2011 and SSDBM 2010) respectively. An abridged version of this thesis [8] appeared in the ACM SIGSPATIAL Special, Volume 4. The work in Chapter 5 is going to appear in the Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD 2013).

Overview

In this chapter, we will formally introduce the concept of mining Trajectory Databases (TJDBs) for Multi-object Movement Patterns and give an overview of the pattern-mining problems we are going to explore in the proposed thesis. We will also discuss the platform, i.e. data and settings, we use in this thesis in order to assess the performance of our proposed mining techniques.

2.1 Mining Trajectory Databases for Multi-object Movement Patterns

Definition. 2.1. Trajectory Database — For a given set of objects $O = \{o_1, o_2, \dots, o_n\}$, time-stamps $T = \{t_1, t_2, \dots, t_\tau\}$, and a spatial-space \mathbb{R}^d , a Trajectory Database \mathcal{R} is a set of records of the form $\langle o, t, loc \rangle$ where $o \in O, t \in T$ and $loc \in \mathbb{R}^d$.

In a Trajectory Database (TJDB), o and t form a composite key that uniquely determines loc . However, a given TJDB can be incomplete – i.e. for all $\{o, t\} \in O \times T$, there may not be $\langle o, t, loc \rangle \in \mathcal{R}$ — since, in reality, some objects may be untraceable in certain time-stamps – i.e. the locations of some objects may not be known for some time-stamps due to hardware limitations. Although time is assumed as a discrete sequence with equal intervals between each consecutive points, generality of Def. 2.1 is not undermined since any application can set an

arbitrarily small interval.

We will define some preliminaries before we move on to define Multi-object Movement Patterns and Mining Trajectory Databases for them.

Definition. 2.2. *Collective Movement* — *A collective-movement X is a set of movement records found in a Trajectory Database \mathcal{R} , i.e. $X \subseteq \mathcal{R}$.*

Definition. 2.3. *Member Objects of a Collective Movement* — *The set of member objects $O(X)$ of a collective-movement X is the set of all objects, whose movement data is included in X , i.e.*

$$O(X) = \{o : \langle o, t, loc \rangle \in X\}.$$

Definition 2.2 defines a collective movement as a description of a set of movements some objects made as found in a Trajectory Database (TJDB). Definition 2.3 defines the member objects, who perform a given collective-movement. For example, in the Trajectory Database \mathcal{R} visualized in Fig. 2.1, $X_1 = \{\langle o, t, loc \rangle : o \in \{a, b, c\}\}$, $X_2 = \{\langle o, t, loc \rangle : o \in \{d, e\}\}$, and $X_3 = \{\langle o, t, loc \rangle : o \in \{f, g, h\}\}$ are some collective-movements found in \mathcal{R} , which describe the movements of their respective sets of member objects, $O(X_1) = \{a, b, c\}$, $O(X_2) = \{d, e\}$, and $O(X_3) = \{f, g, h\}$.

Definition. 2.4. *Collective-movement Predicate* — *A collective-movement predicate is a mapping $q(X)$, which maps a collective-movement $X \subseteq \mathcal{R}$ to either true or false, i.e.*

$$q : \mathcal{P}(\mathcal{R}) \rightarrow \{true, false\}.$$

In Def. 2.4, a collective-movement predicate is defined as a boolean function that determine the movements described in a given collective-movement meets the criteria specified in the predicate. For instance, suppose some predicates are defined as follow:

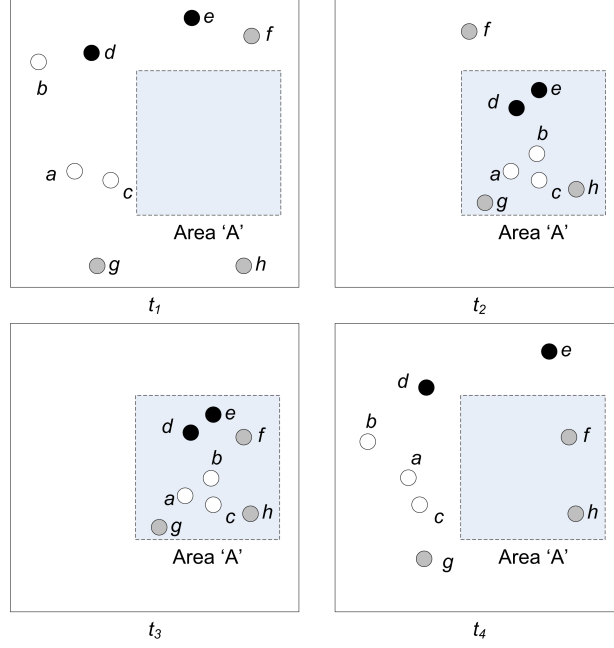


Figure 2.1: An Example Trajectory Database Containing Movement Records of Eight Objects for Four Time-stamps.

- $q_1 = true$ iff $O(X)$ contains (exactly) three objects;
- $q_2 = true$ iff all *loc* of all member objects at t_1 is outside Area 'A';
- $q_3 = true$ iff all *loc* of all member objects at $[t_2, t_3]$ is inside Area 'A';
- $q_4 = true$ iff all *loc* of all member objects at t_4 is outside Area 'A';

For each predicate defined above, we can check whether a collective-movement meets the predicate. Table 2.1 shows whether each of the collective-movements X_1, X_2 , and X_3 (described above) meets the criteria defined in each predicate (*true* means meeting the criteria). For example, the collective-movement X_1 meets the predicate q_1 — $q_1(X_1) = true$ — since it describes the movements of three objects $\{a, b, c\}$ while X_2 does not meet q_1 — $q_1(X_2) = false$ — because X_2 has only two member objects. Likewise, we can see X_2 meets predicate q_3 because all its member objects, d and e , stayed in the Area 'A' during $[t_2, t_3]$ while X_3 does not meets q_3 since one of its member objects, f , was not in the Area 'A' at t_2 .

Table 2.1: Example Predicates and Collective Movements.

	Collective-Movement X_1	Collective-Movement X_2	Collective-Movement X_3
Predicate q_1	<i>true</i>	<i>false</i>	<i>true</i>
Predicate q_2	<i>true</i>	<i>true</i>	<i>true</i>
Predicate q_3	<i>true</i>	<i>true</i>	<i>false</i>
Predicate q_4	<i>true</i>	<i>true</i>	<i>false</i>
Instance of Movement-Pattern Q	yes	no	no

Definition. 2.5. Multi-object Movement Pattern – A Multi-object Movement Pattern Q is a set of collective-movement predicates, i.e. $Q = \{q_1, q_2, \dots, q_p\}$.

Definition 2.5 defines a Multi-object Movement Pattern (MOMO Pattern) as a set of collective-movement predicates, which describes the characteristics (criteria) of the MOMO Pattern. For example, a MOMO Pattern, “three commuters lunch movement pattern” can be defined as “three commuters enter the restaurant (Area ‘A’) at t_2 , have lunch and leave the restaurant at t_4 ”. This movement pattern has four criteria, (a) there must be three objects, (b) these objects must be outside Area ‘A’ before t_2 , (c) these objects must be inside Area ‘A’ during $[t_2, t_3]$, and (d) these objects must be outside Area ‘A’ again at t_4 . This Movement Pattern, therefore, will be defined as $Q = \{q_1, q_2, q_3, q_4\}$, where q_1, q_2, q_3 , and q_4 as defined above.

Definition. 2.6. Instance of a Multi-object Movement Pattern – Given a Trajectory Database \mathcal{R} , a collective-movement X is an instance of the Multi-object Movement Pattern Q , or simply “ $O(X)$ forms Q ” (as evidence by X), if and only if X meets all collective-movement predicates in Q , i.e.

$$X \in N(Q, \mathcal{R}) \iff \bigwedge_{q \in Q} q(X) = \text{true},$$

where $N(Q, \mathcal{R}) =$ the set of all instances of Multi-object Movement Pattern Q found in \mathcal{R} .

Following Def. 2.6, the member objects of a collective-movement is said to

form a Multi-object Movement Pattern if the collective-movement meets all the criteria (the collective-movement predicates), which describe the characteristics of the said Movement Pattern. For example, for the Trajectory Database depicted in Fig. 2.1, the tracked objects $\{a, b, c\}$ forms the three commuters' lunch movement pattern $Q = \{q_1, q_2, q_3, q_4\}$ — or the collective-movement X_1 is an instance of Q — as X_1 meets all four predicates in Q . In this report, we will propose a thesis on finding instances of Multi-object Movement Patterns (MOMO Instances). The knowledge of the instances of such movement patterns formed by the tracked objects is embedded and hidden in the data archived in the Trajectory Databases.

Definition. 2.7. Mining Trajectory Databases for Multi-object Movement Patterns – Mining Trajectory Databases for a given Multi-object Movement Pattern Q is a process $\mathcal{M}_Q(\mathcal{R})$ that takes a Trajectory Database \mathcal{R} as its input and outputs the information of all instances of Q found in \mathcal{R} .

Following Def. 2.7, the process of Mining Trajectory Databases (TJDB) to look for a pre-defined Multi-object Movement Pattern (MOMO Pattern) takes a TJDB and reports all instances of the Multi-object Movement Pattern (MOMO Instances) found in the TJDB. Figure 2.2 depicts the concept of mining Trajectory Databases for a MOMO Pattern.

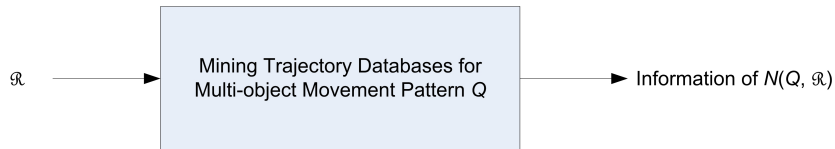


Figure 2.2: Mining Trajectory Databases for Multi-object Movement Patterns.

2.2 Proposed Mining Problems

2.2.1 Finding Closed Meetings of Moving Objects

The proposed concept of mining a Trajectory Database (TJDB) for **Meetings of Moving Objects (MEMOs)** is depicted in Fig. 2.3. The definition of MEMO (embedded in the mining process) will allow users to customize the meeting pattern by specifying minimum number of members, minimum meeting duration, and maximum spatial size of meeting place in order to make the process of mining TJDB for MEMOs report instances of the customized meeting pattern. Given a TJDB and MEMO parameters, the mining process will produce information of all instances of closed MEMOs found in the given TJDB as output.

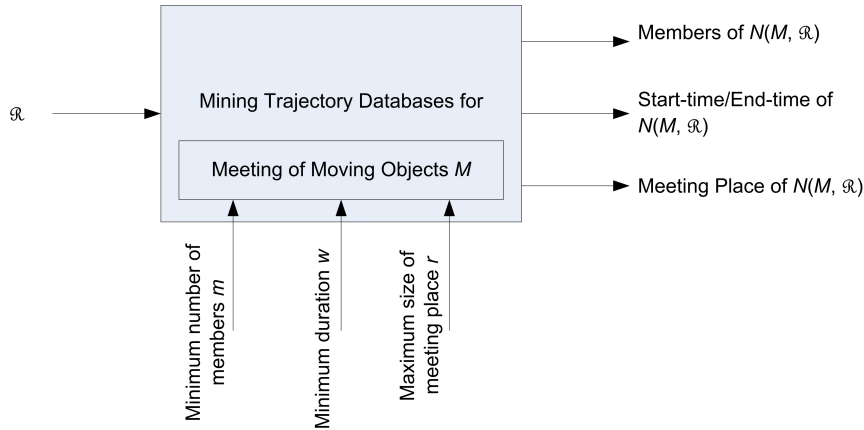


Figure 2.3: Mining Trajectory Databases for Closed Meetings of Moving Objects.

2.2.2 Mining Sub-trajectory Cliques to Extract Frequent Routes

An overview of the process we are going to develop to find Frequent Routes in Trajectory Database (TJDB) is demonstrated in Fig. 2.4. The process is a two-step process involving (a) the first step that finds Sub-**trajectory cliques (TRAJCLIQS)** and (b) the second step that infers a frequent route for each TRAJCLIQ. The definition of TRAJCLIQS is customizable by the users by supplying parameters for clique mining,

namely minimum length (not time duration) of the sub-trajectories in a TRAJCLIQ must have, maximum distance between the sub-trajectories in a TRAJCLIQ, and the minimum number of sub-trajectories in a TRAJCLIQ. The mining process will extract the information of all the frequent routes from the TRAJCLIQs and report their information.

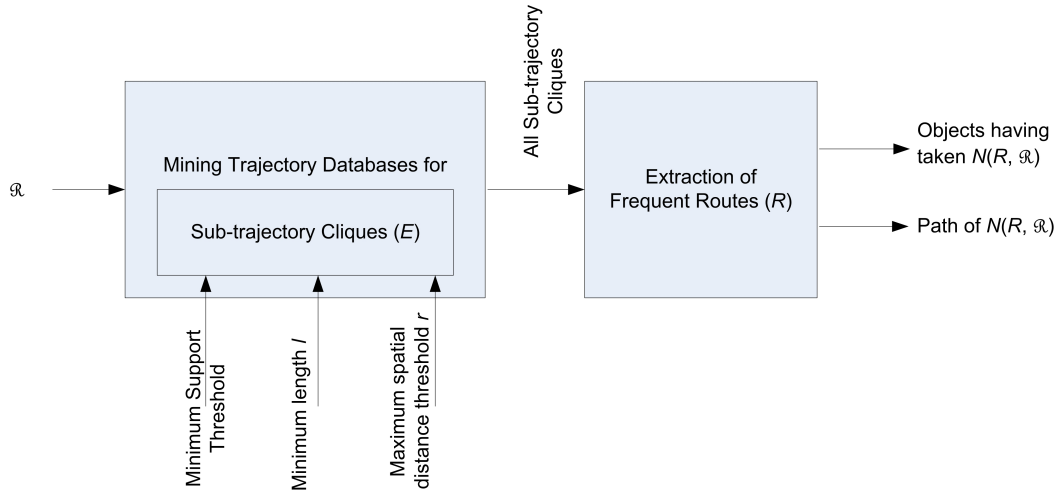


Figure 2.4: Mining Trajectory Databases for Sub-trajectory Cliques to Extract Frequent Routes.

2.2.3 Discovery of Evolving Convoys

The specifications of the algorithms that mine a Trajectory Database (TJDB) for **Evolving Convoys (EVOCO)** are depicted in Fig. 2.5. The definition of EVOCO pattern can be customized through its parameters, minimum number of members, minimum duration, and dynamic-member/non-member threshold (this parameter is to differentiate a dynamic-member from a noise object). The mining process reports stages of all evolving convoy instances in a given TJDB.

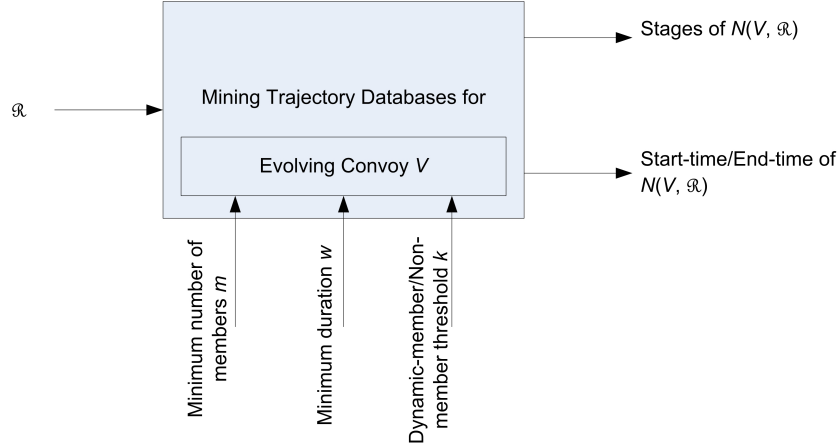


Figure 2.5: Mining Trajectory Databases for Evolving Convoys.

2.3 Platform to Assess the Proposed Algorithms

2.3.1 Datasets and Data Cleaning

We will use five human movement datasets and two taxi movement datasets to assess our proposed algorithms designed to extract the instances of the Multi-object Movement Patterns (defined above). Five human movement datasets are Statefair, Orlando, New York, NCSU and KAIST obtained from [28]. New York consists of traces of the volunteers commuting by subways, by buses and on foot, while NCSU and KAIST consist of traces of students on campuses. Two taxi movement datasets we will use are SF-Cab21 and SF-Cab22 consisting of taxi movement extracted from [45]. SF-Cab21 (SF-Cab22) consists of taxi movement from 8AM to 4PM in San Francisco Bay Area on 21-Apr-08 (22-Apr-08). We also derive subsets of the taxi datasets, SF-Cab21rand100 and SF-Cab22rand100, which consists of movement of 100 random taxis during the aforementioned time-frames.

In addition, we will also use Trucks [1] consisting of trajectories of 50 trucks moving in Athens and Ships consisting of trajectories of 458 ships. The Ships dataset is obtained from AIS transmissions received from the ships moving in Singapore waters during September 5, 2011 from 0800 - 1200 hrs. A summary of the datasets is given in Table 2.2.

Table 2.2: A Summary of the Datasets Used to Assess the Proposed Algorithms.

Name	Object Count	Covers	No. of Records
Statefair	19	3.5 hour	5,861
NCSU	35	21.7 hour	42,829
New York	39	22.7 hour	118,584
Orlando	41	14.3 hour	133,076
KAIST	92	23.4 hour	404,981
SF-Cab21	482	8 hour	159,107
SF-Cab22	477	8 hour	182,911
Trucks	50	40 days	112,203
Ships	458	4 hour	149,660

The five human movement datasets are published after a process of location-privacy preservation filtering, i.e. they use planar coordinate systems with arbitrary reference origins to prevent inferring the actual locations each tracked objects visited. In contrast, other datasets use well known coordinate systems. Trucks dataset uses a planar coordinate system called GGRS87/Greek Grid (EPSG:2100). The spatial distance unit used for human movement datasets and Trucks dataset is in metre. The two taxi movement datasets, SF-Cabs21 and SF-Cabs22, and Ships movement dataset use spherical coordinate system called WGS84 (EPSG:4326) or the longitude/latitude system. Therefore, we are able to infer the physical locations the tracked objects visited in these datasets.

Since planar coordinate systems make it easier to compute spatial distance, we project the datasets in spherical coordinate system, namely SF-Cab21, SF-Cab22, and Ships datasets, to planar coordinate systems, which use metre as their distance unit. We project taxi datasets (SF-Cab21 and SF-Cab22) and Ships dataset into NAD83(HARN)/California zone 3 (EPSG:2768) and SVY21/Singapore TM (EPSG:3414) respectively. Now, the distance unit in all datasets is in metre.

The datasets, SF-Cab21, SF-Cab22, and Ships, consist of a few erroneous location measurements, i.e. the reported locations for certain time-stamps are inaccurate. For instance, some of the ships reported their location at $\langle 0, 0 \rangle$ in WGS84

coordinate system, which is near the west coast of Africa, although they are in Singapore waters making short-range radio contact with AIS receiver system located in the National University of Singapore (NUS). We remove such records by removing the records with location reported outside the projected coordinate systems' bound.

We also notice SF-Cab21, SF-Cab22, Ships, and Trucks contain missing chunks in some trajectories, i.e. a portion of the trajectory is not reported in the dataset at all, which is characterized by a huge distance between consecutive reports. We learn the distribution of the distances between locations of two consecutive records (of the same tracked object) in order to determine the ideal threshold value to identify such gaps. Figure 2.6 shows such distribution of dataset, SF-Cab21. Using the threshold values of 1.4km, 1km, and 0.5km for taxi datasets, Ship dataset, and Trucks dataset respectively, we remove all missing portions in the trajectories in these datasets and mark the records before and after the gap as the last point and the first point of two separate trajectories. Figure 2.7 shows a comparison of some portions of SFCab21rand100 before and after the cleaning process. We can clearly see some (taxi) trajectories travelling through the water in the East are removed in the cleaned dataset.

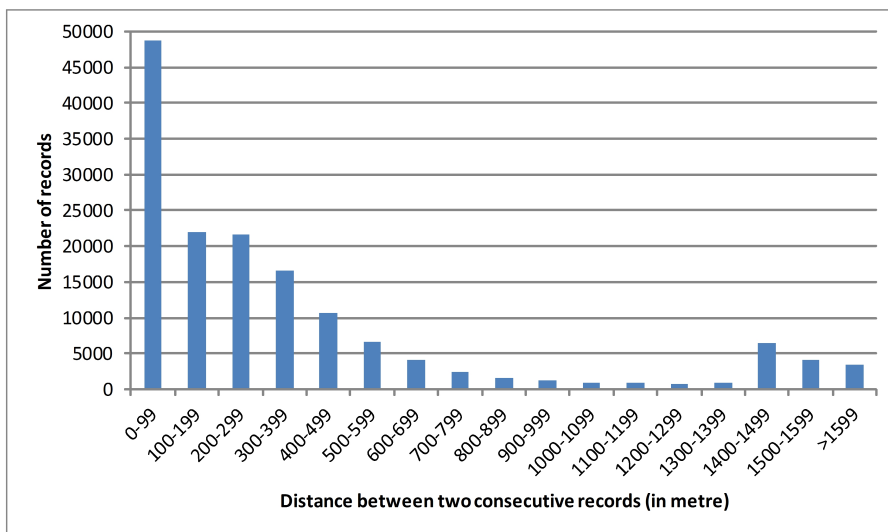


Figure 2.6: Distribution of the Distances between Two Locations Consecutively Reported by the Same Tracked Object in SF-Cab21.

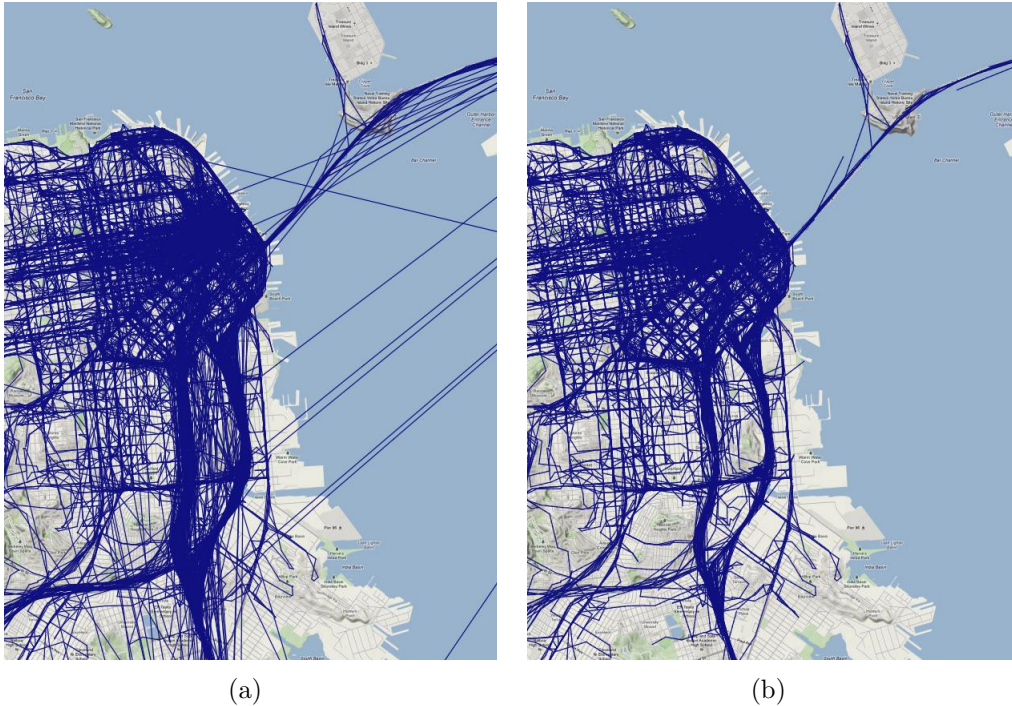


Figure 2.7: Comparison of a Portion of the Taxi Dataset, SF-Cab21rand100, (a) before Cleaning and (b) after Cleaning Using a Threshold of 1.4km.

2.3.2 Computational Environment

All datasets are stored in $\langle pkey, objid, trajid, ts, loc \rangle$ format in a PostgreSQL 8.4 with $pkey$ set as the primary key (int8), while $objid$, $trajid$, ts , and loc stands for object-id (varchar), trajectory-id (int8), time-stamp (int8), and location (PostGIS 2.0 Point). Some of the $objid$ has multiple corresponding $trajid$ due to our data cleaning process, which divides a trajectory containing an anomalous gap into two trajectories. The data is clustered together by $objid$. The time-stamp ts is stored in Unix Epoch format while the granularity of the interval between two consecutive time-stamps is 10 second for meeting and convoy experiments. We have a B-tree index on $\langle trajid, ts \rangle$.

We will implement all the proposed algorithms and existing (base-line) algorithms in Java. Performance studies for the meeting pattern mining algorithms are to be conducted on a server equipped with Intel Xeon X5365 CPU running

at 3.00GHz and 16GB of RAM, while the rest of the experiments (including the trajectory clique pattern mining and evolving convoy pattern mining) are to be conducted on a server equipped with Intel Xeon CPU E5607 running at 2.27GHz and 32GB of RAM. During the experiments, we will have the amount of memory available to the Java Virtual Machine capped at at 8GB and 16GB on the servers respectively. Both of the servers are running a Linux Distro.

Related Works

In this chapter, we will discuss the existing works, which is related to Mining Trajectory Databases (TJDBs) for various Multi-object Movement Patterns (MOMO Patterns). We will start with the general data-mining techniques including those traverse power-sets to find association rules and those for clustering spatial data. Then we will discuss more closely related works to find movement patterns in TJDBs.

3.1 General Data-mining Techniques

3.1.1 Traversing Power-sets

For a given set S , its power-set $\mathcal{P}(S)$ is defined as the set of all its subsets, i.e. $\mathcal{P}(S) = \{V | V \subseteq S\}$. A set S is said to have the *apriori*-properties with respect to a predicate p if and only if the following statement is true: if $V \subseteq S$ fulfils p , then its subsets $V' \subseteq V$ must fulfil p too. The *Apriori* algorithm, the first data-driven algorithm to traverse the power set $\mathcal{P}(S)$ of a given set S having the *apriori*-properties (with respect to whether the number of occurrence of a set is at least the given support threshold) appears in [3]. It traverses the search space, starting with all the interesting sets with exactly one member each, building up interesting sets containing $(k + 1)$ members from those containing k members.

Although the *Apriori* algorithm designed to perform association rule mining is fast enough, it requires a large amount of memory. Therefore, Zaki [58] proposed

Equivalence CLASS Transformation (ECLAT). In ECLAT-based data mining algorithms, the power set $\mathcal{P}(S)$ of a given set $S = \{s_1, s_2, s_3, \dots, s_n\}$ is divided into n equivalent classes $C_1, C_2, C_3, \dots, C_n$. Using an arbitrary order \preceq on S , the k^{th} equivalent class C_k is defined as $C_k = \{V | s_k \in V \text{ and if } s_i \in V \text{ then } s_k \preceq s_i\}$. Each equivalent class C , which is a sub-lattice and whose elements follow the *apriori*-properties, is recursively divided into sub-classes until each of the resulting classes fits entirely into the memory for processing by the *Apriori*-algorithm. It limits the memory requirement of frequent-itemset-mining at the expense of some redundant processing. FP-growth, the depth-first-search approach to frequent item-set mining, is proposed in [25].

3.1.2 Clustering of Data

Clustering of spatial-points is to be used as basic operations in mining Trajectory Databases for some Multi-object Movement Patterns (such as convoy patterns). Existing works on spatial clustering consist of hierarchical [24, 34] and partitioning [4] algorithms but they need domain-specific parameters (number of clusters or inter-cluster distance) in advance. These parameters are hard to pre-determine in the applications like mining convoy patterns, where grouping and movement behaviour of the objects should not be assumed. Ng and Han [43] proposed an efficient partition algorithm CLARANS and suggested running it multiple times to determine the best number of targeted clusters. When clustering of points for each time-stamp is required, this may be expensive.

Methods for clustering of point-objects in a spatial network are presented in [57]. Similar tasks dealing with moving objects (like vehicles on the road network) are handled by CMON framework [14]. CMON framework is capable of clustering in distance-based, density-based, and k-partition fashions.

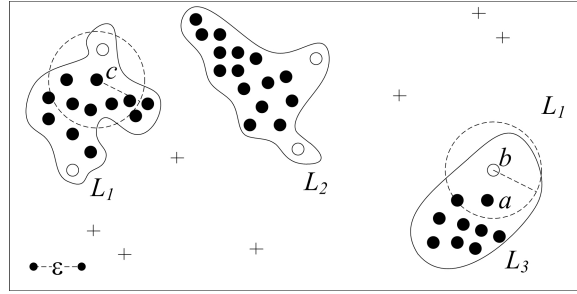


Figure 3.1: An Example of Density-based Clustering.

DBSCAN

Ester et al. [21] suggested density-based clustering, DBSCAN, which does not need any domain-specific parameters and is scalable. DBSCAN distinguishes each object in a density-connected clusters into two categories: core and border. A core object has at least min_pts objects within its ϵ -proximity and is used to expand the clusters. An object, which has less than min_pts objects within its ϵ -proximity and has a core object as its ϵ -neighbors is a border objects. Other objects are identified as noise objects, which do not belong to any cluster. For example, in Fig. 3.1, (for $min_pts = 3$) black circles like c are core objects while white circles like b are border objects (plus signs are noise objects not belonging to any cluster). In DBSCAN, only the maximal clusters are reported. DBSCAN is able to handle clusters of arbitrary spatial-shape and is tolerance to noise.

Insertion and deletions of data points may void the current clustering results in dynamic databases. Dynamic clustering, to cope with such insertions and deletions, is done with incremental DBSCAN [20]. GDBSCAN [49] is generalization of DBSCAN, which allows spatially extended objects (not points) to be clustered with an arbitrary neighbourhood predicate and neighbourhood cardinality.

3.2 Multi-object Movement Patterns

In this section, we will present the existing models (definitions) of Multi-object Movement patterns (MOMO Patterns) and finding them in Trajectory Databases (TJDB). In order to assist comparison between the models, we will name some variables differently from the original articles they first appeared. A similar detailed survey is also available in [31].

3.2.1 Meetings

One of the early works on extracting Multi-object Movement Patterns from a Trajectory Database (TJDB) is the study of *Meeting* patterns found in [23]. A (fixed) meeting is defined as a set M of m or more objects, which are within a *fixed single* circle of radius $r > 0$ in w or more consecutive time-stamps, where r , m , and w are given parameters.

Gudmundsson and Kreveld [23] reported that for a given TJDB containing movement records of n objects and τ time-stamps, the complexity to compute longest-duration meeting pattern(s) from the TJDB is $O(n^4\tau^2\log(n) + n^2\tau^2)$. Their algorithm is designed to report the longest-duration meeting(s) and never reports any meeting M , a subset of whose members form a longer duration meeting M' . For instance, the movement of five objects in Fig. 3.2, which depicts a scenario where two friends d and e arrive late to and leave early from a lunch appointment, it is clear that (for $w = 2$ and $m = 2$), there are two meetings – $M' = \{a, b, c\}$ forms a meeting during $[t_1, t_4]$ and $M = \{a, b, c, d, e\}$ forms a meeting during $[t_2, t_3]$. In this scenario, their algorithms will not report the larger meeting M .

Their algorithm is based on the observation that the meeting place and life-span of a meeting instance form the base and the height of a three-dimension (two space and one time) cylinder, in which the trajectory segments representing the movement

made by the meeting members during the meeting time-span. Since a cylinder can be defined by three points, their algorithm involves rotating a cylinder around each object trajectory across space and time, trying to fix two other points. Since such rotation is performed for each tracked object in turn, it ends up detecting a single meeting multiple times. Following their own definition, their algorithm focuses only on the objects that meet but does not calculate the smallest enclosing circle (meeting place) while it traverses the search-space, i.e. it requires a post-processing step if the user wants to know the precise (tight circular) meeting place that the tracked objects meet. They did not provide any experimental evaluations for their algorithms.

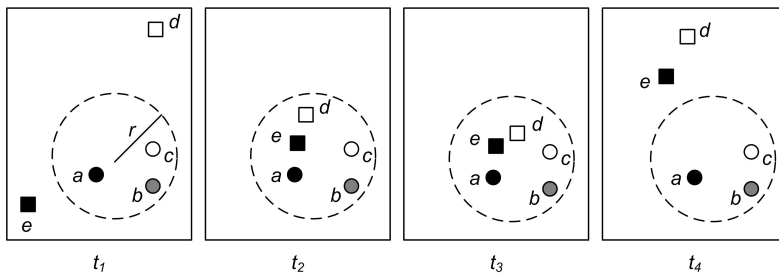


Figure 3.2: An Instance of Two Overlapping Meetings.

Minimum covering circles

Given a Trajectory Database \mathcal{R} , a distance r , a set of objects M containing at least m objects, and a time-interval I spanning at least w time-stamps, whether M forms a meeting during I for parameters m , w , and r can be verified by checking if there is a circle loc , which covers the set of points $P = \{p | \langle o, t, p \rangle \in \mathcal{R}, o \in M, \text{ and } t \in I\}$ and whose radius is smaller than r . The earliest algorithm to calculate the smallest circle enclosing a finite set of points is found in the translated text [46]. It starts with an arbitrarily large circle and shrinks (and move) the circle until no more shrinking is possible. Since it is difficult to implement on computers, Elzinga and Hearn [19] proposed a new algorithm called Euclidean Messenger Boy algorithm

(EMB) that monotonously increases the radius until all points are covered.

The radius r of the circle C covering a set of points P can be shown to be bounded by the inequality $r \leq \frac{1}{\sqrt{3}}md(P)$, where $md(P) = \max(dist(p, q))$ for all $p, q \in P$ [16, 46]. It is also known that the smallest enclosing circle $C(P)$ always exists and is unique for a set of points P [16].

3.2.2 Flocks

A closely related Multi-object Movement Pattern to the meeting pattern is the (fixed) *Flock* pattern, in which the circle covering the locations of member objects is not fixed but free to move through the entire life-span of the flock pattern. In other words, instead of a single fixed circle, the locations of all member objects, which form a flock pattern, is covered by a *different* circle of radius r in each time-stamp during the flock's life-span. Figure 3.3 illustrates an instance of a meeting and that of a flock. The instance of the flock has three circles, which cover its member objects in three time-stamps, while that of a meeting has a single (stationary) circle covering its members.

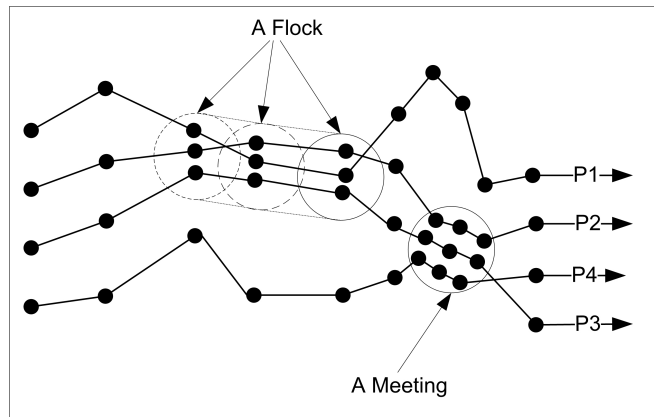


Figure 3.3: An Example Contrasting a Flock and a Meeting ($k = 3$ and $m = 3$).

Gudmundsson and Kreveld [23] reported that given a Trajectory Database (TJDB) containing the movement records of a set of n objects for a time-span of τ (consecutive) time-stamps, the complexity to compute and report the longest

flock pattern(s) formed in the TJDB is NP-Hard and they gave complexities for various approximations. Benkert et al. [10] described a method that transforms d dimensional trajectories (containing τ points) into $d\tau$ dimensional points and performs range-query to find flocks. Their work also provided three approximations for their method. It is apparent that their methods cannot be extended when the TJDB contains movement records of a long time-span.

Vieira *et al.* [53] reported polynomial-time algorithms to find flocks of fixed durations $dur = w$. Their method is based on a theorem that says if a set of points L can fit in a circular disk, they can also fit in another disk of the same size, on whose boundary two of the points $p, q \in L$ lie.

3.2.3 Moving Groups

Hwang et al. [26] described a definition of moving groups. In order to form a moving group pattern, each member of a set of objects G must be within min_dis away from the others for min_dur or more consecutive time-stamps. In other words, throughout the group's life-span, its members must form a clique in each time-stamp. They proposed an algorithm based on the *Apriori*-algorithm, which uses the *apriori*-properties, and VG-Growth algorithm, which is based on FP-growth algorithm, to find all such groups of moving objects. These algorithms are extended to find maximal groups in [54].

3.2.4 Convoys

In order to form a flock (or a moving group), the member objects must stay within a circle not larger than the user-defined size (or be within a user-defined distance to all other members). In other words, the size of the spatial region a flock or a moving group can cover is limited by the user-defined parameter r (min_dis). As a consequence, the number of members each flock (group) can have is limited as a

typical real-world object such as a human, a vehicle *etc* has a volume and more than a certain number of objects cannot be fit into the circular area of a pre-defined size (cannot maintain intra-group distance). In other words, flocks (and moving groups) may not be able to model groups of moving objects for certain applications well and the algorithms designed to find flocks (and moving groups) may not be able to find groups of moving objects for such applications.

Jeung et al [30,32] termed the situation described above as lossy-flock problem and defined a convoy as a group of at least m objects being density-connected with each other throughout w consecutive time-points, where m , w , and DBSCAN parameter ϵ are provided by the user (m also doubles as DBSCAN parameter *min_pts*). Since a convoy can occupy a spatial region of arbitrary size and shape in its lifetime, there is no limit to the maximum number of objects, which can form a convoy. They proposed a filter-and-refinement scheme called CuTS and introduced a version of GDBSCAN called TRAJ-DBSCAN for filtering.

In CuTS, TRAJ-DBSCAN (DBSCAN for trajectories) is used for filtering. By using the closest distance between two trajectories as their distance, TRAJ-DBSCAN clusters simplified trajectories of objects' movement during a time-partition $T = [t_a, t_b]$ in order to produce a set of trajectory-clusters \mathcal{L} such that for $t_a \leq t_i \leq t_b$, if a density-connected cluster S_i is found at time t_i , then there exists $L \in \mathcal{L}$ and $S_i \subseteq L$. In the filtering step, trajectory-clusters of consecutive time-partitions are joined to form convoy candidates that probably exist for w time-stamps or more. In verification phase, density-connection between the members of each convoy candidate are examined.

Although CuTS is better than the verification step alone, it cannot produce a complete list of convoys since false negatives can be introduced as described as follow. In Fig. 3.4, movement of nine objects across five λ -length time partitions are shown. There are two convoys (shown in shaded area) — $C_1 = \{a, b, c\}$ and

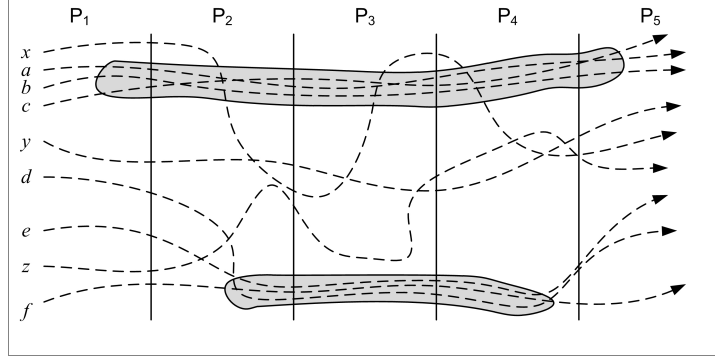


Figure 3.4: An Example Scenario, Where Algorithm CuTS Has False-negatives due to Accidentally Linked Convoys.

$C_2 = \{d, e, f\}$ — for $min_pts = m = 2, w = \lambda$. Since TRAJ_DBSCAN uses the shortest distance between two trajectories, the objects x , y , and z will density-connect the trajectories of a , b , and c with those of x , y , and z in P_2 resulting in a single trajectory-cluster $\{a, b, c, d, e, f, x, y, z\}$ for P_2 . After the filtering step has joined it with the trajectory cluster $\{a, b, c\}$ that existed in the previous partition P_1 to obtain convoy-candidates, CuTS will only maintain the trajectory cluster $\{a, b, c\}$ from P_1 to P_2 as the only convoy-candidate. Therefore, in refinement phase, C_2 will not be checked whether it forms a convoy or not and will not be reported.

Table 3.1 summarizes the models of a group of tracked objects moving together, which we have presented. All three models — fixed flocks, moving groups, and convoys — do not allow new members to join the existing group or allow existing members to leave (whether it returns or not) but enforce all members of a group to stay together throughout the group’s life-span. The only difference between these models is how they define togetherness of the group’s members. In this thesis, we will refer these models (definitions) of moving groups as “traditional convoys”.

3.2.5 Moving Clusters

Kalnis et al. [33] proposed two exact algorithms and an approximation algorithm to find *Moving Clusters*. Clusters found in consecutive time-slices are defined to be

Table 3.1: A Comparison of the Traditional Convoy Models.

Characteristics	Fixed Flocks	Moving Groups	Convoy
Togetherness Criteria	Covered in a circle of user-defined fixed-size	Within a user-defined fixed-distance from each other	In the same maximal density-connected cluster
Continuity	Set equivalence	Set equivalence	Set equivalence
Join	No	No	No
Leave	No	No	No
Leave and Return	No	No	No

a moving cluster if their Jacquard similarity is higher than a user-defined threshold (θ). Moving cluster model does not have an explicit notion of members and allows member objects to leave or enter in the cluster in its lifetime. Therefore, moving cluster permits clusters to be completely different (larger, smaller *etc*) in a short time. It also report multiple moving clusters when cluster transitions (merge, split *etc*) occurs.

3.2.6 Swarm

In more recent works [39, 40], a Swarm is defined as a group of at least m' objects, which are found together for k (possibly non-consecutive) time-stamps. Therefore, the definition of the Swarm pattern allows member objects to move away from each other yet it enforces a discipline such that they must re-group frequently-enough to form a Swarm pattern. ObjectGrowth algorithm, which is based on the *apriori*-properties and FP-growth algorithm, is proposed in [39] and is later incorporated into the MoveMine data-mining system [40].

Figure 3.5 compares all existing models (definitions) described above in a two-dimensional plot. Traditional convoys neither allow dynamic members, which temporarily move away from the parent convoy, nor present convoys in different evolving stages. Moving Cluster is able to capture the different stages of a convoy but has

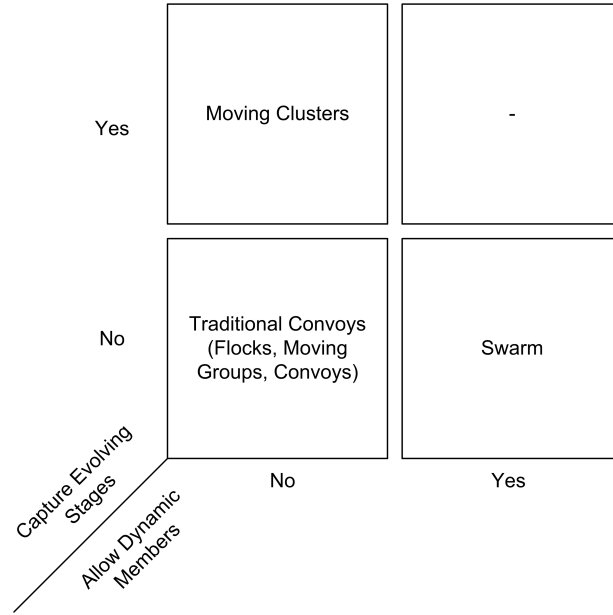


Figure 3.5: Comparison of Existing Moving Group Models.

no explicit rule to enforce objects leaving the convoys to return. Swarm allows members to move away from the group yet it cannot capture the evolving stages of convoy in an intuitive manner. To the best of our knowledge, there is no convoy model that can both allow dynamic members and capture evolving stages of a convoy. The dynamic convoy we are going to introduce in Chapter 6 is similar to Swarm [31]. The new definition of convoy, namely “evolving convoy,” which we are going to define as an evolution sequence of dynamic convoys and study in Chapter 6, allows dynamic members, captures evolving stages of a convoy, and, hence, will fit in the upper right quadrant in Fig. 3.5.

3.2.7 Sub-trajectory Clusters

In order to discover frequent routes in a Trajectory Database (TJDB), the paths on the spatial space that the moving objects frequently use to travel, the earlier works [22, 42] suggested to divide the spatial region into regions (or a hierarchy of regions), transform the object movements (trajectories) in the TJDB as sequences of regions and perform sequence mining on the resulting sequences. This approach has

two obvious drawbacks. First, they need a pre-processing step and a prior knowledge of the underlying spatial region in order to divide the entire map (spatial space) into regions. Secondly, the granularity of resulting frequent route is reduced depending on the size of the regions. Therefore, they are not suitable in situations, where a prior knowledge of the spatial region is not available and/or a high granularity (higher resolution/more accurate) output frequent routes are required.

In the absence of underlying spatial information such as road network data, however, researchers suggested a two-step methods — (a) group similar sub-trajectories together and (b) extract a representative route from each group [12, 37, 61]. Lee *et al* [37] suggested to divide the trajectories into simplified trajectory-segments (line-segments), perform clustering using an extension of GDBSCAN and calculate representative trajectories. They proceeded to experiment their proposed solution on hurricane trajectories. Their method, however, is not applicable to GPS traces containing trajectories that cross each other often [35]. Zhu *et al* [61] proposed a similar method, in which partition is performed by a grid of uniformly sized cells and combining trajectory clusters found across cells.

Buchin *et al* [12] suggested that one can simply choose an arbitrary sub-trajectory in a group as its representative route (reference trajectory) if the group contains only sub-trajectories, which are *strictly similar* to each other. They proposed to use Fréchet distance as the similarity measure to ensure each sub-trajectory group contains strictly similar sub-trajectories. They define a sub-trajectory cluster as a set of sub-trajectories such that (a) it contains m distinct sub-trajectories, (b) the longest of them is not shorter than l (length, not duration), and (c) all its sub-trajectories are within a distance r from each other, where m , l , and r are user-defined parameters. They proved that, given a TJDB of size n and parameters, r , m , and l , finding longest sub-trajectory clusters is NP-Complete. They also proposed approximation algorithms having an approximation factor of 2 for

several variants of their problem. Their solution tries to traverse and report all sub-trajectories having $m - 1$ other sub-trajectories within a Fréchet distance of r as the approximations of the sub-trajectory clusters and does not have any pruning mechanism to remove false-positives that the approximation introduces.

Fréchet Distance

Fréchet distance is regarded as a natural measure to quantify similarity of curves [17]. Given two polygonal curves, $a : [s_a, e_a] \rightarrow \mathbb{R}^2$ and $b : [s_b, e_b] \rightarrow \mathbb{R}^2$, and the sets of re-parametrizations $\mathcal{A} = \{\alpha | \alpha : [0, 1] \rightarrow [s_a, e_a]\}$ and $\mathcal{B} = \{\beta | \beta : [0, 1] \rightarrow [s_b, e_b]\}$ for them, the Fréchet distance $dist_{Fr}$ between a and b , is defined as the minimum $dist_{max}$ over \mathcal{A} and \mathcal{B} , where $dist_{max}(\alpha, \beta)$ is the maximum distance between $a(\alpha(x))$ and $b(\beta(x))$ for all $x \in [0, 1]$.

An early treatment on computing Fréchet distance for two polygonal curves is given in [5]. It reports that for two curves defined by p and q points, computing Fréchet distance between them needs $O(pq \log(pq))$ time, while deciding whether the Fréchet distance between them is less than a given threshold r needs $O(pq)$ time. They defined a two-dimensional data structure called Free-space as a set of points that visualize whether a pair of two points in the given two polygonal curves are within a Euclidean distance of r and proved that the Fréchet distance between two polygonal curves is not more than r if and only if there is a monotone curve in the corresponding Free-space. Their results are extended for a set of polygonal curves in [18] by reporting the computing and decision problems for a set of k curves defined by n_i points (for $1 \leq i \leq k$) need $O(n_1.n_2.\dots.n_k \log(n_1.n_2.\dots.n_k))$ and $O(n_1.n_2.\dots.n_k)$ times respectively.

Based on the definition given in [55], we conclude that the discrete version of Fréchet distance is essentially Dynamic Time Wrapping (DTW). In this regard, the Fréchet distance can be considered as the generalization of (the continuous version

of) DTW, where (periodic re-)sampling of points for long line segments are not required to calculate the distance between two polygonal curves.

3.2.8 Other Movement Patterns

Co-location pattern mining to report co-occurrence of objects in different sets (with different labels) satisfying a minimum occurrence threshold (across time) and spatial-support threshold is found in [13]. Notation of micro-clustering introduced by Han *et al.* [38] is capable of finding clusters of moving objects not only by closeness of their location but also by similarity of their velocity (direction and speed).

Various spatial-patterns (Spatial Object Association Patterns — SOAP) of the moving objects are described in [56] along with definitions of the related distance metrics (*isClose*) as well as orientation-specific (*isAbove*) predicates. In their work, four different types of SOAPs — Star, Clique, Sequence, and minLink — and respective algorithms based on ECLAT to mine those SOAPs in spatial-temporal datasets are also described. When we limit a label to be assigned to exactly a single object, clique SOAP and minLink SOAP are equivalent to Swarm patterns with distance-based grouping criteria and Density-based cluster grouping criteria respectively. Since spatial-relationships (SOAPs) evolves over-time, various temporal episodes — known as Formation (at least one of the SOAP type in question appears), Dissipation (all instances of the SOAP type in question disappear), and Continuation (at least one of the SOAP type in question exists in next time-stamp) — and techniques to analyze and to make inference from them are also presented. A similar work dealing with orientation of features is [59], where mining of complex spatial-patterns are reduced to simpler sequence mining problem. They claim that the decomposition algorithm they proposed is both efficient and scalable than tradition *Apriori* and Enumeration based algorithms.

Finding Closed MEMOs

4.1 Introduction

In this chapter, we will present our model (definition) of meeting patterns, the three algorithms we developed to extract meeting patterns from a given Trajectory Database (TJDB), and the results of the experiments we conducted to assess the performance of our proposed algorithms in finding meeting patterns.

Informally, a meeting is formed when a group of objects comes to and stays in a fixed (circular) area for a while. Although the meeting area (meeting place) can be any geometric shape, we choose to define it to be a circle because there is an existing work [23] that studies meetings formed in circular regions. However, the techniques we develop to find meetings in circular areas can also be used to report meetings in other geometric shapes, say, rectangles.

The primary application of the meetings is to use the information of the member objects, which form a meeting pattern, to light insights into the behaviours and interactions of the objects under analysis. For example, from the information of the commuters, who form meeting patterns during lunch time, advertising agencies can discover their social interactions — and deduce purchasing decision influence among social groups. Likewise, the existence of meeting patterns formed by navel vessels, except at designated places like docks and ports, indicates suspicious activities such as illegal transfer of goods and personnels.

The secondary application of the meetings is to analyse the information of meetings in order to discover knowledge on trends of meeting places (and times), in which the meetings are formed. For instance, meetings of commuters show trends in popular restaurants and lunch-time services such as shopping centres and hair saloons *etc.* Similarly, meeting places, times, and durations of wild-animals show changes in their natural habitats. Such information can be used by market-researchers and ecologists to plan advertisements and further researches.

A useful application scenario of the meeting place information is in planning the deployment of mobile service centres. For example, in developing countries, the concept of a mobile internet centres becomes very popular as it allows resources (equipments, staffs *etc.*) to be shared by patrons residing in different geographic areas. In other words, the mobile internet centres are rotating among different geographical regions (different villages). In order to maximize the utilization of a mobile internet centre, it should be deployed in such a way that it is available (within r meters) to a number of (say m) patrons. To maximize its utility and availability, a mobile internet centre must be accessible for a sufficient amount of time (say w hours) for each patron so that he can come to the internet centre at his convenience (e.g. during lunch hour, after work *etc.*). On the other hand, if the mobile internet centre is not accessible long enough (e.g. the patron can only glimpse it while he is rushing to work), the patron may choose not to visit the mobile internet centre in favour of following his tight schedule, reducing the utility of the mobile internet centre. Knowledge of the past meetings (in circular areas of radius r) formed by patrons can assist in efficient planning of the deployment of mobile internet centres and other similar services like mobile libraries and mobile clinics (by indicating the centres of past meeting places as the ideal places to locate the mobile services).

The above application scenario of meeting places is similar to finding ideal region to start up a business described in [60]. However, the works in [60] finds an ideal

region to start a **permanent** business based on a set of static locations of customers objects with weights, each representing a building residing a number of potential customers, while the application scenario meeting places has is to identify an ideal place for **temporary** (mobile) service based on the historical patterns of the moving potential customers' movements (or the lack thereof). Moreover, the ideal region provided by [60] aims to be the k^{th} **nearest neighbour** of (or closer than the k^{th} nearest neighbour to) as many customers as possible, while the meeting place indicates a spot, which is **not more than a given distance** r from any potential customer. We noted that, in the cases, where the user needs to place a temporary mobile service centre such that it is the k^{th} nearest neighbour of as many mobile customers as possible, the two techniques can be combined — in the first step, the user can find meetings formed in smaller meeting places and, in the second step, use the meeting places and number of members (and meeting duration) as the static locations and their corresponding weights as input to find ideal region(s).

To discover meetings, one may opt to count objects using proximity sensors (RFID readers) at the potential meeting places rather than mining Trajectory Databases containing GPS traces. However, this approach has several drawbacks. Firstly, proximity sensors have limited ranges, thus, it often requires to aggregate data from multiple sensors to discover a single meeting. Since an object can be detected by multiple sensors, resulting in duplicate readings, aggregation is not a straight-forward task. Secondly, as the sensors need to be deployed at *potential* meeting places, meetings formed in unexpected places will not be discovered. Moreover, tokens (RFID tags) must be attached to the objects in advance, which is highly impractical. On the other hand, many people are accustomed to sharing their GPS traces with friends and businesses (or sponsors of mobile internet centres) through location-sharing services.

Similarly, clustering the sampled GPS points using a three-dimension (two space

and one time) distance in order to find meetings is impractical because it cannot guarantee that each object that visits the sampled location remains in the area for a specific user-defined time. Moreover, in order to use k -mean clustering or its variants, we need a prior knowledge of the number of meetings we are going to find and the potential meeting places, which we do not have. Clustering techniques, which do not need any prior knowledge of the number of meetings and meeting locations such as density-based clusterings (like DBSCAN) also cannot find meetings confined in a meeting place of user-defined size as points far apart may also be density-connected enlarging the resulting meeting place.

Chapter Contributions

To the best of our knowledge, there has been only a limited amount of studies on discovery of meeting patterns and no implementation (experimental evaluation) exists. Our contributions include (a) introducing a new definition of meeting patterns that defines what meeting members, places, and times are, (b) developing three new algorithms to discover meeting patterns from Trajectory Databases, and c) experimenting them along with our adaptation of the algorithm proposed in [23].

4.2 Finding Closed MEMOs

Definition. 4.1. MEMO Pattern – For given parameters: $m > 1$, $r > 0$, and $w \geq 1$, a set of objects M forms a **Meeting of Moving Objects**, or a **MEMO**, during the time-interval $I(M)$ at the circular region $loc(M)$ if (i) M has at least m objects, (ii) $I(M)$ spans for at least w consecutive time-stamps, (iii) $loc(M)$ has a minimal radius $r(M) \leq r$, and (iv) all objects $o \in M$ reside in $loc(M)$ in each $t \in I(M)$.

Definition 4.1 defines a MEMO formed by m or more objects. In order to form a MEMO, the participating objects must stay in a circle, whose radius is not larger

than r , for at least w time-stamps. In Fig. 4.1(a), which shows movement of five objects for three time-stamps and a circle (drawn in dotted lines) with radius r , $M = \{o_1, o_2, o_3, o_4\}$ forms a MEMO from t_1 to t_3 for parameters : $m = 3$ and $w = 2$. The meeting place of M is the circular region $loc(M)$, whose radius is $r(M) \leq r$ and in which all objects in M stay. For the mobile service centre application discussed in the previous section, Sect. 4.1, the planner can set m , r , and w as the minimum number of patrons the service should be available to, the maximum distance of the service centre to all patrons' typical roaming locations, and the minimum interval the service is available to the patrons to find places (and time-interval) in the past, which are ideal for the service centre's location (and operating hours). From the past MEMO information, the planner can deduce an ideal place (and operating hours) of the service centre.

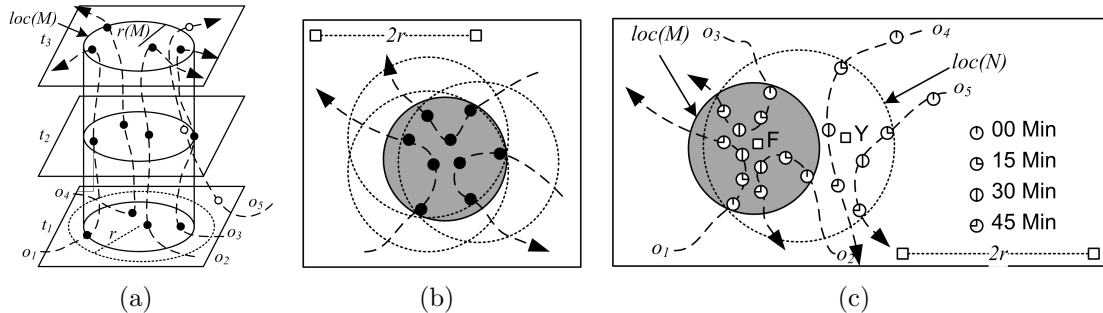


Figure 4.1: Examples of (a) a MEMO, (b) an Accurate Meeting Place, and (c) Two Overlapping Closed MEMOs.

Our definition of the meeting pattern, MEMO, is slightly different from the one given in [23] as it explicitly defines the meeting place as the smallest possible circle in contrast to implicitly defining it as a circle of fixed radius r . As a result, algorithms, which are based on our definition, are able to report more accurate and non-ambiguous meeting places. For example, for the meeting pattern formed by three animals in Fig. 4.1(b), the most accurate place of their habitat (the shaded circle), rather than a larger, less accurate circular regions of radius r (three shown as dotted circles), can be reported.

Definition. 4.2. Closed MEMO – A MEMO M is a closed MEMO if and only if there is no other MEMO $M' \neq M$ such that M' contains all members of M and the life-span of M' completely covers that of M .

Definition 4.2 formally defines the concept of a closed MEMO of maximal interval and maximal set of members. For example, in Fig. 4.1(a), for parameters : $m = 3$ and $w = 2$, $M'_1 = \{o_1, o_2, o_3\}$ from t_1 to t_3 and $M'_2 = \{o_1, o_2, o_3, o_4\}$ from t_1 to t_2 are non-closed MEMOs as there is a closed MEMO $M = \{o_1, o_2, o_3, o_4\}$ from t_1 to t_3 covering them.

Overlaps between closed MEMOs are possible and, in fact, necessary. Consider two docks, dock F serving class-F ships and dock Y serving class-Y ships as shown in Fig. 4.1(c). For parameters $m = 3$ and $w = 30$ minute, both $M = \{o_1, o_2, o_3\}$ from 0 to 45 minute and another MEMO $N = \{o_1, o_2, o_3, o_4, o_5\}$ from 15 to 45 minute are closed MEMOs as one does not cover the other. This result intuitively agrees with the observation that each meeting may be significant and informative to different types of users.

Definition. 4.3. Finding Closed MEMOs – Given a Trajectory Database \mathcal{R} and parameters : $m > 1$, $r > 0$, and $w \geq 1$, the task of **Finding closed MEMOs** is to list the complete information of all closed MEMOs formed in \mathcal{R} according to m , r , and w .

4.3 Algorithms for Finding Closed MEMOs

We developed three new algorithms to discover closed meetings of moving objects (closed MEMOs). The first algorithm uses the *apriori*-properties of MEMOs, while the second employs ECLAT partitioning to divide the search space into partitions, achieving practical efficiency. The last algorithm introduces a filtering step for the first and second algorithms. We will present some preliminaries before a detailed

discussion on our algorithms.

Definition. 4.4. k -object-set – A set of objects containing exactly k members will be called a k -object-set.

Definition. 4.5. k -MEMO – For given parameters: $r > 0$ and $w \geq 1$, a k -object-set O_k forms a k -MEMO M_k during the time-interval $I(M_k)$ at the circular region $loc(M_k)$ if (i) $I(M_k)$ spans for at least w consecutive time-stamps, (ii) $loc(M_k)$ has a minimal radius $r(M_k) \leq r$, and (iii) all objects $o \in O_k$ resides in $loc(M_k)$ in each $t \in I(M_k)$.

Definition. 4.6. k -closed MEMO – A k -object-set O_k forms a k -closed MEMO M_k if and only if O_k does not form another k -MEMO $M'_k \neq M_k$ such that the life-span of M'_k covers that of M_k .

Definition. 4.7. MEMO-List (M-List) – For a given k -object-set O_k , the set of all its k -closed MEMOs $L(O_k) = \{M_{k1}, M_{k2}, M_{k3}, \dots, M_{kj}\}$ is called the MEMO-List (M-List) of O_k .

Definition 4.4 defines a k -object-set, which can form zero or more k -MEMOs in a given dataset \mathcal{R} . Definition 4.5 is a relaxed version of Def. 4.1. A k -MEMO has exactly k participants (in contrast to the fact that a MEMO must have at least m participants). For $k \geq m$, a k -MEMO is a MEMO. Following Def. 4.6, a k -closed MEMOs cannot be covered by another MEMO having k (or fewer) members. Therefore, all closed MEMO having k participants are k -closed MEMO (the reverse is not always true). Definition 4.7 defines a MEMO-List, which groups all k -closed MEMO formed by a single k -object-set.

4.3.1 An Apriori-based Closed MEMO Miner

Lemma. 4.1. If a set of points P is covered by a minimum covering circle C , the minimum covering circle C' of its subset P' is not larger than C .

Proof. For a set of points P , the minimum covering circle C is defined as the smallest possible circle that covers all points $p \in P$.

For any set of points P' , its minimum covering circle C' always exists and C' is unique for P' [16].

Therefore, any circle D , which covers all points in P' , cannot be smaller than the minimum covering circle C' of P' , i.e. if D covers all points in P' , then $r(C') \leq r(D)$, where $r(X)$ stands for the radius of circle X .

Since C covers P and $P' \subseteq P$, C also covers P' .

Therefore, $r(C') \leq r(C)$. □

Using Lemma 4.1, we can derive the *apriori*-properties of MEMOs as follow: for a given time-interval I , if a set of objects M forms a MEMO, there is a circle $loc(M)$ having a radius $r(M) \leq r$ and enclosing the set of locations (points) $L = \{loc|\langle o, t, loc \rangle \in \mathcal{R}, o \in M, \text{ and } t \in I\}$. The corresponding set of locations $L' = \{loc|\langle o, t, loc \rangle \in \mathcal{R}, o \in M', \text{ and } t \in I\}$ of $M' \subseteq M$ is, by Lemma 4.1, covered by a circle $loc(M')$ not larger than $loc(M)$, i.e. $r(M') \leq r(M)$. Thus, $r(M') \leq r$ and M' forms a MEMO during the interval I . Therefore, all subsets of a MEMO are MEMOs. In other words, for any time-interval, M does not form a valid MEMO if any of its subset does not.

The *Apriori*-based closed MEMO miner (A-miner), adapted from the *Apriori*-algorithm in [3], exploits the *apriori*-properties of the MEMOs to systematically discover the MEMOs formed by $(k + 1)$ -object-sets only when those formed by its subsets, k -object-sets, exist. An outline of the A-miner is given in Algorithm 4.1. The function Closed-MEMO(\mathcal{R} , w , r , O_k) returns the sorted list (M-List) of k -closed MEMOs formed by O_k .

The A-miner initializes the M-Lists of 1-object-sets, which are likely to form larger MEMOs (lines 2-5). Starting with $k = 1$, the M-List of each $k + 1$ -object-set

Algorithm 4.1 Apriori-based Closed MEMO Miner (A-Miner).

Input: \mathcal{R} , r , m and w .

Output: A set of closed MEMO \mathcal{M} .

```
1: The set of 1-object-sets  $\mathcal{C}_1 \leftarrow \emptyset$ ,  $\mathcal{M} \leftarrow \emptyset$  and  $k \leftarrow 1$ 
2: for all  $o \in O$  do
3:   Object set  $O_1 \leftarrow \{o\}$  and M-List  $L(O_1) \leftarrow \text{Closed-MEMO}(\mathcal{R}, w, r, O_1)$ 
4:   if  $L(O_1)$  is not empty then
5:      $\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup \{O_1\}$ 
6: while the set of  $k$ -object-sets  $\mathcal{C}_k \neq \emptyset$  do
7:   for all  $O_k \in \mathcal{C}_k$  do
8:     if  $k \geq m$  then
9:        $\mathcal{M} \leftarrow \mathcal{M} \cup L(O_k)$ 
10:  The set of  $(k + 1)$ -object-sets  $\mathcal{C}_{k+1} \leftarrow \emptyset$ 
11:  for all  $O_k, O'_k \in \mathcal{C}_k$  such that  $|O_k \cap O'_k| = k - 1$  do
12:     $O_{k+1} \leftarrow O_k \cup O'_k$  and  $L(O_{k+1}) \leftarrow \text{Closed-MEMO}(\mathcal{R}, w, r, O_{k+1})$ 
13:    if  $L(O_{k+1})$  is not empty then
14:       $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_{k+1} \cup \{O_{k+1}\}$ 
15:   $k \leftarrow k + 1$ 
16:  $\mathcal{M} \leftarrow \mathcal{M} - \{M \mid M \text{ is not a closed-MEMO}\}$ .
```

is built only if two of its subset k -object-sets have non-empty M-Lists (lines 6-15).

In doing so, if $k \geq m$, the MEMOs in the M-Lists of the k -object-sets are potential closed MEMOs, thus, they are put into the result set \mathcal{M} (lines 7-9), which is finally filtered (line 16).

Figure 4.2(a) shows an example of moving object dataset containing four objects. For parameters: $m = 2$ and $w = 3$, the lattice, which represent the corresponding search space is shown in Fig. 4.2(b), while Table 4.1 shows the corresponding trace of execution of Algorithm 4.1. A-miner starts at the bottom of the lattice with 1-object sets $\mathcal{C}_1 = \{\{a\}, \{b\}, \{c\}, \{d\}\}$, each of which are attached with its corresponding M-List in the lattice. For example, since, during each of the intervals $[t_1, t_3]$ and $[t_4, t_6]$, the locations of b is covered by a circle, $\{b\}$ is attached with two entries, 1-3 and 4-6. From 1-object-sets, 2-object-sets are extracted in the first iteration $k = 1$. For example, from Fig. 4.2(a), we can see a and b are together in t_1, t_2 and t_3 . Therefore, $\{a, b\}$ is put into \mathcal{C}_2 . However, a and d never met (never were inside the same small circle with) each other, therefore, \mathcal{C}_2 does not contain

$\{a, d\}$. In the next iteration $k = 2$, the 2-MEMOs in the M-Lists of each 2-object-set in \mathcal{C}_2 are put into potential result set \mathcal{M} . The 3-object-set $\{a, b, c\}$ is put into \mathcal{M} in the last iteration $k = 3$, in which no 4-object-set has non-empty M-List. A-miner will finally remove $\{b, c\}$ from \mathcal{M} as the k -MEMO it forms is covered by the one formed by $\{a, b, c\}$ and, hence, is not a closed-MEMO.

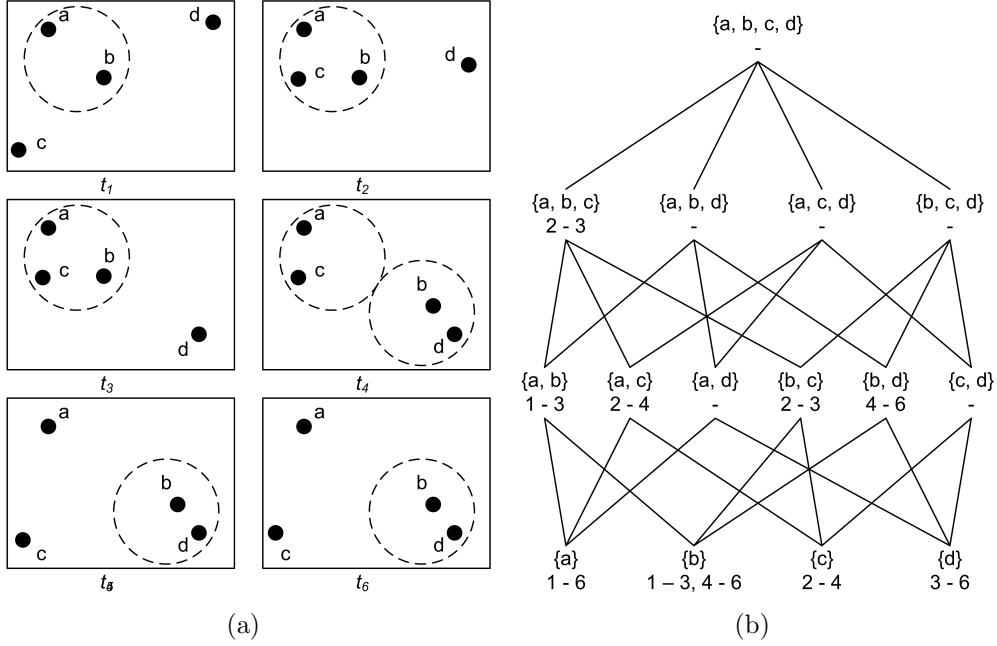


Figure 4.2: (a) Movements of Four Objects and (b) the Corresponding Lattice.

Table 4.1: A Trace of A-miner on the Movments Shown in Fig. 4.2.

k	\mathcal{C}_k	\mathcal{M}
1	$\{\{a\}, \{b\}, \{c\}, \{d\}\}$	\emptyset
2	$\{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}\}$	$\{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}\}$
3	$\{\{a, b, c\}\}$	$\{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{a, b, c\}\}$
4	\emptyset	$\{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{a, b, c\}\}$

Definition. 4.8. k -Prefix – For a set of objects $S \subseteq O$ containing at least k objects and an order \prec defined on O , the list $P_k(S)$ containing the first k elements of S sorted according to \prec is called a k -prefix of S . 0-prefix is always an empty set.

In A-miner, finding pairs of k -object-sets sharing $k - 1$ objects to build the M-Lists of $(k + 1)$ -object-sets (line 11) is an expensive operation. Moreover, for each

$k + 1$ -object-set, O_{k+1} , there are $k^2 + k$ possible pairs of $O_k, O'_k \in \mathcal{C}_k$ such that $|O_k \cap O'_k| = k - 1$, $O_k \subset O_{k+1}$ and $O'_k \subset O_{k+1}$ leading to redundant calculations of its M-List $L(O_{k+1})$. Therefore we use an order among the moving objects in order to have a canonical form of each object set. If two k -object-sets, O_k and O'_k , share the same $k - 1$ -prefix, then we build the M-List of $O_{k+1} = O_k \cup O'_k$, ignoring other pairs of the subsets of O_{k+1} .

Building M-List of O_{k+1}

A frequent component in A-Miner is calculating the M-List of $(k + 1)$ -object-set (line 13 in Algorithm 4.1). A naive method to do so is to check if O_{k+1} forms a MEMO in each maximal time-interval I spanning w or more time-stamps as outlined in Algorithm 4.2. For each potential start-time $start$ and end-time $ts - 1$, the minimum covering circle C of all locations of $o \in O_{k+1}$ from $start$ to ts is calculated (lines 5-6). The variables, $start$ and ts , are adjusted whenever the circle C is larger than that of radius r (lines 11 - 12). Otherwise, ts is increased until the radius of C becomes larger than r (line 14). If the interval spans w or more time-stamps, the algorithm finds a k -closed MEMO in the maximal interval spanning from $start$ to $ts - 1$ at location C' and appends it to the result $L(O_{k+1})$ (lines 7-10).

We introduced two optimizations to Algorithm 4.2. In the first optimization, we further exploited the *apriori*-properties of MEMOs. For a given interval I , if O_{k+1} forms a $(k + 1)$ -closed MEMO, a k -object-set $O_k \subset O_{k+1}$ and $O_{k'} \subset O_{k+1}$ such that $|O_{k'} - O_k| = 1$ must form k -closed MEMO(s) and k' -closed MEMO(s) covering I . Therefore, we utilized the M-Lists of O_k and O'_k , which are readily available in memory, to compute the M-List of O_{k+1} . Our implementation applies Algorithm 4.2 only on the intervals covered by the k -closed MEMOs $M_i \in L(O_k)$ and $M'_j \in L(O'_k)$. Since k -closed MEMOs cannot cover each other, each M-List can be sorted in the temporal order, enabling us to utilize a simple sort-merge-join

Algorithm 4.2 Sub-routine Closed-MEMO Used in the Apriori-based closed MEMO Miner (A-Miner).

Input: \mathcal{R} , w , r and O_{k+1} .

Output: A sorted list of $(k+1)$ -closed MEMO $L(O_{k+1})$.

```

1:  $start \leftarrow \min(\{t | \langle o, t, loc \rangle \in \mathcal{R} \text{ and } o \in O_{k+1}\})$ 
2:  $end \leftarrow \max(\{t | \langle o, t, loc \rangle \in \mathcal{R} \text{ and } o \in O_{k+1}\})$ 
3:  $ts \leftarrow start, L(O_{k+1}) \leftarrow \emptyset, C' \leftarrow null$ 
4: while  $ts \leq end$  do
5:    $P \leftarrow \{loc | \langle o, t, loc \rangle \in \mathcal{R}, o \in O_{k+1} \text{ and } t_{start} \leq t \leq ts\}$ 
6:    $C' \leftarrow C, C \leftarrow \text{Min-Covering-Circle}(P)$ 
7:   if  $radius(C) > r$  and  $ts - start \geq w$  then
8:      $members(M) \leftarrow O_{k+1}, loc(M) \leftarrow C'$ 
9:      $t_{start}(M) \leftarrow start, t_{end}(M) \leftarrow ts - 1$ 
10:    Append  $M$  to  $L(O_{k+1})$ 
11:   if  $radius(C) > r$  then
12:      $start \leftarrow start + 1, ts \leftarrow start$ 
13:   else
14:      $ts \leftarrow ts + 1$ 
15:   if  $end - start + 1 \geq w$  then
16:      $members(M) \leftarrow O_{k+1}, loc(M) \leftarrow C$ 
17:      $t_{start}(M) \leftarrow start, t_{end}(M) \leftarrow end$ 
18:     Append  $M$  to  $L(O_{k+1})$ 

```

algorithm to efficiently check such intervals. We only use the naive approach to calculate the (sorted) M-List of 1-object-sets.

In Algorithm 4.2, calculating the minimum covering circle (line 6) from scratch using the Euclidean Messenger Boy algorithm (EMB) described in [16] dominates a substantial amount of runtime during our initial tests. Therefore, as the second optimization, we developed an incremental version of EMB that can derive the new circle C from C' (and P') to introduce further improvements to A-Miner.

Filtering the Result Set \mathcal{M}

Lemma. 4.2. *Consider a k -closed MEMO M_k . If there is no $(k+1)$ -closed MEMO M_{k+1} such that M_{k+1} contains all members of M_k and the life-span of M_{k+1} completely covers that of M_k , then M_k is a closed MEMO.*

Proof. For a MEMO M , we will denote its members and its life-span as $O(M)$ and

$I(M)$, respectively.

Following Def. 4.6, M_k cannot have both its members, $O(M_k)$, and its life-span, $I(M_k)$, be covered by those of another MEMO having k (or fewer) members.

Suppose there is no $(k + 1)$ -closed MEMO such that it contains all members of M_k and its life-span completely covers $I(M)$ but M_k is not closed because there is a $(k + i)$ -closed MEMO M_{k+i} , where $|O(M_{k+i})| = k + i$ and $i > 1$, such that $O(M_k) \subset O(M_{k+i})$ and $I(M_k) \subseteq I(M_{k+i})$.

Lemma 4.1 states that all subsets of M_{k+i} forms a MEMO during $I(M_{k+i})$, i.e. there is a $(k + 1)$ -closed MEMO M_{k+1} such that M_{k+1} contains all members of M_k and the life-span of M_{k+1} completely covers that of M_k .

□

The last step of Algorithm 4.1 is to filter out MEMOs in the result set \mathcal{M} , which are not closed MEMOs (line 16). Using Lemma 4.2, we perform this step earlier during the main loop (before line 15) using the M-Lists of $(k + 1)$ -closed MEMOs, i.e. we use $L(O_{k+1})$ for all $O_{k+1} \in \mathcal{C}_{k+1}$ to determine the k -closed MEMOs in $L(O_k)$ for all $O_k \in \mathcal{C}_k$ are closed or not and immediately output the closed MEMOs containing k participants. This helps us avoid maintaining a large result set until the last filtering step (or writing to disk and reading it back).

Finding MEMOs Formed in Rectangular Places

A-Miner (and the subsequent algorithms we are going to describe next) can be extended to find MEMOs formed in rectangular meeting places. In order to achieve it, instead of calculating minimum covering circle in line 6 of Algorithm 4.2, we calculate minimum bounding rectangle (MBR). The maximum allowable size of meeting place can be controlled by user-defined threshold(s) — maximum area or combination of maximum length and breadth of the MBR. For mining MEMOs formed in rectangular places, the first optimization of the two optimizations we described

above is still applicable. The rectangular equivalent of the second optimization is also trivial — we just maintain the two opposite corners of the MBR.

4.3.2 An ECLAT-based Closed MEMO Miner

In the worst case scenario, the *Apriori*-based closed MEMO miner (A-miner) needs $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ M-Lists of k -object-sets in memory in order to calculate those of $(k + 1)$ -object-sets. For datasets containing records of a large number of moving objects (large n values), the memory requirements of A-miner is tremendous even for modern workstations equipped with several gigabytes of physical memory. Therefore, Equivalent CLAss Transformation (ECLAT), proposed in [58], is used to partition the search space in our ECLAT-based close MEMO Miner (E-miner).

Definition. 4.9. A k -equivalent-class, denoted as $C(Q_k, k)$ contains all object-sets, each of which has at least k objects and has Q_k as their k -prefix, i.e. $C(Q_k, k) = \{S | P_k(S) = Q_k \text{ and } |S| \geq k\}$

Definition 4.9 defines the equivalent-class $C(Q_k, k)$, which contains all object-sets (not necessarily of the same size) having the same k -prefix, Q_k . For example, for the moving objects shown in Fig. 4.2(a), $\{b\}$, $\{b, c\}$, $\{b, d\}$ and $\{b, c, d\}$ belong to the 1-equivalent-class $C(\{b\}, 1)$ as they all have the same 1-prefix, $\{b\}$. All object-sets having more than k objects in $C(P_k, k)$ can be divided into $(k+1)$ -equivalent-classes, each of which having one of the $(k+1)$ -object-sets in $C(P_k, k)$ as its $(k+1)$ -prefix. For example, $C(\{b\}, 1)$ has two 2-object-sets $\{b, c\}$ and $\{b, d\}$ and, thus, the object-sets having more than 1 object, i.e. $\{b, c\}$, $\{b, d\}$ and $\{b, c, d\}$, can be divided into two 2-equivalent-classes, $C(\{b, c\}, 2) = \{\{b, c\}, \{b, c, d\}\}$ and $C(\{b, d\}, 2) = \{\{b, d\}\}$. Algorithm 4.3 shows an outline of E-miner, which, starting with 0-equivalent-class (the whole search space), recursively divides the k -equivalent-class into $(k+1)$ -equivalent-classes until each can fit in memory. E-miner maintains a stack of

equivalent-classes that needs further partitioning.

Algorithm 4.3 ECLAT-based Closed MEMO Miner (E-Miner).

Input: \mathcal{R} , r , m , w and an order \prec on O

Output: A set of closed MEMO \mathcal{M} .

```

1:  $\mathcal{M} \leftarrow \emptyset$  and Push 0-equivalent-class  $C(\emptyset, 0)$  to stack.
2: while Stack is not empty do
3:   Pop  $k$ -equivalent-class  $C(Q_k, k)$  from stack
4:   M-List  $L(Q_k) \leftarrow \text{Closed-MEMO}(\mathcal{R}, w, r, Q_k)$ 
5:   if  $L(Q_k)$  is not empty then
6:      $\mathcal{M} \leftarrow \mathcal{M} \cup L(Q_k)$ 
7:   if  $C(Q_k, k)$  fits in memory then
8:     Prefix  $Q \leftarrow Q_k$ ,  $k \leftarrow k + 1$  and  $\mathcal{C}_k \leftarrow \emptyset$ 
9:     for all  $O_k \in \{Q \cup \{o_i\} \mid \text{if } o_j \in Q \text{ then } o_j \prec o_i \text{ or } o_j = o_i\}$  do
10:       $\mathcal{C}_k \leftarrow \mathcal{C}_k \cup \{O_k\}$ 
11:     while  $\mathcal{C}_k \neq \emptyset$  do
12:       for all  $O_k \in \mathcal{C}_k$  do
13:         if  $k \geq m$  then
14:            $\mathcal{M} \leftarrow \mathcal{M} \cup L(O_k)$ 
15:           The set of  $(k + 1)$ -object-sets  $\mathcal{C}_{k+1} \leftarrow \emptyset$ 
16:           for all  $O_k, O'_k \in \mathcal{C}_k$  s.t.  $|O_k \cap O'_k| = k - 1$  do
17:              $O_{k+1} \leftarrow O_k \cup O'_k$  and  $L(O_{k+1}) \leftarrow \text{Closed-MEMO}(\mathcal{R}, w, r, O_{k+1})$ 
18:             if  $L(O_{k+1})$  is not empty then
19:                $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_{k+1} \cup O_{k+1}$ 
20:              $k \leftarrow k + 1$ 
21:         else
22:           for all  $Q_{k+1} \in \{Q_k \cup \{o_i\} \mid \text{if } o_j \in Q_k \text{ then } o_j \prec o_i\}$  do
23:             Push  $C(Q_{k+1}, k + 1)$  to stack
24:  $\mathcal{M} \leftarrow \mathcal{M} - \{M \mid M \text{ is not a closed-MEMO}\}$ .

```

E-miner starts by pushing the 0-equivalent-class, represented by $C(\emptyset, k)$, onto the stack (line 1). For each top-most k -equivalent-class $C(Q_k, k)$ popped out from stack, E-miner checks if its prefix Q_k forms a k -MEMO first and maintains the result list \mathcal{M} accordingly (lines 3-6). Then, E-miner checks if the $C(Q_k, k)$ can fit into the memory (line 7). If so, it is processed in the same fashion as in A-miner (lines 8-20). Otherwise, the k -equivalent-class is divided into $k+1$ -equivalent-classes (lines 22-23).

Table 4.2 shows a partial trace of E-miner on the search space shown in Fig. 4.2(b) for the same set of parameters: $m = 2$ and $w = 3$. Let us assume the memory

can only hold two object-sets (and their M-Lists). In step 1, it checks the whole search space lattice, $C(\emptyset, 0)$, and, since it cannot fit into the memory (it contains 16 object-sets), pushes four 1-equivalent-classes, $C(\{a\}, 1)$, $C(\{b\}, 1)$, $C(\{c\}, 1)$ and $C(\{d\}, 1)$ onto the stack. In steps 2 and 3, $C(\{d\}, 1) = \{\{d\}\}$ and $C(\{c\}, 1) = \{\{c\}, \{c, d\}\}$ are popped and, since these equivalent-classes can fit into the memory, their members are examined (but nothing is put into the result set, \mathcal{M} since $\{c\}$, $\{d\}$ and $\{c, d\}$ does not form any MEMO for the given parameters). In step 4, $C(\{b\}, 1) = \{\{b\}, \{b, c\}, \{b, d\}, \{b, c, d\}\}$ is popped from the stack. Since it has four object-sets and cannot fit in the memory, it is divided into two 2-equivalent-classes $C(\{b, c\}, 2)$ and $C(\{b, d\}, 2)$, which are pushed onto the stack for later processing. In steps 5 and 6, $C(\{b, d\}, 2) = \{\{b, d\}\}$ and $C(\{b, c\}, 2) = \{\{b, c\}, \{b, c, d\}\}$ are popped out, checked and $\{b, d\}$ and $\{b, c\}$ are inserted into the result set \mathcal{M} . In the next steps, the equivalent-class $C(\{a\}, 1)$ containing eight object-sets will be divided and processed.

Table 4.2: A Partial Trace of E-miner on the Movements Shown in Fig. 4.2.

Step	$C(Q_k, k)$	Stack	\mathcal{M}
1	$C(\emptyset, 0)$	$\{C(\{a\}, 1), C(\{b\}, 1), C(\{c\}, 1), C(\{d\}, 1)\}$	\emptyset
2	$C(\{d\}, 1)$	$\{C(\{a\}, 1), C(\{b\}, 1), C(\{c\}, 1)\}$	\emptyset
3	$C(\{c\}, 1)$	$\{C(\{a\}, 1), C(\{b\}, 1)\}$	\emptyset
4	$C(\{b\}, 1)$	$\{C(\{a\}, 1), C(\{b, c\}, 2), C(\{b, d\}, 2)\}$	\emptyset
5	$C(\{b, d\}, 2)$	$\{C(\{a\}, 1), C(\{b, c\}, 2)\}$	$\{\{b, d\}\}$
6	$C(\{b, c\}, 2)$	$\{C(\{a\}, 1)\}$	$\{\{b, d\}, \{b, c\}\}$
...	

In E-Miner, calculating M-List of the k -Prefix Q_k popped out from the stack (line 4) is a frequent component. Since it is costly to use the naive computation described in Algorithm 4.2, in our implementation, their M-Lists are computed before they are pushed onto the stack. We maintain the M-List of all 2-object-sets and, for any Q_{k+1} about to be pushed onto the stack, its M-List $L(Q_{k+1})$ is computed from the M-List of Q_k and any 2-object-set O'_2 such that $|O'_2 - Q_k| = 1$.

4.3.3 A Filter-And-Refinement Closed MEMO Miner

In A-miner and E-miner, the dataset is referred for the locations of moving objects in calculating the minimum covering circles to verify if the objects actually form a MEMO for the given parameters. Those queries (and computation of the circle) are often wasted when the objects do not form a MEMO (when the radius of the circle is larger than the given r). In Filter-And-Refinement-based Closed MEMO Miner (FAR-miner), we introduced a filtering step, which needs less access to the dataset, to avoid computation of minimum covering circles.

Lemma. 4.3. *If a set of points P is covered by a minimum covering circle C , whose radius $r(C) \leq r$, then two points $p, q \in P$ cannot be further apart than $2r$.*

Proof. No two points $p, q \in P$, which are either inside or on the edge of C , can be further apart than the length of its diameter, i.e. $distance(p, q) \leq 2r(C)$, and $2r(C) \leq 2r$. Therefore, $distance(p, q) \leq 2r$. \square

Lemma 4.3 claims that if the distance between two points $p, q \in P$ is more than $2r$, the minimum covering circle C of P must have a radius larger than r . In other words, if the distance between location of object o_i at t_j and that of object o'_i at t'_j is further than $2r$, o_i and o'_i do not form a MEMO at interval I containing t_j and t'_j .

Definition. 4.10. Potential MEMO – *For given parameters : m , w and r , a subset of the dataset $\mathcal{R}' \subseteq \mathcal{R}$, which contains all movement records of a set of objects O' in an interval $I(\mathcal{R}')$ is termed as a potential-MEMO if (i) O' has at least m objects, (ii) $I(\mathcal{R}')$ spans for at least w consecutive time-stamps and (iii) all locations the objects $o \in O'$ visited during $I(\mathcal{R}')$ are not further than $2r$ from each other.*

Definition 4.10 defines a potential-MEMO, which is likely to form a MEMO for the given parameters. Closed potential-MEMO, k -potential-MEMO and k -closed potential-MEMO can be defined in ways similar to Def. 4.2, 4.5 and 4.6, respectively. It is also apparent that potential-MEMOs also have *a priori*-properties. FAR-miner consists of two steps (i) the **Filtering step**, which finds the set of all closed potential-MEMOs $\mathcal{M}' = \{\mathcal{R}' | \mathcal{R}' \text{ is a closed potential-MEMO}\}$ and (ii) the **Verification step**, which, for each potential-MEMO $\mathcal{R}' \in \mathcal{M}'$, verifies if the objects actually form a MEMO.

To perform the filtering step, we use A-miner (or E-miner), using a slightly modified version of Closed-MEMO as its subroutine, since potential-MEMOs also have the *a priori*-properties. Therefore for the filtering step, instead of building minimum covering circles and checking their radii (lines 6-7), the modified algorithm would simply check the distance between all $p, q \in P$. It is easy to show that, if no two points in each of the sets, $A \cup B$, $A \cup C$ and $B \cup C$ are further than $2r$, no two points in the set $A \cup B \cup C$ are. Therefore, when O_k and O'_k (such that $O_k - O'_k = \{o_i\}$ and $O'_k - O_k = \{o_j\}$) are known to form potential-MEMOs in interval I , whether $O_{k+1} = O_k \cup O'_k$ forms a potential-MEMO in I can be easily derived by checking if all the locations of o_i and o_j in I are within distance $2r$ of each other. In other words, by maintaining all 2-closed potential-MEMOs in memory, when $k \geq 2$, the potential-MEMOs formed by O_{k+1} object-sets can be derived without referring the dataset for the actual locations of the objects. To perform the verification step, we directly apply A-miner (or E-miner) discussed in Sect. 4.3.1 (4.3.2) on \mathcal{R}' with the given parameters.

Since it is possible to perform the filtering and verification using either A-miner or E-miner dialects, there are four possible flavors of FAR-miner. However, in Sect. 4.4, we report the performance of filtering and verification steps, both using A-miner dialect, as all flavors show similar performance during our initial tests.

As a hindsight, we noted that, due to the *a priori*-properties of the MEMOs, intermediate mining results (k -MEMOs and potential k -MEMOs) obtained using parameters : m , w and r can be easily reused for subsequent mining tasks on the same dataset using different parameters : m' , w' and r' , bounded by the criterion : $r' \leq r$ and $w' \geq w$ (the value of m' does not matter as M-Lists are built anyway and can be saved). However, in practice, only the intermediate results obtained from using a short w and large r should be saved as they are more likely to be reused than those from using a longer w and/or a larger r . The intermediate results in question can be further trimmed down to its subset based on the domain knowledge. For example, in an application where meetings of 3 objects are very common and useless (the user is not likely to give parameter $m = 3$), only k -MEMOs for $k \geq 4$ are to be saved for reuse. In our experiments in Sect. 4.4, we do not reuse any intermediate result.

4.4 Experimental Evaluations

4.4.1 Experiment Setup

We adapted the column-sweeping algorithm (CS-miner) proposed in [23] for reference because it is the only work in the literature, which, theoretically, can report all MEMOs accurately. However, as CS-Miner was originally developed for finding the longest duration meetings, it reports a single MEMO multiple times¹. We developed a discrete version of CS-miner, which comprises of two steps (a) computing all MEMOs by rotating the columns representing meeting candidates discretely by a user-defined angular step, θ , which is set to 1° in our experiments unless otherwise stated, and (b) filtering out the MEMOs, which are not closed and/or reported multiple times (duplicates). We implemented all the algorithms in Java. Although

¹For more information of CS-Miner, refer to Sect. 3.2.1

our algorithms (A-miner, E-miner and FAR-miner) can also work on disk-based datasets, we conducted our experiments on an in-memory database because CS-Miner’s run-time benefits from fast time-stamp \times location queries. The in-memory database is indexed by primary key $\langle o, t \rangle$ and a separate R-Tree for location queries in each time-stamp. In order to further bias in favour of the discrete CS-Miner, we also pre-processed the location of each tracked object in each time-stamp by linear interpolation. Table 4.3 shows the number of records each dataset has after preprocessing.

Table 4.3: The Size of the Datasets Used to Assess the Meeting Mining Algorithms after Pre-processing.

Name	No. of Records
Statefair	17,545
Orlando	133,076
New York	118,584
NCSU	128,417
KAIST	404,981
SF-Cab21	1,156,458
SF-Cab22	1,236,497

4.4.2 Results and Analysis

The outcome of the first set of experiments, comparing the performance of the algorithms on human movement datasets, is shown in Fig. 4.3 (note that the y-axis is in log-scale). We were looking for meetings of at least two people ($m = 2$) lasting for at least 15 minute ($w = 15$ minute), which were reasonable choices of parameters for the corresponding datasets. In NCSU and KAIST, we even discovered meetings of up to 3 and 5 students. Our proposed data-driven algorithms, A-miner and E-miner, run faster than CS-miner to find the closed MEMOs as they ignore the fast-moving objects in building M-List of 1-object-sets while CS-miner attempts to build MEMOs containing them in vain. FAR-miner outperforms A-miner and E-miner by a large order of magnitude due to its cheap pruning mechanism.

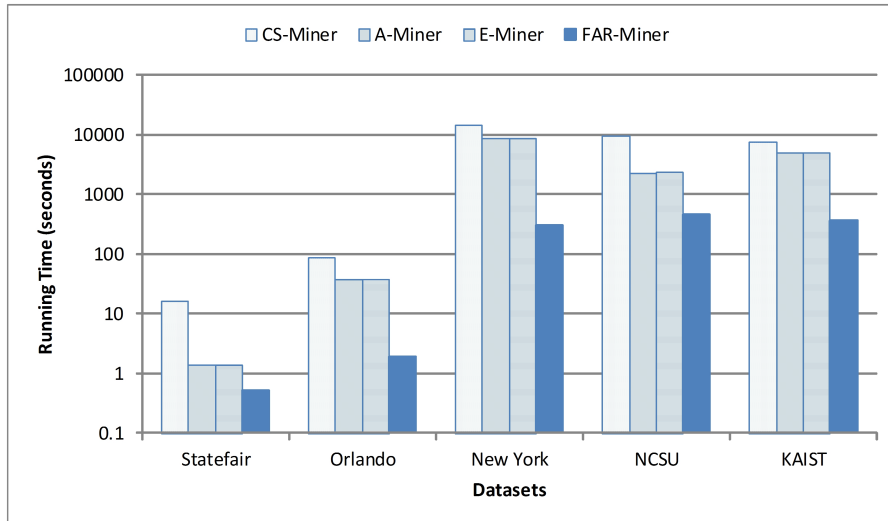


Figure 4.3: Comparison of the Performance of the Closed MEMO Mining Algorithms on Human-movement Datasets Using Parameters $m = 2$, $w = 15$ Minutes and $r = 10$ Metres.

We also collected the garbage collector footprints of A-miner and E-miner in this experiment for New York and KAIST datasets in order to assess the memory usage of the two algorithms. For New York dataset, both algorithm used a comparable amount of memory (allocated 999.2MB at peak usage and freed up 1513.1M by both A-miner and E-miner). However, for KAIST dataset, A-miner allocated 993.6MB at peak usage and freed up a total of 5.6GB through the course of the experiment while E-miner allocated 995.6MB and freed up a total of 6.1GB. Although their peak memory usage are comparable because Java Virtual Machine prefers allocating memory than garbage-collecting (reusing), we concluded that, since E-miner was able to free up more memory than A-miner could, with a trivial programming effort in manual garbage-collection languages like C/C++, E-miner may need less memory than A-miner for the same dataset and parameter settings.

In the next set of experiments, whose outcome is plotted in Fig. 4.4(a) and 4.4(b), we assessed the scalability of the algorithms on different sizes of datasets. We randomly picked 30, 60, and 90 moving objects from the largest human-movement dataset, KAIST, and 150, 300, and 450 moving objects from SF-Cab21 for these

experiments. We set the value of r to 30 metres in order to find MEMOs in smaller subsets of KAIST. For executions on subsets of SF-Cab21, we chose parameters ($m = 5$, $w = 30$ minute, and $r = 25$ metre) to reflect taxis waiting in taxi queues and severe traffic jams. We noticed that the larger value of r we set, the smaller value of θ we should use to maintain CS-miner reports all results since, when we set its internal parameter $\theta = 1^\circ$ in KAIST, it did not report all MEMOs. Therefore, for experiments on SF-Cab21, we used a θ value of 0.5° . Our algorithms (A-Miner, E-Miner, and FAR-Miner) scale well on the increasing dataset size but CS-Miner does not — exceeding eight hours to process data of 90 moving objects in KAIST as there are a large number of MEMOs in this dataset forcing CS-miner to remove more duplicates and the MEMOs, which are not closed. Among our proposed algorithms, FAR-miner performs better than the others except in the largest subset of KAIST, when the very large value of $r = 30$ metres increased the number of potential MEMOs to nearly three thousand and A-miner outperforms FAR-miner.

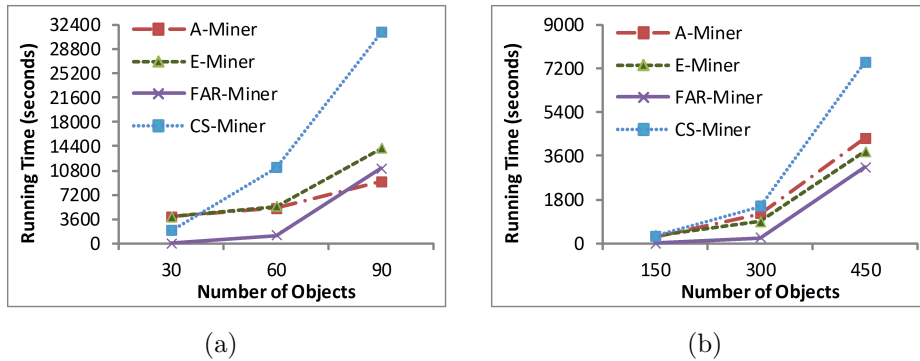


Figure 4.4: Impact of the Size of the Dataset on Performance of the Closed MEMO Mining Algorithms (a) for Parameters $m = 2$, $w = 15$ Minutes and $r = 30$ Metres on Subsets of KAIST and (b) for Parameters $m = 5$, $w = 30$ Minutes and $r = 25$ Metres on Subsets of SF-Cab21.

Analyzing the run-time statistics of FAR-miner in previous experiments given in Table 4.4, we found that run-time is dominated by the verification step as the filtering step took only a few seconds regardless of the dataset. The verification time is not dependent on the size of the dataset but on the number of potential

results (\mathcal{D}'). To verify this claim, we conducted another experiment on SF-Cab22, which is comparable in size to SF-Cab21, using the same set of parameters: $m = 5$, $w = 30$ minutes and $r = 25$ metres. It turns out that there are fewer MEMOs for the given parameters in SF-Cab22 as well as fewer potential results. Subsequently, verification time (and total running time) of SF-Cab22 is smaller than that of SF-Cab21. However, the difference between the verification time of SF-Cab22 and that of SF-Cab21 (3110.3 second vs 1139.8 second) is not directly proportional to the the difference in the numbers of potential MEMOs (1230 vs 129) as the amount of processing the verification step needs also depends on the size of potential MEMOs and the sizes of the potential MEMOs in SF-Cab22 are larger than those of SF-Cab21. Since, in typical circumstances, the number of meetings formed in a dataset are supposed to be few, we noted that FAR-miner will give reasonable performance regardless of the size of input dataset. Even when there are many meetings formed in a dataset, FAR-miner outperforms A-miner and E-miner as they take longer to complete in SF-Cab21 for the same parameters (see Fig. 4.4(b)).

Table 4.4: Run-time Statistics of FAR-miner in the Experiments.

Dataset	Filtering (seconds)	Verification (seconds)	Total (seconds)	No. of closed MEMOs	No. of \mathcal{D}'
Statefair	0.2	0.2	0.4	5	3
Orlando	1.2	0.6	1.8	15	18
New York	17.5	284.1	301.6	16	19
NCSU	6.2	452.1	458.3	48	64
KAIST	9.9	356.6	365.5	126	182
SF-Cab21	18.4	3110.3	3128.7	561	1230
SF-Cab22	19.8	1139.8	1159.6	10	129

In the subsequent sets of experiments, we studied the impact of parameter values on the performance of the algorithm. Figure 4.5(a) and 4.5(b) show the impact of value of r on the performance of the algorithms. Increasing r relaxes the conditions by allowing MEMOs with larger meeting places and increases the number of closed MEMO in a dataset. Thus, it, in turn, increases the run-time of all algorithms.

However, performance of CS-miner degraded rapidly (especially in KAIST) as r increases while our algorithms' performance were stable. Most of the time, FAR-miner significantly outperforms the rest except in KAIST at $r = 30$ metres. In this peculiar instance, there was nearly three thousand potential MEMOs to verify and, since verification time dominates the run-time and depends on the number of potential MEMOs as we noted earlier, FAR-miner took a few minutes more than A-miner to complete.

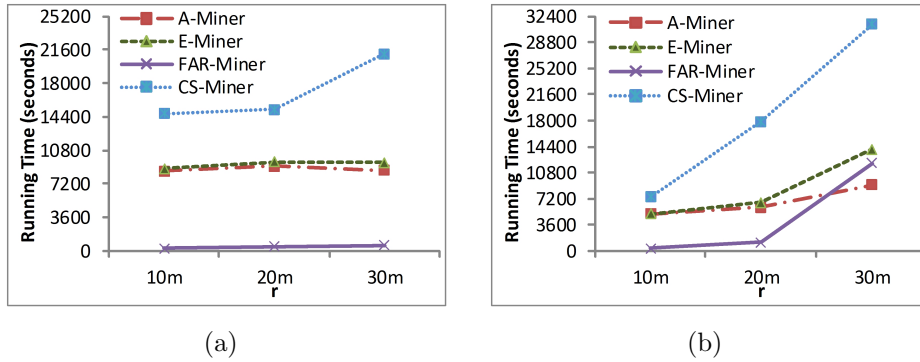


Figure 4.5: Impact of the Parameter r on the Performance of the Closed MEMO Mining Algorithms for $m = 2$ and $w = 15$ Minutes on (a) New York and (b) KAIST.

Figure 4.6 shows the impact of value of m on the performance of algorithms. We used a lower value of $\theta = 0.5^\circ$ to improve CS-miner's accuracy ($r = 30$ metre). Increasing m reduces the number of MEMOs found in a particular dataset. Therefore, CS-miner needs less run-time as m increases while all our algorithms' performance are stable regardless the value of m given. Although FAR-miner ran slower than A-miner in this set of experiments (due to the large value of r leading to a huge number of potential MEMOs), it still performs better than the others due to its powerful filtering step.

Figure 4.7(a) and 4.7(b) shows the impact of value of w on the performance of the algorithms. Increasing w puts more restriction by demanding participants to stay still longer and decreases the number of closed MEMO in a dataset. Thus, it, in turn, decreases the run-time of all algorithms. Our data-driven algorithms still

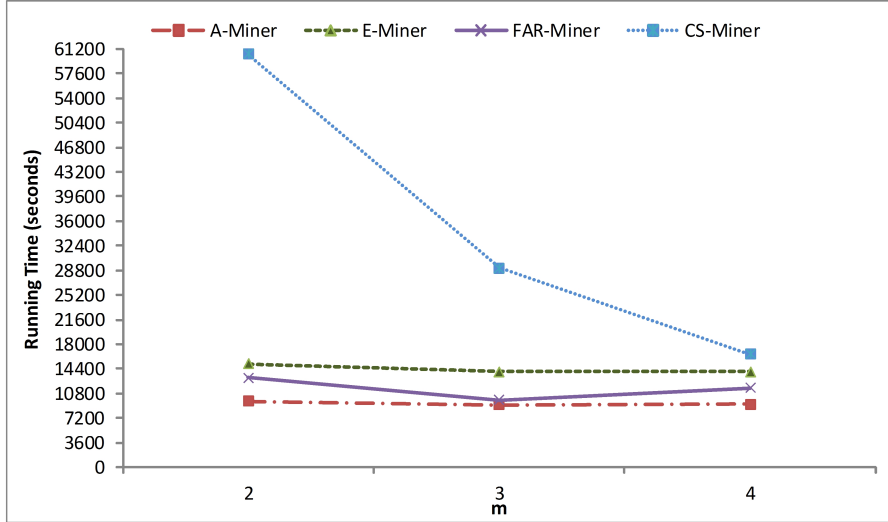


Figure 4.6: Impact of the Parameter m on the Performance of the Closed MEMO Mining Algorithms for $w = 15$ Minutes and $r = 30$ Metres on KAIST.

outperform CS-miner in all cases and, most of the time, FAR-miner significantly performs better than the rest.

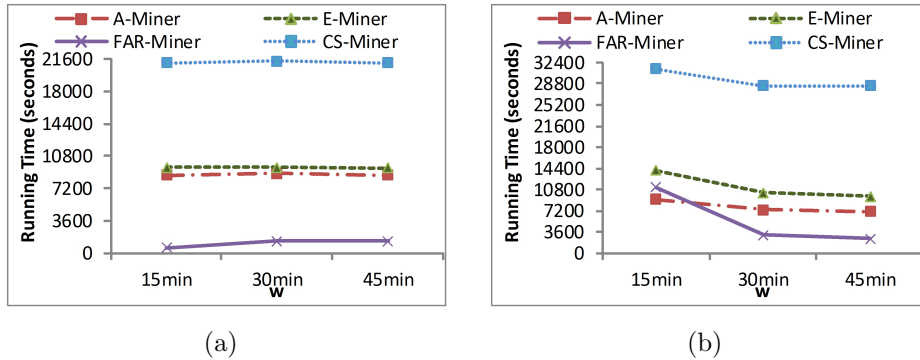


Figure 4.7: Impact of the Parameter w on the Performance of the Closed MEMO Mining Algorithms for $m = 2$ and $r = 30$ Metres on (a) New York and (b) KAIST.

As the last set of experiments, we compared density-clustering of GPS points with our algorithms based on our definition to observe their differences in the output. We were trying to find meetings of at least $m = 2$ objects, which form in a circle of radius $r = 20$ metres and last for $w = 15$ minutes. We used DBSCAN [21] with parameters $min_pts = 5$ and $\epsilon = 20$ metres using the following three-dimensional (two spatial and one temporal) distance:

$$distance(\langle t_1, loc_1 \rangle, \langle t_2, loc_2 \rangle) = \infty \text{ if } |t_2 - t_1| > 1,$$

$$distance(\langle t_1, loc_1 \rangle, \langle t_2, loc_2 \rangle) = distance_{euclidean}(loc_1, loc_2) \text{ if } |t_2 - t_1| \leq 1.$$

The distance measure ensures that two GPS sample points can belong to the same cluster only when they are close not only in space but also in time. The choice of $min_pts = 5$ was due to the reasoning we made that in order to discover meetings containing two members (like our algorithms do), there must be two points in each time-stamp during the meeting duration. Hence, a GPS sample point may have a neighbour in the current time-stamp, two in the next and previous time-stamps each — totalling in five points. The meeting instances our algorithms produced contain members, which stay in their corresponding meeting place for fifteen minutes but many of those clusters reported by DBSCAN contain GPS sample points of objects staying in the area only for a shorter period of time as DBSCAN has no way to ensure each object stayed in the meeting place with others.

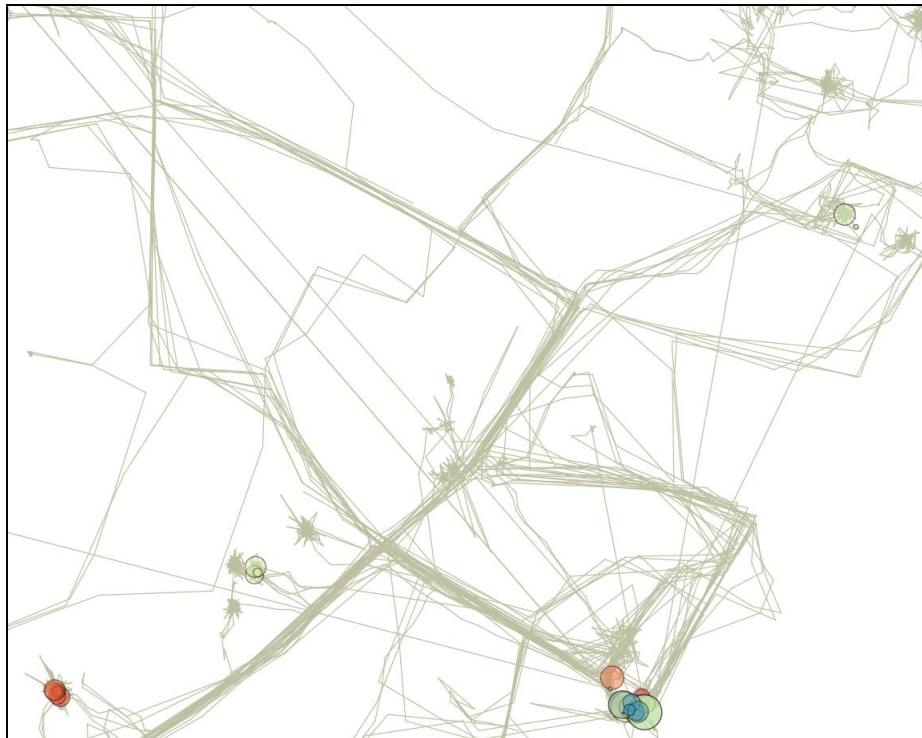
Figure 4.8 compares the meeting places our proposed algorithms produced with the convex-hulls (bounding polygons) of the GPS sample point clusters of DBSCAN. Although a closer look at the raw trajectory data suggested these places as offices and/or service areas in NCSU campus, we were not able to map the meeting locations with actual physical location as the coordinate of the dataset was not known. The meeting places our algorithms produced contain various sizes not larger than a circular area of radius $r = 20$ metre. On the other hand, many of the bounding polygons DBSCAN produced using parameter $\epsilon = 20$ metres are far larger than a circle of radius $r = 20$ metre — most notably the pale triangular shape in the north-western corner of the map, which also appeared even when we use $\epsilon = 10$ metres. It is, we hypothesize, because DBSCAN connects the GPS sample points without any control over the (spatial) size of the resulting cluster. We theorize that it is hard to use clustering to find meeting patterns as it may need a careful calibration of parameters to control the spatial size of meeting place as well as more mechanisms to ensure each of the resulting clusters contains only those GPS sample points of

the objects, which are persistently present in the meeting place for a duration not shorter than the user-defined duration threshold.

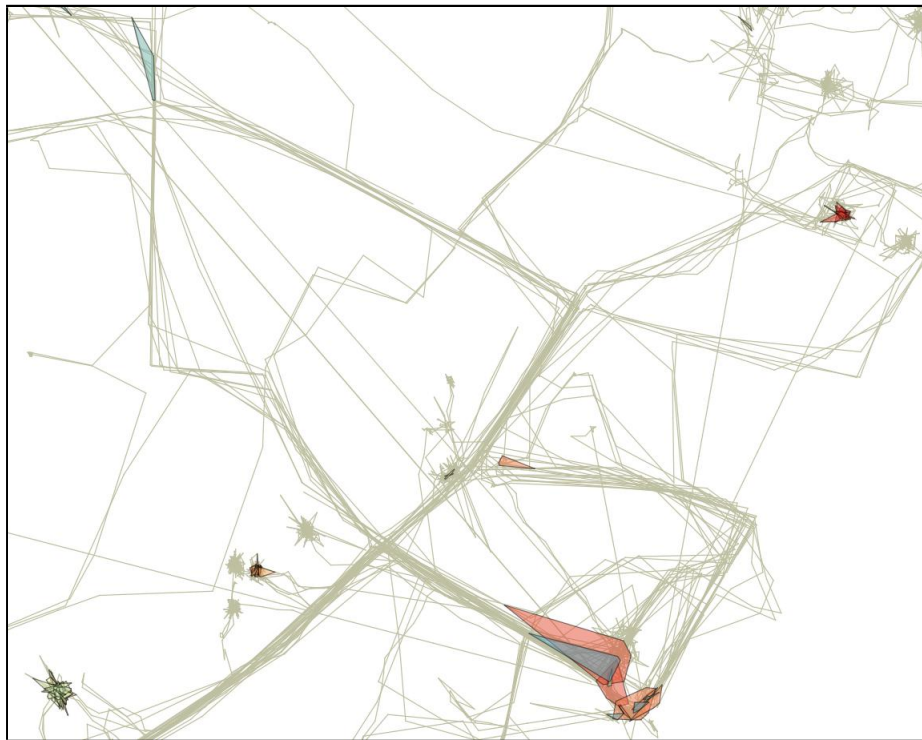
From our experiments, we concluded that our proposed data-driven algorithms, A-miner and E-miner, performed better than CS-miner in many realistic settings. Although E-miner took slightly longer to complete than A-miner, it can limit its memory needs and is suitable for larger datasets. In real-life scenarios, where few MEMOs are expected, we recommend to use FAR-miner as its fast filtering step would improve performance significantly.

4.5 Summary

In this chapter, we introduced a definition of meeting patterns called MEMO, taking account of the related information such as the meeting place and time. We developed three novel algorithms to discover closed MEMOs. Experiments on real-life datasets showed that our proposed algorithms perform better than the existing one in finding closed MEMOs.



(a)



(b)

Figure 4.8: A Comparison of (a) MEMOs Discovered by Closed MEMO Mining Algorithms in NCSU Using $m = 2$, $w = 15$ Minutes and $r = 30$ Metres and (b) Density-connected Clusters of Three-dimension GPS Points in NUCS Using $min_pts = 5$ and $\epsilon = 20$ Metres.

Mining Sub-trajectory Cliques to Find Frequent Routes

5.1 Introduction

In this chapter, we consider a Multi-object Movement Pattern, called “Sub-trajectory Clique”, from an instance of which we extract information of a “Frequent Route”. More precisely, we study the problem of extracting sub-trajectory cliques from a Trajectory Database (TJDB) in order to obtain the knowledge of frequent routes without using any prior knowledge of the underlying spatial region. We propose two novel methods to extract knowledge of the routes, which are frequently used by the moving entities in question, from a TJDB. Experiments on real-life datasets reveal that our proposed methods are efficient.

Along with the widespread use of Global Positioning Systems (GPS) and archiving movement data for various purposes, the tracks taken by the entities in question — including pedestrians’ paths, wild-animals’ tracks, navel vessels’ trajectories and taxis’ routes — are available in the TJDB. The knowledge of the routes frequently taken — or frequent routes — are embedded in the massive TJDBs. The frequent route knowledge is useful in many applications: the tried and true paths of local pedestrians can be used to suggest safe walking routes for tourists; tracks frequently used by wild-animals can help scientists better understand their movement behaviours; maritime authorities can use the frequent routes of vessels to optimize

their port operations and taxi companies can identify the trips the passengers frequently taken.

Informally, a frequent route is a path, along which the entities in question have travelled frequently. We focus on finding frequent routes without the help of road network data for several reasons. Firstly, for many applications, the tracked entities are not confined to a road network. For example, pedestrians are not restricted to walk along road segments and they often walk through buildings and open-spaces. Secondly, the underlying road network that the entities travel may not be available for use even if it exists. For instance, ecologists trying to understand wild-animals' movement do not have a complete map of routes the animals may take. Even for road-vehicles, which move on roads, it is legally not possible to use an arbitrary off-the-shelf map as many map licences limit the set of applications the licensee may use. Lastly, in addition to route recommendation, the knowledge of the instances of frequent route patterns can be used to fine-tune the existing maps by highlighting newly built roads, which the commuters take as a licensed map may not be up-to-date. By mining frequent routes, it is even possible to discover innovative routes such as using a car-park, which has two entry/exit gates and a long-enough grace period, as a short-cut from one road segment to another.

Given a Trajectory Database (TJDB) without any information of the underlying spatial space (like road networks), finding frequent routes is a challenging task since the spatial space is continuous and there is an infinite number of potential paths. Moreover, in many real-life situations, two objects using the same route (probably in different time) do not have identical sequences of locations due to differences in speed, acceleration and, more importantly, GPS sampling rates. In response to these challenge, current researches focus on grouping similar paths together and inferring a frequent route from each group. The existing solutions use two schools of techniques to perform the first grouping step. The first school [22,42]

converts the tracks logged in the TJDB into a finite sequences of regions (the set of regions is also finite) and performs sequence mining on the converted tracks while the second [37, 61] partitions the tracks into line-segments and clusters the line-segments. Each approach has its limitations. In the former, a prior knowledge of the spatial-space and/or a pre-processing step is required to divide the space into a set of regions. In addition, the end results contain sequences of regions, which have lost many subtle but important details of the frequent routes, instead of precise routes. In the latter, long frequent routes are reported as multiple line-segment clusters. In order to infer these long frequent routes, a post-processing step on the line-segment clusters is required.

Chapter Contributions

We propose to find cliques (clusters) of sub-trajectories using Fréchet distance as the similarity measure and deriving a corresponding frequent route from each sub-trajectory clique. In order to extract frequent routes from a Trajectory Database, our proposed solution does not require any prior knowledge of the road-network. Moreover, by using Fréchet distance, our solution is speed-invariant and works regardless of GPS sampling rate. It is also able to produce detailed frequent route information, while it does not require any expensive post-processing step.

Finding cliques of sub-trajectories — or sub-trajectory clusters — using Fréchet distance is a known NP-Complete problem with all its optimizations being NP-Hard and there is no polynomial approximation for approximation factor less than 2 [12]. We approach the problem from data-driven perspective by proposing an output-sensitive approximation algorithm based on the *Apriori* algorithm. In the experiments using real-life data, our proposed algorithm performs faster and more accurately than the known polynomial-time approximation algorithm proposed in [12]. In addition, we propose a divide and conquer algorithm based on our proposed

algorithm both to maintain the memory requirements under a manageable amount and to achieve parallelism.

5.2 Sub-trajectory Cliques and Frequent Routes

Before we define the problem of Mining Trajectory Databases for Frequent Routes, we will present some preliminaries.

Definition. 5.1. Trajectory of an Object — *The Trajectory of a given object o as evidenced in a trajectory database \mathcal{R} is an ordered sequence:*

$$\text{traj}(o, \mathcal{R}) = \langle t_{\text{start}}(o), \text{loc}(o, t_{\text{start}}(o), \mathcal{R}) \rangle, \dots, \langle t_{\text{end}}(o), \text{loc}(o, t_{\text{end}}(o), \mathcal{R}) \rangle,$$

where $t_{\text{start}}(o) = \min(\{t : \langle o, t, \text{loc} \rangle \in \mathcal{R}\})$, $t_{\text{end}}(o) = \max(\{t : \langle o, t, \text{loc} \rangle \in \mathcal{R}\})$, and $\text{loc}(o, t, \mathcal{R}) = \text{loc}$ such that $\langle o, t, \text{loc} \rangle \in \mathcal{R}$.

Definition 5.1 defines a trajectory of an object found in a Trajectory Database (TJDB). We will shorthand “ $\text{loc}(o, t, \mathcal{R})$ ” as $\text{loc}_o(t)$ and “ $\text{traj}(o, \mathcal{R})$ ” as traj_o , when the context is clear. Figure 5.1 and Table. 5.1 shows the visualization and records of a trajectory of a ship, s , respectively. The trajectory of object s , thus, is $\text{traj}_s = \langle t_0, (x_0, y_0) \rangle, \dots, \langle t_3, (x_3, y_3) \rangle$. Without loss of generality, in this chapter, we will assume the time is continuous and the TJDB in question has no missing record. In other words, we will assume there is a method to derive the location of objects in any real time-stamp $t \in \{ts \in \mathbb{R} | t_{\text{start}}(o) \leq ts \leq t_{\text{end}}(o)\}$.

Definition. 5.2. Sub-trajectory — *Given the trajectory of an object traj_o and two time-stamps s and e such that $t_{\text{start}}(o) \leq s \leq e \leq t_{\text{end}}(o)$, the sub-trajectory of o between s and e is the ordered sequence, i.e. $\langle s, \text{loc}_o(s) \rangle, \dots, \langle e, \text{loc}_o(e) \rangle$.*

In mathematical notation:

$$\text{sub}(s, e, \text{traj}_o) = \langle s, \text{loc}_o(s) \rangle, \dots, \langle e, \text{loc}_o(e) \rangle.$$

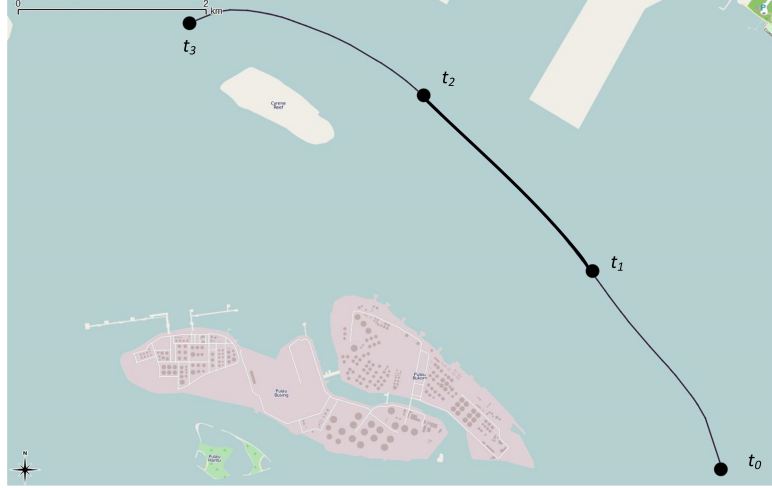


Figure 5.1: Trajectory of a Ship.

Table 5.1: Records for the Ship Trajectory in Fig. 5.1.

o	t	loc
s	t_0	(x_0, y_0)
\vdots	\vdots	\vdots
s	t_1	(x_1, y_1)
\vdots	\vdots	\vdots
s	t_2	(x_2, y_2)
\vdots	\vdots	\vdots
s	t_3	(x_3, y_3)

Definition 5.2 defines a sub-trajectory of an object between two given time-stamps. We will also use the same term “sub-trajectory” to refer to the corresponding route of a sub-trajectory (without time-information), along which the object travelled during two time-stamps defining the sub-trajectory. For instance, in Fig. 5.1, sub-trajectory of ship s between time-stamp t_1 and t_2 is $sub(t_1, t_2, traj_s) = \langle t_1, (x_1, y_1) \rangle, \dots, \langle t_2, (x_2, y_2) \rangle$ and its corresponding route is the poly-line: $(x_1, y_1), \dots, (x_2, y_2)$ (shown as a thickened section in the Fig. 5.1). Notice that a route is a poly-line which can intersect itself.

Definition. 5.3. Fréchet Distance between Two Sub-trajectories — Let $st_a = sub_a(s_a, e_a)$ and $st_b = sub_b(s_b, e_b)$ be two sub-trajectories. Also let $\mathcal{A} = \{\alpha \mid \alpha : [0, 1] \rightarrow [s_a, e_a] \text{ and } \alpha \text{ is monotone.}\}$ and $\mathcal{B} = \{\beta \mid \beta : [0, 1] \rightarrow [s_b, e_b] \text{ and } \beta \text{ is}$

monotone } be two sets of re-parametrizations.

The Fréchet distance $dist_{\text{Fr}}$ between the two sub-trajectories, sub_a and sub_b , is defined as the minimum $dist_{\text{max}}$ over \mathcal{A} and \mathcal{B} , where $dist_{\text{max}}(\alpha, \beta)$ is the maximum distance between $loc_a(\alpha(x))$ and $loc_b(\beta(x))$ for all $x \in [0, 1]$.

In mathematical notation,

$$dist_{\text{Fr}}(st_a, st_b) = \min_{\alpha \in \mathcal{A}, \beta \in \mathcal{B}}(dist_{\text{max}}(\alpha, \beta)), \text{ where}$$

$$\mathcal{A} = \{\alpha | \alpha : [0, 1] \rightarrow [s_a, e_a] \text{ and } \alpha \text{ is monotone.}\},$$

$$\mathcal{B} = \{\beta | \beta : [0, 1] \rightarrow [s_b, e_b] \text{ and } \beta \text{ is monotone.}\}, \text{ and}$$

$$dist_{\text{max}}(\alpha, \beta) = \max_{x \in [0, 1]}(f_{(\alpha, \beta)}(x)), \text{ where } f_{(\alpha, \beta)}(x) = dist(loc_a(\alpha(x)), loc_b(\beta(x))).$$

Definition 5.3 defines a distance measure between two sub-trajectories, known as Fréchet distance in the literature. A more intuitive explanation of the Fréchet distance traditionally given in the literature is that the Fréchet distance between two given routes is the minimum length of leash required for a man and his dog, each walking on a route. Each is (intelligently) trying to minimize the required leash length under a constraint that they cannot walk backward (but they may stop walking to wait for the other).

We are going to briefly discuss how the Fréchet distance of the two example routes, a and b , in Fig. 5.2(a)¹ is calculated. Let us first assume route a (b) is taken by the man (his dog). Each re-parametrization $\alpha \in \mathcal{A}$ ($\beta \in \mathcal{B}$) corresponds to how the man (his dog) chooses to walk. The Fréchet distance between the two given routes is the minimum $dist_{\text{max}}(\alpha, \beta)$ of all pairs of $(\alpha, \beta) \in \mathcal{A} \times \mathcal{B}$. Two such pairs, (α_1, β_1) and (α_2, β_2) , are given in Table. 5.2 and some of the location pairs given

¹The time-stamps in this example are arbitrary, i.e. two time-stamps with the same numerals do not necessarily equal — t_{a1} may not be equal to t_{b1} — and the time-interval between two time-stamps with the same alphabet and consecutive numerals are not necessarily the same — the interval between t_{a1} and t_{a2} may not be the same as that between t_{a2} and t_{a3}

by the two re-parametrization pairs are visualized (shown as thin dotted lines) in Fig. 5.2(b) and Fig. 5.2(c).

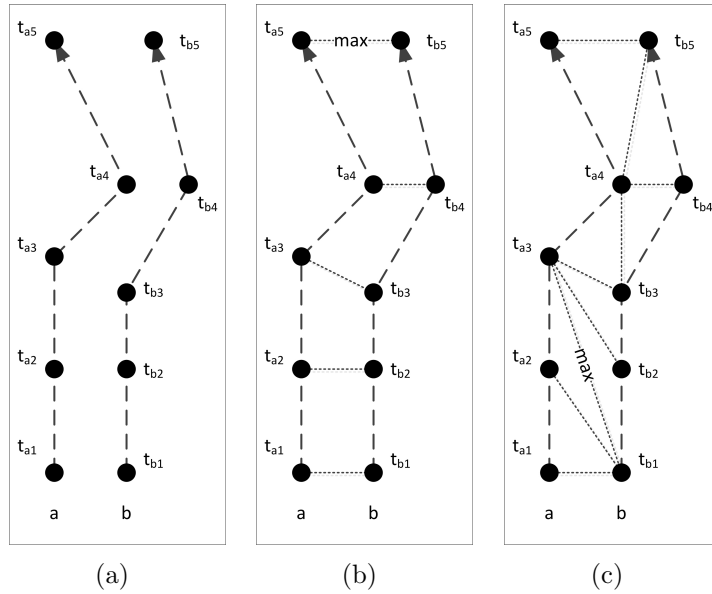


Figure 5.2: (a) Two Polygonal Curves, (b) a Pair of Possible Re-parametrizations and (c) Another Pair of Possible Re-parametrizations.

For each pair (α, β) , the **continuous** function $f_{(\alpha, \beta)} : [0, 1] \rightarrow \mathbb{R}$ is defined as $f_{(\alpha, \beta)}(x)$ is the Euclidean distance between $loc_a(\alpha(x))$ and $loc_b(\beta(x))$. The leash length required for each pair of movement is the maximum value of the image of f . For instance, $f_{(\alpha_1, \beta_1)}(0.25) = dist(loc_a(t_{a2}), loc_b(t_{b2}))$ and $f_{(\alpha_2, \beta_2)}(0.25) = dist(loc_a(t_{a3}), loc_b(t_{b1}))$. Thus, the required (maximum) leash length for the first pair, $dist_{\max}(\alpha_1, \beta_1)$, is given by $f_{(\alpha_1, \beta_1)}(1.00) = dist(loc_a(t_{a5}), loc_b(t_{b5}))$ — denoted as “max” in Fig. 5.2(b) — while that of the second re-parametrization, $dist_{\max}(\alpha_2, \beta_2)$, is by $f_{(\alpha_2, \beta_2)}(0.2) = dist(loc_a(t_{a3}), loc_b(t_{b1}))$ — denoted as “max” in Fig. 5.2(c).

Hence, we can say that, in the first pair of re-parametrizations (see Fig. 5.2(b)), the man and his dog walk in a more synchronized manner requiring shorter (maximum) leash length than the second pair (see Fig. 5.2(c)), in which the man has walked (on route a) far before his dog begins to move (on route b). The Fréchet

Table 5.2: Two Pairs of Re-parametrizations of the Two Sub-trajectories Shown in Fig. 5.2(a).

Re-parametrization Pair 1			Re-parametrization Pair 2		
x	$\alpha_1(x)$	$\beta_1(x)$	x	$\alpha_2(x)$	$\beta_2(x)$
0.00	t_{a1}	t_{b1}	0.00	t_{a1}	t_{b1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.01	t_{a1}	t_{b1}	0.01	t_{a2}	t_{b1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.02	t_{a1}	t_{b1}	0.02	t_{a3}	t_{b1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.25	t_{a2}	t_{b2}	0.25	t_{a3}	t_{b1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.26	t_{a2}	t_{b2}	0.26	t_{a3}	t_{b2}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.50	t_{a3}	t_{b3}	0.50	t_{a3}	t_{b3}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.51	t_{a3}	t_{b3}	0.51	t_{a4}	t_{b3}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.75	t_{a4}	t_{b4}	0.75	t_{a4}	t_{b4}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0.76	t_{a4}	t_{b4}	0.76	t_{a4}	t_{b5}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
1.00	t_{a5}	t_{b5}	1.00	t_{a5}	t_{b5}

distance is the shortest among the (maximum) leash lengths required for all manners of progressing the man and his dog can make on their routes (including those depicted in this example).

In essence, Fréchet distance considers only the direction (the man and his dog are not allowed to walk back) and the shape of the curves (it chooses the minimum among the leashes for all ways of progression they can make). Yet, it neither considers the speed the objects move (each parametrization represent a different way of walking) nor the number of sampled points (the re-parametrizations are defined as functions to a continuous range), i.e. the Fréchet distance between two curves in the above example is the same even if we remove the sample point at t_{a2} and put

multiple sample points between t_{b_1} and t_{b_2} by linear interpolation. Therefore, we choose the Fréchet distance to measure the similarity between sub-trajectories².

Definition. 5.4. Sub-trajectory Clique — Consider a Trajectory Database \mathcal{R} , which contains the trajectories of a set of objects O . Given parameters m , l , and r , a set of sub-trajectories J forms a sub-**trajectory clique** (TRAJCLIQ) if (i) each $j \in J$ is within r Fréchet distance away from all sub-trajectories $j' \in J$, (ii) the lengths of all sub-trajectories $j \in J$ are longer than l , and (iii) J contains at least m non-identical sub-trajectories.

Definition 5.4 defines a sub-trajectory clique (TRAJCLIQ) which includes m unique sub-trajectories, which are at least l units in spatial length (not time duration). We choose the Fréchet distance, defined in Def. 5.3, to measure the similarity between routes because Fréchet distance ensures sub-trajectories in the same clique are spatially close, similar in shape, and similar in direction. Therefore, all nearby sub-trajectories of similar shape and direction to a sub-trajectory will belong to the same TRAJCLIQ even though the corresponding movements were taken at different speeds during different time-spans. In other words, a TRAJCLIQ groups the sub-trajectories, which are on the same route and, hence, a track-clique containing m sub-trajectories corresponds to a frequent route taken at least m times.

Figure 5.3 illustrates a TJDB containing trajectories of four objects, a , b , c , and d . For $m = 3$, the sub-trajectory of a from t_{a1} to t_{a4} , another sub-trajectory of a from t_{a7} to t_{a10} , and the sub-trajectory of b from t_{b1} to t_{b4} form a Sub-trajectory clique (TRAJCLIQ) as the three sub-trajectories are within a Fréchet distance of r . Note that two sub-trajectories of a single object, i.e. object a , can involve in the same TRAJCLIQ. However, the sub-trajectory of a from t_{a5} to t_{a6} , that of c from t_{c1} to t_{c2} and that of d from t_{d1} to t_{d2} do not form a TRAJCLIQ because, although the

²For more information of Fréchet distance, refer to the Section 3.2.7

sub-trajectory of a is within a Fréchet distance r from those of c and d , the Fréchet distance between the sub-trajectories of c and d is more than r .

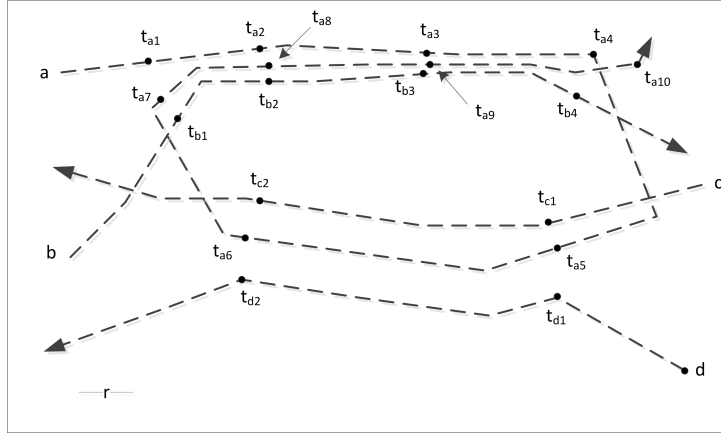


Figure 5.3: A Visualization of a Trajectory Database Containing Four Trajectories.

Definition. 5.5. Frequent Route — A sub-trajectory clique is closed if and only if there is no other sub-trajectory clique covering it. The longest sub-trajectory in a closed sub-trajectory clique is defined as the corresponding **Frequent Route (FreqRo)** of the closed sub-trajectory clique.

Definition 5.5 defines a Frequent Route as the longest sub-trajectory (spatial length, not time duration) of a closed TRAJCLIQ. A route, which is frequently used, derived from a non-closed sub-trajectory is of little interest to the users since its information is conveyed in the frequent route derived from a closed TRAJCLIQ. Continuing with our example in Fig. 5.2, the TRAJCLIQ formed by $sub_a(t_{a2}, t_{a3})$, $sub_a(t_{a8}, t_{a9})$, and $sub_b(t_{b2}, t_{b3})$ is covered by a larger TRAJCLIQ formed by $sub_a(t_{a1}, t_{a4})$, $sub_a(t_{a7}, t_{a10})$, and $sub_b(t_{b1}, t_{b4})$. Hence the corresponding route of the former is covered by the frequent route derived from the latter TRAJCLIQ.

Definition. 5.6. Mining Frequent Routes — Given a Trajectory Database \mathcal{R} and parameters, $m > 1$, $r > 0$, and $l > 0$, Mining Frequent Routes is to find all closed sub-trajectory cliques and derive a Frequent Route for each sub-trajectory clique found.

Definition 5.6 defines the problem to discover Frequent Routes from a TJDB as a two-step problem, finding sub-trajectory cliques, each of which including movement records of the entities taking the same route for at least m times, and deriving frequent routes from the cliques.

5.3 Methods to Mine Sub-trajectory Cliques to Extract Frequent Routes

In this section, we will first prove that mining all closed sub-trajectory cliques (TRAJCLIQS) is NP-Complete. Then, we will describe an exact algorithm to discover all closed TRAJCLIQS and extract a corresponding frequent route from each TRAJCLIQ. The exact algorithm is based on the *apriori*-properties of the TRAJCLIQS. We will proceed to discuss why such an algorithm is not feasible in reality and present two new algorithms to approximate TRAJCLIQS from a given Trajectory Database (TJDB). The first approximation algorithm is also based on the *apriori* algorithm and is able to quickly approximate TRAJCLIQS at an approximation factor of 2. The second algorithm divides the TJDB into distinct subsets so that an instance of the first algorithm can be applied to each subset, reducing memory requirement and/or enabling parallel processing.

5.3.1 Hardness of Mining Sub-trajectory Cliques from a Trajectory Database

A proof, which proves a problem similar to mining all closed TRAJCLIQS is NP-Complete and from which we inspire our following proof, is given in [12].

Theorem. 5.1. *Given a Trajectory Database \mathcal{R} and parameters, m , l , and r , answering if there exist a TRAJCLIQ is NP-Complete, i.e. any of its solutions can be verified in polynomial time but finding one cannot be unless $P = NP$.*

Proof. The first portion of the theorem is proved as follow: Verifying whether the Fréchet distance between a given set of sub-trajectories in a dataset is within r can be performed in $O(n_1.n_2.\cdots.n_m)$ time, where n_i ($1 \leq i \leq m$) is bounded by the size of the Trajectory Database (TJDB) [18]. The number of sub-trajectories and their lengths can also be verifiable in time polynomial in terms of the TJDB's size. Therefore, a solution to the query whether there exist a sub-trajectory cluster (TRAJCLIQ) in a given TJDB and parameters can be verifiable in polynomial time in terms of the size of the TJDB.

The second portion of the theorem is proved by presenting a solution of the problem of MAXCLIQUE, which uses finding TRAJCLIQ as an atomic step. The MAXCLIQUE problem is a known NP-Complete problem. In MAXCLIQUE, the input is a graph $G = (V, E)$ with n vertices v_1, \dots, v_n and E is a subset of the Cartesian product set of $V \times V$. Given a parameter m , the output is to answer whether there is a complete sub-graph (clique) C , i.e. $C \subseteq V$, $|C| = m$, and $C \times C \subseteq E$.

We can transform an instance of MAXCLIQUE problem into that of a TRAJCLIQ using the following steps. Each vertex v_i in V will be represented by a trajectory $traj_i$, which is defined as a poly-line formed by connecting the following $3n + 1$ points (in ascending order of the x coordinate) :

- $(0, 0)$,
- for all $1 \leq j < i$,
 - if $(v_i, v_j) \in E$, $(3j - 2, 0)$ and $(3j - 1, 0)$,
 - else $(3j - 2, -1)$ and $(3j - 1, -1)$,
 - $(3j, 0)$
- $(3i - 2, +1)$, $(3i - 1, +1)$, and $(3i, 0)$
- for all $i < j \leq n$,

- if $(v_i, v_j) \in E$, $(3j - 2, 0)$ and $(3j - 1, 0)$,
 else $(3j - 2, -1)$ and $(3j - 1, -1)$,
- $(3j, 0)$

In essence, if there is not an edge between the vertices v_i and v_j in G , the corresponding trajectories $traj_i$ and $traj_j$ separate themselves away from each other in the region where their x coordinates are between $3i - 2$ and $3i - 1$. On the other hand, the trajectories are within a distance of 1 if the vertices they represent have an edge. In other words, the Fréchet distance between two sub-trajectories $traj_i$ and $traj_j$ will be 1 if and only if there exists an edge between v_i and v_j in G . In Fig. 5.4, there are edges between v_1 and v_2 , v_1 and v_3 , and v_1 and v_4 . Hence, the trajectory representing v_1 stays at $y = 0$, when x is in $[4, 5]$, $[7, 8]$ and $[10, 11]$, which are highlighted using grey regions. Notice that there is no edge between v_1 and v_5 in the graph and the trajectory of v_1 moves lower in the y coordinate (away from that of v_5 , which moves higher in the y coordinate) when x is in $[13, 14]$.

Therefore, we can answer the query whether there is a clique of size m by transforming the graph G into a TJDB as stated above and checking if there is a TRAJCLIQ for parameters m , $l = 3n + 1$, and $r = 1$.

□

5.3.2 Apriori-based Frequent Route Miner

Since mining all closed TRAJCLIQ is an NP-Complete problem, we are going to analyze the feasibility of using an *Apriori*-based data-driven algorithm to discover closed TRAJCLIQS as intermediate results and extract Frequent Routes from them. First, we will present the *apriori*-properties of TRAJCLIQS.

Lemma. 5.1. *Suppose a set of sub-trajectories $J = \{sub_1, \dots, sub_p\}$ and its subset*

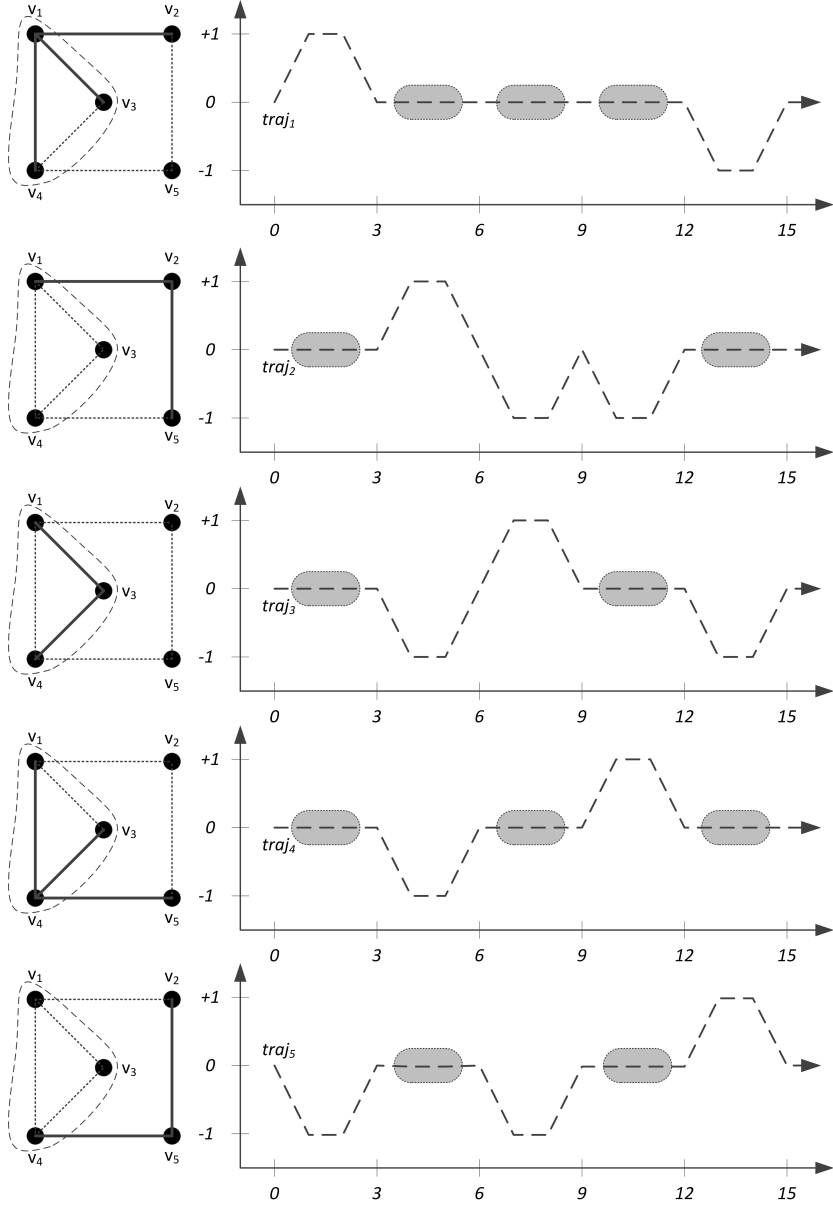


Figure 5.4: Conversion of a MAXCLIQUE problem into TRAJCLIQ problem.

$J' = \{sub'_1, \dots, sub'_q\}$ ($p \geq q$). If there is a re-parametrization α_i for each sub-trajectory $sub_i \in J$ such that for any $i, j \in \{1, \dots, p\}$, $dist_{\max}(\alpha_i, \alpha_j) \leq r$, then there is a re-parametrization α'_k for each sub-trajectory $sub'_k \in J'$ such that for any $i, j \in \{1, \dots, q\}$, $dist_{\max}(\alpha'_i, \alpha'_j) \leq r$.

Proof. For a pair of sub-trajectories $sub'_i, sub'_j \in J'$, suppose there is no pair of re-parametrizations α'_i and α'_j , which gives the $dist_{\max}(\alpha'_i, \alpha'_j) \leq r$. By definition, (although not necessarily unique) there is a pair of re-parametrizations that gives

minimum $d = \text{dist}_{\max}(\alpha'_i, \alpha'_j)$ — i.e. $r < d$.

However, since $J' \subseteq J$, $\text{sub}'_i \in J$ and $\text{sub}'_j \in J$. Therefore, α_i and α_j for sub-trajectories sub'_i and sub'_j gives $\text{dist}_{\max}(\alpha_i, \alpha_j) \leq r < d$ (it contradicts with $d = \text{dist}_{\max}(\alpha'_i, \alpha'_j)$ is minimum).

□

Using Lemma 5.1, we can derive the *apriori*-properties of TRAJCLIQs as follows: if a set of sub-trajectories $J = \{\text{sub}_1, \dots, \text{sub}_p\}$ containing p sub-trajectories forms a TRAJCLIQ, there is a re-parametrization α_i for each $\text{sub}_i \in J$, which gives $\text{dist}_{\max(\alpha_i, \alpha_j)} \leq r$ for any $\text{sub}_i, \text{sub}_j \in J$. Following Lemma 5.1, there is also (at least) a re-parametrization α'_i for each $\text{sub}_i \in J' \subseteq J$, which gives $\text{dist}_{\max(\alpha'_i, \alpha'_j)} \leq r$ for any $i, j \in J'$. Therefore, all $J' \subseteq J$ also forms a TRAJCLIQ. In other words, J does not form a TRAJCLIQ if any of its subset does not.

The *Apriori*-based Frequent Route Miner (A-0), adapted from the *Apriori*-algorithm in [3], exploits the *apriori*-properties of the TRAJCLIQs to systematically discover the TRAJCLIQs formed by $(k + 1)$ sub-trajectories only when those formed by its subsets exist. An outline of the A-0 is given in Algorithm 5.1. The function $\text{Closed-CLIQ}(\mathcal{R}, l, r, O_k)$ returns list of TRAJCLIQs, which may or may not be closed, formed the k sub-trajectories of the objects in O_k .

The algorithm A-0 first initializes the Clique-list containing the TRAJCLIQs formed by sub-trajectories of all possible pairs of objects (lines 2 - 5). Starting with $k = 2$, the Clique-list of TRAJCLIQs formed by $k + 1$ sub-trajectories of a list of objects (O_{k+1}) are built only when those of its sub-lists (O_k and O'_k) sharing the same k -prefix form TRAJCLIQs (lines 6 - 15). In doing so, if $k \geq m$, then the TRAJCLIQs in the Clique-list, which are formed by the k sub-trajectories of the current Object-list (O_k), are potential closed TRAJCLIQ. Thus, they are put into the set of TRAJCLIQs \mathcal{U} (lines 7 - 9), which is later filtered to remove non-closed

Algorithm 5.1 Apriori-based Frequent Route Miner (A-0).

Input: \mathcal{R} , r , m and l .

Output: A set of frequent routes \mathcal{F} .

```
1: The set of object-lists  $\mathcal{C}_1 \leftarrow \emptyset$ ,  $\mathcal{U} \leftarrow \emptyset$ ,  $\mathcal{F} \leftarrow \emptyset$ , and  $k \leftarrow 2$ 
2: for all  $(o, o') \in O \times O$  do
3:   Object-list  $O_2 \leftarrow [o, o']$  and Clique-list  $L(O_2) \leftarrow \text{Closed-CLIQ}(\mathcal{R}, l, r, O_2)$ 
4:   if  $L(O_2)$  is not empty then
5:      $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup \{O_2\}$ 
6: while  $\mathcal{C}_k \neq \emptyset$  do
7:   for all  $O_k \in \mathcal{C}_k$  do
8:     if  $k \geq m$  then
9:        $\mathcal{U} \leftarrow \mathcal{U} \cup L(O_k)$ 
10:  The set of Object-lists  $\mathcal{C}_{k+1} \leftarrow \emptyset$ 
11:  for all  $O_k, O'_k \in \mathcal{C}_k$  such that  $k\text{-prefix}(O_k) = k\text{-prefix}(O'_k)$  do
12:     $O_{k+1} \leftarrow \text{append}(O_k, \text{last}(O'_k))$  and  $L(O_{k+1}) \leftarrow \text{Closed-CLIQ}(\mathcal{R}, l, r, O_{k+1})$ 
13:    if  $L(O_{k+1})$  is not empty then
14:       $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_{k+1} \cup \{O_{k+1}\}$ 
15:   $k \leftarrow k + 1$ 
16:  $\mathcal{U} \leftarrow \mathcal{U} - \{U \mid U \text{ is not a closed-TRAJCLIQ}\}$ .
17: for all  $Q \in \mathcal{U}$  do
18:   $\mathcal{F} \leftarrow \mathcal{F} \cup \{\text{Get-Frequent-Route}(U)\}$ 
```

TRAJCLIQs (line 16). As the last step, the algorithm A-0 extracts a frequent route from each closed TRAJCLIQ in \mathcal{U} (lines 17 - 18).

In order to allow a TRAJCLIQ to contain two (different) sub-trajectories of the same object, unlike other *Apriori*-based algorithms, A-0 uses a list notation for objects, which may contain an object multiple times. Figure 5.5 shows a portion of the search space the algorithm A-0 traverses to find the TRAJCLIQs in the data in Fig. 5.3 for given parameters r , l , and $m = 3$. A-0 starts with finding TRAJCLIQ formed by two sub-trajectories, beginning with the TRAJCLIQs formed by sub-trajectories of the Object-list $[aa]$. Since sub-trajectories $sub_a(t_{a1}, t_{a4})$ and $sub_a(t_{a7}, t_{a10})$ forms a TRAJCLIQ, it is stored in the Clique-list $L([aa])$. It continues to find TRAJCLIQs formed by all the Object-lists containing two objects with prefix $[a]$, i.e. $[ab]$, $[ac]$, and $[ad]$. Then A-0 tries to find TRAJCLIQs formed by all Object-lists containing two objects with prefixes $[b]$, $[c]$, and $[d]$, which do not exist in the TJDB depicted in Fig. 5.3. The first row ($k = 2$) of the Table 5.3 lists the state of the variables after

this initialization step. In the next step ($k = 3$), A-0 tries to find the TRAJCLIQS formed by sub-trajectories of $[aaa]$, $[aab]$, $[aac]$ and $[aad]$ since their subsets are in \mathcal{C}_2 . It only finds a TRAJCLIQ formed by sub-trajectories of Object-list $[aab]$ and put $[aab]$ into \mathcal{C}_3 (see the second row in Tab. 5.3). Since $k \geq m$, $L([aab])$ is put into the result set \mathcal{U} . The algorithm exits the loop at $k = 4$ as \mathcal{C}_4 becomes empty and output the frequent route extracted from $L([aab])$.

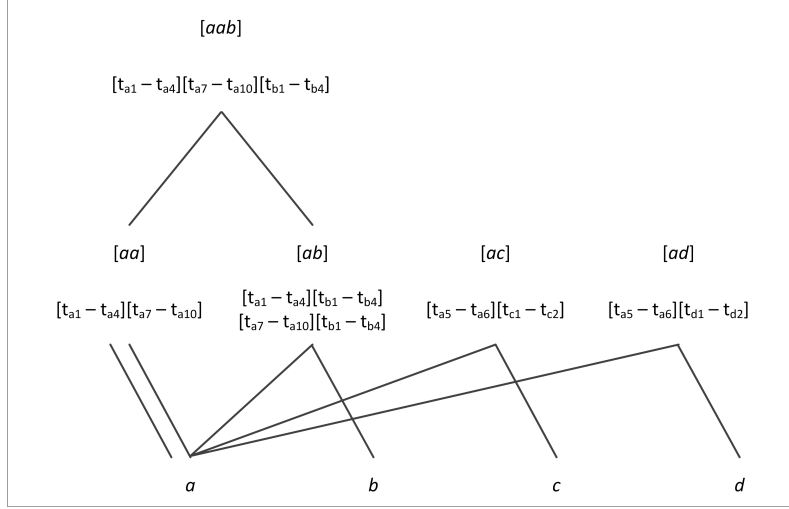


Figure 5.5: How Algorithm A-0 Finds Frequent Routes from the Trajectories Depicted in Fig. 5.3.

Table 5.3: A Trace of A-0 Running on the Trajectory Database Shown in Fig. 5.3

k	\mathcal{C}_k	$L(O_k)$	\mathcal{U}
2	$[aa]$	$\langle sub_a(t_{a1}, t_{a4}), sub_a(t_{a7}, t_{a10}) \rangle$	—
	$[ab]$	$\langle sub_a(t_{a1}, t_{a4}), sub_b(t_{b1}, t_{b4}) \rangle,$ $\langle sub_a(t_{a7}, t_{a10}), sub_b(t_{b1}, t_{b4}) \rangle$	—
	$[ac]$	$\langle sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}) \rangle$	—
	$[ad]$	$\langle sub_a(t_{a5}, t_{a6}), sub_d(t_{d1}, t_{d2}) \rangle$	—
3	$[aab]$	$\langle sub_a(t_{a1}, t_{a4}), sub_a(t_{a7}, t_{a10}), sub_b(t_{b1}, t_{b4}) \rangle$	$L([aab])$
4	ϕ	ϕ	$L([aab])$

Finding Sub-trajectory Cliques of k Sub-trajectories.

In algorithm A-0, lines 3 and 12 call a sub-routine (Closed-CLIQ) in order to find all sub-trajectory cliques composed of k sub-trajectories. Closed-CLIQ uses Free-space (defined below) to extract the sub-trajectory cliques.

Definition. 5.7. Free-space — Given a distance threshold r and two sub-trajectories sub_i and sub_j containing p and q segments respectively, their corresponding Free-space is the set $F(i, j) = \{(x, y) \in [0, p] \times [0, q] : dist(loc_i(x), loc_j(y)) \leq r\}$.

Definition 5.7 defines a free-space as a two-dimensional map that maintain which parts of the two sub-trajectories in questions are within distance r . Figure 5.6 illustrates a Free-space (cell) of two sub-trajectories i and j containing one segment each and a distance r as the white region. The two points marked by black rectangles in the Fig. 5.6 is closer than the distance r . Therefore the point in the Free-space that represents these two points of the trajectories falls in the white region while that of two points marked by black circles, in the grey region. For sub-trajectories containing p and q segments, their Free-space for a given r is a two-dimensional array of such Free-space cells.

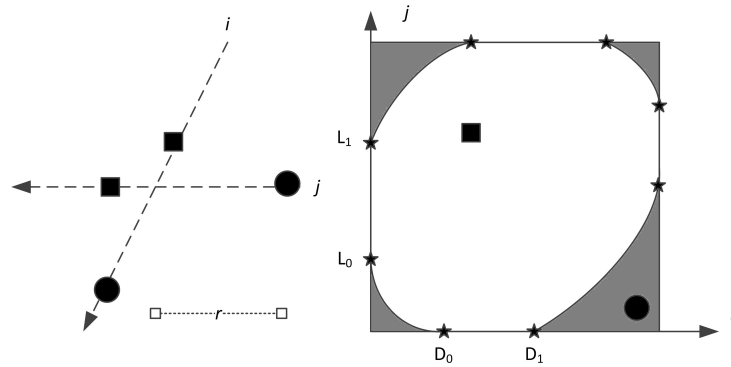


Figure 5.6: Two Trajectory Segments and Their Corresponding Free-space Cell.

For two-dimensional GPS trajectories, the white region of a Free-space cell is always an intersect of an eclipse and a rectangle [5]. The white region, therefore, is defined by eight points called critical points (illustrated as black stars in Fig. 5.6). Adjacent Free-space cells share the critical points between them. For example, the critical points on the right boundary of the Free-space cell at $(x - 1, y)$ is the same as those on the left of Free-space cell at (x, y) . Given a Free-space cell c at (x, y) , we will denote its two critical points on the left side as $L_0(x, y)$ and $L_1(x, y)$ while

those on the down side as $D_0(x, y)$ and D_1 .

Alt and Godau [5] proved that the Frèchet distance between two sub-trajectories is less than r if and only if there is a monotone path in their Free-space. We will denote a path in the Free-space as $\langle (x_{\text{start}}, y_{\text{start}}), (x_{\text{end}}, y_{\text{end}}) \rangle$. For $k = 2$, the sub-routine Closed-CLIQ (see Algorithm 5.2) finds all monotone paths in the Free-space of the trajectories of two given objects in O_k . For simplicity, the outline describes how to find monotone paths starting and ending at critical points.

Algorithm 5.2 Sub-routine Closed-CLIQ for $k = 2$ Used in the Apriori-based Frequent Route Miner (A-0).

Input: $traj_i, traj_j, O_2 = [o_i, o_j], r$ and l
Output: All Closed TRAJCLIQS $L(O_2)$.

- 1: Set $p \leftarrow \text{len}(traj_i)$ and $q \leftarrow \text{len}(traj_j)$.
- 2: **for all** $(x, y) \in [0, p] \times [0, q]$ **do**
- 3: Calculate $L_0(x, y), L_1(x, y), D_0(x, y)$, and $D_1(x, y)$ for $traj_i$ and $traj_j$
- 4: Set of Monotone Paths $Left(x, y) \leftarrow \{ \langle L_0(x, y), L_0(x, y) \rangle \}$
- 5: Set of Monotone Paths $Down(x, y) \leftarrow \{ \langle D_0(x, y), D_0(x, y) \rangle \}$
- 6: Set of Monotone Paths $Mono \leftarrow \phi$ and Closed TRAJCLIQS $L(O_2) \leftarrow \phi$
- 7: **for all** $x = 0$ to $p - 1$ **do**
- 8: **for all** $y = 0$ to $q - 1$ **do**
- 9: **for all** Path $p \in Left(x, y) \cup Down(x, y)$ s.t. p ends at (x_e, y_e) **do**
- 10: **if** $\text{append}(p, \langle (x_e, y_e), L_1(x + 1, y) \rangle)$ is monotone **then**
- 11: $Mono \leftarrow Mono \cup \{ \text{append}(p, \langle (x_e, y_e), L_1(x + 1, y) \rangle) \}$
- 12: $p \leftarrow \text{append}(path, \langle (x_e, y_e), \text{selectGrY}(L_0(x + 1, y), (x_e, y_e)) \rangle)$
- 13: **for all** Path $p \in Left(x, y) \cup Down(x, y)$ s.t. p ends at (x_e, y_e) **do**
- 14: **if** $\text{append}(p, \langle (x_e, y_e), D_1(x, y + 1) \rangle)$ is monotone **then**
- 15: $Mono \leftarrow Mono \cup \{ \text{append}(p, \langle (x_e, y_e), D_1(x, y + 1) \rangle) \}$
- 16: $p \leftarrow \text{append}(path, \langle (x_e, y_e), \text{selectGrX}(D_0(x, y + 1), (x_e, y_e)) \rangle)$
- 17: **for all** $M \in Mono$ **do**
- 18: TRAJCLIQ $tc \leftarrow \text{Extract-Cliq}(M)$
- 19: **if** tc is closed and $\text{len}(sub) \geq l$ for all $sub \in tc$ **then**
- 20: $L(O_2) \leftarrow L(O_2) \cup \{tc\}$

In Closed-CLIQ outlined in Algorithm 5.2, for each Free-space cell at (x, y) , $Left(x, y)$ and $Down(x, y)$ maintains the monotone paths started in cells at the left or lower sides of (x, y) and reachable to its left and lower sides. The algorithm Closed-CLIQ first calculates the critical points of the Free-space and the monotone path sets, $Left(x, y)$ and $Down(x, y)$, with monotone paths of length zero (lines

2 - 5). Starting with the lowest-leftmost cell located at $(0,0)$, it systematically propagates the monotone paths in the cell at (x,y) to the cells above and right while maintaining the resulting monotone paths in *Mono* (lines 7 - 16). For each monotone path in *Mono*, a corresponding set of sub-trajectories tc having their Fréchet distance within r is extracted and, if tc is close and all its sub-trajectories are not shorter than l , it is returned (lines 17 - 20).

The algorithm described in Algorithm 5.2 can be easily extended for any $k \geq 2$ by extracting all monotone curves (monotone in all coordinates) in a k dimensional-space. However, for $k \geq 3$, despite being polynomial, this process is not feasible for real-life datasets containing hundreds of trajectories with thousands of segments. Actually, the task of “answering the query whether the Fréchet distance between any pair in a given set of k sub-trajectories is not more than r ” is proven to be $O(n_1.n_2.\dots.n_k)$ in [18]. In addition, the generalized process needs $k(k-1)/2$ free-spaces to be kept in the memory for all $k \geq 2$. Therefore, we turn our focus to approximation algorithms, which we present next.

5.3.3 Approximation of Sub-trajectory Cliques for Frequent Route Mining

We observe that, by definition, if a set of sub-trajectories $J = \{sub_1, sub_2, \dots, sub_p\}$ forms a TRAJCLIQ, the Fréchet distance between the first sub-trajectory, sub_1 , and other sub-trajectories, sub_2, \dots, sub_p is at most r (although the reverse is not always true). We will use this observation to approximate TRAJCLIQs formed by $k \geq 3$ sub-trajectories. We will denote the first³ sub-trajectory in a set of sub-trajectories as the “reference sub-trajectory” (or simply “reference trajectory” if context is clear) of the TRAJCLIQ it forms. The following Lemma proves that the approximation factor of our proposed solution is 2.

³We can use an arbitrary order among the sub-trajectories to define the first.

Lemma. 5.2. *Suppose there are three sub-trajectories, sub_a , sub_b , and sub_c . If both $dist_{Fr}(sub_a, sub_b) \leq r$ and $dist_{Fr}(sub_a, sub_c) \leq r$ hold, then it follows that $dist_{Fr}(sub_b, sub_c) \leq 2r$.*

Proof. The Fréchet distance complies triangle inequality [17], i.e.

$$dist_{Fr}(sub_b, sub_c) \leq dist_{Fr}(sub_a, sub_b) + dist_{Fr}(sub_a, sub_c).$$

Since $dist_{Fr}(sub_a, sub_b) + dist_{Fr}(sub_a, sub_c) \leq 2r$, $dist_{Fr}(sub_b, sub_c) \leq 2r$.

□

The approximation algorithm we develop, namely *Apriori*-based Approximate Frequent Route Miner (A-1), is essentially following the same steps of A-0 described in Algorithm 5.1. The only difference between the exact algorithm (A-0) and the approximate one (A-1) is that A-1 uses an approximation sub-routine instead of the exact Closed-CLIQ in line 13. For the trajectories in Fig. 5.3, A-1 finds the TRAJCLIQs of Object-list containing two objects (the same as A-0) using the exact algorithm given in Algorithm 5.2. Then, for $k = 3$, it tries to approximate TRAJCLIQs of sub-trajectories in Object-lists, whose subsets are found in O_2 . A-1 finds two (approximate) TRAJCLIQs, $J_1 = \{sub_a(t_{a1}, t_{a4}), sub_a(t_{a7}, t_{a10}), sub_b(t_{b1}, t_{b4})\}$ and $J_2 = \{sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}), sub_d(t_{d1}, t_{d2})\}$ using $sub_a(t_{a1}, t_{a4})$ and $sub_a(t_{a5}, t_{a6})$ as their reference trajectories respectively. It does not find any TRAJCLIQs for $k = 4$. Detailed states of the variables through the above steps are given in Table 5.4.

Table 5.4: A Trace of A-1 Running on the Trajectory Database Shown in Fig. 5.3

k	\mathcal{C}_k	$L(O_k)$	\mathcal{U}
2	[aa]	$\langle sub_a(t_{a1}, t_{a4}), sub_a(t_{a7}, t_{a10}) \rangle$	—
	[ab]	$\langle sub_a(t_{a1}, t_{a4}), sub_b(t_{b1}, t_{b4}) \rangle,$ $\langle sub_a(t_{a7}, t_{a10}), sub_b(t_{b1}, t_{b4}) \rangle$	—
	[ac]	$\langle sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}) \rangle$	—
	[ad]	$\langle sub_a(t_{a5}, t_{a6}), sub_d(t_{d1}, t_{d2}) \rangle$	—
3	[aab]	$\langle sub_a(t_{a1}, t_{a4}), sub_a(t_{a7}, t_{a10}), sub_b(t_{b1}, t_{b4}) \rangle$	$L([aab])$
	[acd]	$\langle sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}), sub_d(t_{d1}, t_{d2}) \rangle$	$L([acd])$
4	ϕ	ϕ	$L([aab]), L([acd])$

Optimization to Approximation of Sub-trajectory Cliques.

In our implementation, we optimize the approximation step in algorithm A-1 using the *a priori*-properties of TRAJCLIQs, which states that a set of sub-trajectories does not form a TRAJCLIQ if any of its subsets does not. Therefore, we use the information of TRAJCLIQs found in O_k and O'_k in building the TRAJCLIQs in O_{k+1} . For example, in building TRAJCLIQs formed by Object-list $[aab]$, we take clues from $L([aa])$ and $L([ab])$ that $sub_a(t_{a1}, t_{a4})$, $sub_a(t_{a7}, t_{a10})$ and $sub_b(t_{b1}, t_{b4})$ formed TRAJCLIQs of two sub-trajectories and build TRAJCLIQs formed by (portions of) these sub-trajectories. We perform this by sorting the TRAJCLIQs in $L(O_k)$ and $L(O'_k)$ by the starting times of their reference trajectories and performing a modified version of sort-merge-join over them.

Reducing the Number of False-Positives.

Since the Fréchet distance between the sub-trajectory $sub_c(t_{c1}, t_{c2})$ and the sub-trajectory $sub_d(t_{d1}, t_{d2})$ is larger than r , the set of sub-trajectories J_2 that our algorithm A-1 approximates as a TRAJCLIQ is a false positive the approximation introduces. We further exploit the *a priori*-properties of TRAJCLIQs to reduce the number of false positives in the approximation results. When the approximation sub-routine builds the TRAJCLIQs formed by sub-trajectories $\{s_1, s_2, \dots, s_{k+1}\}$ of $O_{k+1} = [o_1, o_2, o_3, \dots, o_{k+1}]$, it also checks whether $\{o_2, o_3, \dots, o_{k+1}\}$ also forms TRAJCLIQs. This simple check prunes false positives as illustrated in the following example. Consider the sub-trajectories $s_a = sub_a(t_{a5}, t_{a6})$, $s_b = sub_c(t_{c1}, t_{c2})$, and $s_d = sub_d(t_{d1}, t_{d2})$ in Fig. 5.3. Since the Fréchet distance between s_a and s_c and that between s_a and s_d are not more than r , they form TRAJCLIQs of two sub-trajectories — $L(O_2)$ contains $\langle sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}) \rangle$ and $\langle sub_a(t_{a5}, t_{a6}), sub_d(t_{d1}, t_{d2}) \rangle$ (refer to Tab. 5.4). However, the Fréchet distance between s_c and s_d is larger than r

and $\langle sub_c(t_{c1}, t_{c2}), sub_d(t_{d1}, t_{d2}) \rangle \notin L(O_2)$. Therefore, before the approximation sub-routine builds the sub-trajectory cluster $\langle sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}), sub_d(t_{d1}, t_{d2}) \rangle$, it checks whether $\langle sub_c(t_{c1}, t_{c2}), sub_d(t_{d1}, t_{d2}) \rangle$ exists in $L(O_2)$ and, since it does not exist, the approximation sub-routine simply prunes the TRAJCLIQS $\langle sub_a(t_{a5}, t_{a6}), sub_c(t_{c1}, t_{c2}), sub_d(t_{d1}, t_{d2}) \rangle$.

This pruning mechanism may not be able to remove all false-positives the approximation of TRAJCLIQS introduces since A-1 is still approximating a TRAJCLIQ by checking only the Fréchet distances between the reference trajectory and the other sub-trajectories in a candidate sub-trajectory clique. The approximation factor also remains at 2. However, we expect A-1 to have fewer false-positives in its results compared to other approximation algorithms that has no pruning mechanism at all.

5.3.4 A Divide and Conquer Scheme for Scalable Approximation of Sub-trajectory Cliques

Although the *Apriori*-based Approximate Frequent Route Miner (A-1) runs fast for most real-life datasets and prunes many false positives, it still needs a large amount of memory both for the *Apriori* process and the Free-space it needs to calculate and store for each approximation call. For larger datasets and real-life computing settings, in which the available main memory is limited, this memory requirement may become a big challenge. Therefore, we devise a Divide and Conquer Scheme to mitigate the memory requirement issue of A-1.

We observe that, given an arbitrary bounding box B , all sub-trajectories it confines (sub-trajectories are completely within it) cannot have a Fréchet distance less than or equal to r to those sub-trajectories not confined by the bounded box B' , which is B extended by r on all sides. For instance, in Fig. 5.7, the whole trajectory of a is in the shaded bounding box. Therefore, it cannot have its Fréchet

distance with the sub-trajectories of x (such as $sub_x(t_{x2}, t_{x4})$ and $sub_x(t_{x3}, t_{x5})$) that goes beyond the extended bounding box (outer bounding box) shown by the thick borders. In other words, sub-trajectories of a cannot form a TRAJCLIQ with sub-trajectories of x containing portions out of the outer bounding box.

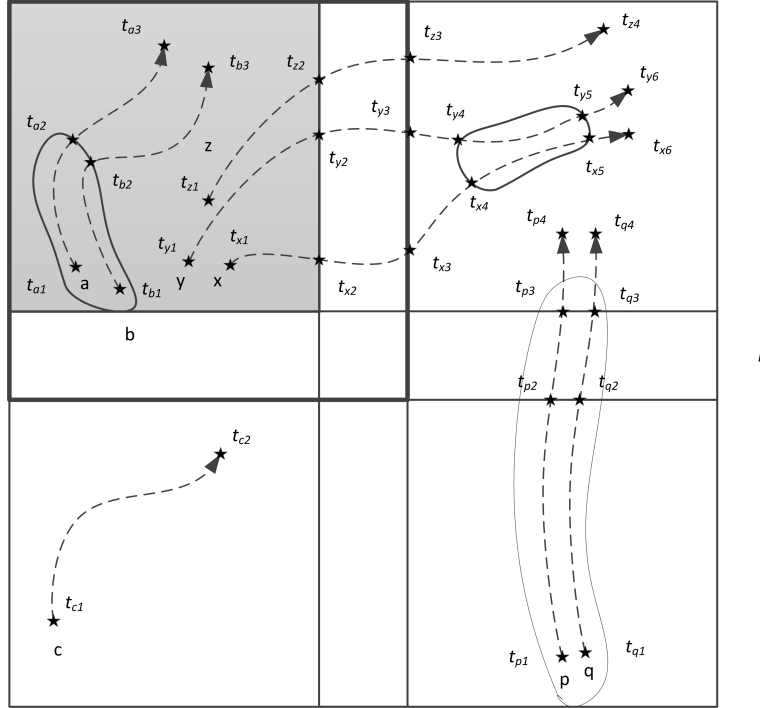


Figure 5.7: How Algorithm A-2 Divides a Trajectory Database for Scalable Approximation of TRAJCLIQs.

From this observation, we deduce that if we divide the spatial-space into multiple zones with stripes of width r between them, the (class of) sub-trajectories, which pass the stripes, cannot form a TRAJCLIQ with the sub-trajectories confined in the zone. In the above example depicted in Fig. 5.7, there are four zones and two stripes — one horizontal and one vertical. The sub-trajectories passing either the horizontal and vertical stripes cannot form TRAJCLIQs with any sub-trajectories confined in the four zones, i.e. sub-trajectory $sub_x(t_{x1}, t_{x4})$ passes the vertical strip and, hence, it cannot form a TRAJCLIQ with any sub-trajectories of a , b , c and some sub-trajectories of x , y , z , p , and q confined in one of the zones like $sub_x(t_{x1}, t_{x2})$, $sub_y(t_{y1}, t_{y2})$, $sub_z(t_{z3}, t_{z4})$, $sub_p(t_{p1}, t_{p2})$ and $sub_q(t_{q3}, t_{q4})$.

Algorithm 5.3 outlines the Divide and Conquer Scheme and it requires two additional parameters, initial zone-size λ_0 and zone-size multiplier θ , than A-0 and A-1 do. $\text{MBR}(\mathcal{R})$ returns the minimum bounding rectangular of all trajectories in \mathcal{R} ; $\text{Divide}(\text{MBR}(\mathcal{R}), \lambda, r)$ returns the $\text{MBR}(\mathcal{R})$ divided into $\lambda \times \lambda$ zones separated by strips of width r ; $\text{Extend}(z, r)$ returns the rectangular area z extended by r on all its sides and $\text{Max-Subs}(\mathcal{R}, z)$ returns all maximal sub-trajectories — sub-trajectories of trajectories found in \mathcal{R} — confined in the rectangular area z .

Algorithm 5.3 Divide-and-Conquer Frequent Route Miner (A-2).

Input: $\lambda_0, \theta, \mathcal{R}, r, m$ and l .

Output: A set of frequent routes \mathcal{F} .

```

1: Set  $\lambda \leftarrow \lambda_0$  and  $max_s \leftarrow \max(\text{MBR}(\mathcal{R}).length, \text{MBR}(\mathcal{R}).breadth)$ 
2: while  $\lambda \leq max_s$  do
3:   Set of zones  $Z \leftarrow \text{Divide}(\text{MBR}(\mathcal{R}), \lambda, r)$ 
4:   for all Zone  $z \in Z$  such that there exists a trajectory confined in  $z$  do
5:      $z^+ \leftarrow \text{Extend}(z, r)$ ,  $\mathcal{R}_z \leftarrow \text{Max-Subs}(\mathcal{R}, z^+)$ , and  $\mathcal{U} \cup$  A-1 ( $\mathcal{R}_z, r, m, l$ )
6:     Set  $\mathcal{R} \leftarrow \mathcal{R} - \{traj : traj \text{ is totally confined in } z\}$ 
7:     Set  $\lambda \leftarrow \lambda \times \theta$ 
8:  $\mathcal{U} \leftarrow \mathcal{U} - \{U | U \text{ is not a closed-TRAJCLIQ}\}$ .
9: for all  $Q \in \mathcal{U}$  do
10:   $\mathcal{F} \leftarrow \mathcal{F} \cup \{\text{Get-Frequent-Route}(U)\}$ 

```

Algorithm A-2 works as follow. The spatial-space is divided into zones with stripes of width r between the zones of size $\lambda \times \lambda$, starting with λ at the user-defined zone-size λ_0 (line 3). For each zone z , which has at least a trajectory wholly confined in, the sub-trajectories in its extended zones z^+ are processed using algorithm A-1 and adding the results to \mathcal{U} (lines 4 - 6). If the zone z does not confine any whole trajectory, all sub-trajectories in it would be processed in the next pass of the loop (with larger λ). After each extended zone z^+ is processed, trajectories wholly contained in z is removed from \mathcal{R} as it cannot form any more TRAJCLIQs with other sub-trajectories (passing the strips) in the next pass (line 6). After all zones are processed, the zone-size is enlarged (line 7). When a single zone span the entire spatial space, A-2 outputs a frequent route for each closed-TRAJCLIQ.

We will illustrate how the algorithm A-2 works using the example trajectories⁴ in Fig. 5.7. Initially, it divides the spatial-space into four zones. For the top-left zone, trajectories a and b , and the maximal sub-trajectories of x , y and z confined in the outer bounded box (shown by thick borders), i.e. $sub_a(t_{a1}, t_{a3})$, $sub_b(t_{b1}, t_{b3})$, $sub_x(t_{x1}, t_{x3})$, $sub_y(t_{y1}, t_{y3})$, and $sub_z(t_{z1}, t_{z3})$, are processed using A-1. Since sub-trajectories $sub_a(t_{a1}, t_{a2})$ and $sub_b(t_{b1}, t_{b2})$ form a TRAJCLIQ, it is added to \mathcal{U} . Before moving on to the next zone, A-2 removes trajectories of a and b from \mathcal{R} . Then, A-2 continues to the top-right zone, in which no trajectory is confined. Therefore, A-2 moves on to bottom-left zone (finds no TRAJCLIQ and removes trajectory c) and bottom-right zone (finds the TRAJCLIQ $U'_{p,q} \langle sub_p(t_{p1}, t_{p3}) \text{ and } sub_q(t_{q1}, t_{q3}) \rangle$, add it to \mathcal{U} , and removes trajectory d). After all four zones are processed, the algorithm A-2 tries to find TRAJCLIQS formed by sub-trajectories that pass the stripes. In this stage, two TRAJCLIQS, TRAJCLIQ $U_{x,y} = \langle sub_x(t_{x4}, t_{x5}), sub_y(t_{y4}, t_{y5}) \rangle$ and $U_{p,q} = \langle sub_p(t_{p1}, t_{p4}), sub_q(t_{q1}, t_{q4}) \rangle$ are reported. Since $U_{p,q}$ covers $U'_{p,q}$, $U'_{p,q}$ is removed from \mathcal{U} . Notice that the approximation processing for $U'_{p,q}$ is wasted. Finally, the algorithm reports the three frequent routes corresponding to the TRAJCLIQS in \mathcal{U} . By design, algorithm A-1 and A-2 has the same output and the same accuracy.

We speculate that for large Trajectory Databases containing thousands of objects with high-resolution trajectories (high sampling rates), A-2 would provide a reasonable trade-off because it divides both the search space of *Apriori* processing and the portions of Free-space needed in the memory to approximate TRAJCLIQS. Even when the memory consumption is not a major issue, the divisions of the TJDBs A-2 provides can be processed independently from each other using A-1 and, hence, A-2 can be used to enable parallel processing for Mining Frequent Routes.

As a hindsight that the performance of A-2 depends on whether it can remove as many (whole) trajectories as possible (in line 6, Algorithm. 5.3) in the earlier

⁴The time-stamps in this example are also arbitrary.

stages (as evidenced by our experiments in Sect. 5.4) and, hence, depends on initial zone-size λ_0 and zone-size multiplier θ . We suggest it is possible to optimize these two parameters in each loop before the zone division in order to improve the performance of A-2.

5.4 Experimental Evaluations

5.4.1 Experiment Setup

We implemented algorithms A-1 and A-2 in Java along with the existing polynomial time Sweep algorithm proposed in [12] and Traclus proposed in [37]. Sweep is a 2-distant approximation algorithm for mining sub-trajectory cliques, i.e. its approximation factor is 2. We made minor changes in Sweep to ensure all sub-trajectories in each reported approximated TRAJCLIQs have length l as well as introduced standard programming optimizations to it. Traclus is a three-step algorithm comprising: partition-step, grouping step and representative trajectory calculation step.

For all algorithms benefit from line-segment view of the data, we stored the point, loc_{next} , which is defined as the loc an object o is going to visit in the immediately next time-stamp recorded in the Trajectory Database, in each record. We also had an R-Tree index for the line-segment $\langle loc, loc_{next} \rangle$. All distance units are in metre.

5.4.2 Results and Analysis

Table 5.5 shows the default parameters and run-time performance of each algorithm for different datasets. Sweep uses the same set of parameters m , l , and r as algorithms A-1 and A-2, while, for Traclus, we use all the parameter values as suggested in [37]. The original proposed partition method for Traclus tends to

eliminate shorter sub-trajectories in favour of the longer one in a dataset. Therefore, we made changes to the partition step to ensure short sub-trajectories were also maintained in our experiments. Sweep, A-1, and A-2 use Free-space, which is pre-computed for each dataset (but the Free-space computation time is included in their overall run-time we report here) and stored as a sparse graph. For A-2, We used different initial zone sizes λ_0 for different datasets (see the last column in Tab. 5.5). We stopped Traclus for Ships dataset after it took several days running. We decided not to run Traclus on land vehicle datasets, SF-Cabs21rand100 and SF-Cabs22rand100 because its output for similar dataset, Taxi is not satisfactory (details will be reported shortly). In all datasets, we observe that our proposed algorithms, A-1 and A-2, finished faster than Sweep and Traclus. Traclus took the most time because the distance measure it uses cannot make use of any spatial index, including the R-tree index we used. A-1 and A-2 performed faster than Sweep because of their pruning mechanisms. A-1 took less time compared to A-2 for all datasets except Trucks because the Free-space data structure fits in the main memory (favouring A-1) and performance of A-2 depends on the division of datasets into zones, which can confine as many (whole) trajectories as possible — hence, on λ_0 and θ . We expect, less difference between the two algorithms when the computation environment has a low-memory settings, which prohibits the Free-space for all trajectory pairs from being pre-computed and put in the memory, and when λ_0 and θ are optimized based on the dataset at hand.

Figure 5.8 shows the frequent routes we extracted from Ships dataset using algorithm A-1 in colour (all the trajectories in the dataset are shown in grey). Trajectories in black are frequent routes used at least five times in four hour period, which we consider — combining with the fact that they are at least 1.5 kilometre long — as significant giving that ships do tend to follow the exact same routes. Algorithm Sweep also reported a very similar results, which contain more false-positives than

Table 5.5: A Summary of the Default Parameters and Performance of the Frequent Route Mining Algorithms for Each Dataset.

Dataset	m	r	l	Run-time (seconds)				λ_0
				Sweep	A-1	A-2	Traclus	
Statefair	3	30	500	22	9	11	870	5,000
NCSU	3	30	500	129	63	65	6,722	5,000
New York	3	30	500	178	103	113	3,427	5,000
Orlando	3	30	500	196	96	101	18,088	5,000
KAIST	3	30	500	2,617	1,237	1,369	71,443	5,000
SF-Cabs21rand100	3	20	2,000	966	508	983	—	15,000
SF-Cabs22rand100	3	20	2,000	1,065	567	924	—	15,000
Trucks	3	20	2,000	5,709	3,323	1,416	88,123	5,000
Ships	3	100	1,500	500	262	610	—	10,000

A-1 and A-2 as, although all of Sweep, A-1, and A-2 have the same approximation factor of 2, A-1 and A-2 exploits the *a priori*-properties of TRAJCLIQs to prune some false-positives while Sweep does not have any pruning mechanism.

Figure 5.9 compares the significant portions of the results produced by Traclus with the frequent routes, which were used more than five times, produced by A-1 for the same area. The results produced by Traclus, using its default parameters, contain a few trajectory clusters, from which we need to extract representative trajectories. The representative routes are often short and straight lines. Traclus failed to find longer and more complicated routes like A-1 did — Traclus did not find the route travelling east to west in the eastern side of the map. The results of A-1 include frequent routes across junctions (involving turns), which Traclus failed to find — notice the route in the middle of the map, which involves turns and crosses junctions. Similarly, in Fig. 5.10, which shows a trajectory cluster and frequent routes found in a different area of Athens than Fig 5.9, A-1 found a longer frequent route, which crosses multiple junctions, while Traclus only found a portion of that frequent route.

Figure 5.11 shows the visualization of the frequent routes (used more than five times) discovered by A-1 in Trucks dataset (in black) against a background of all

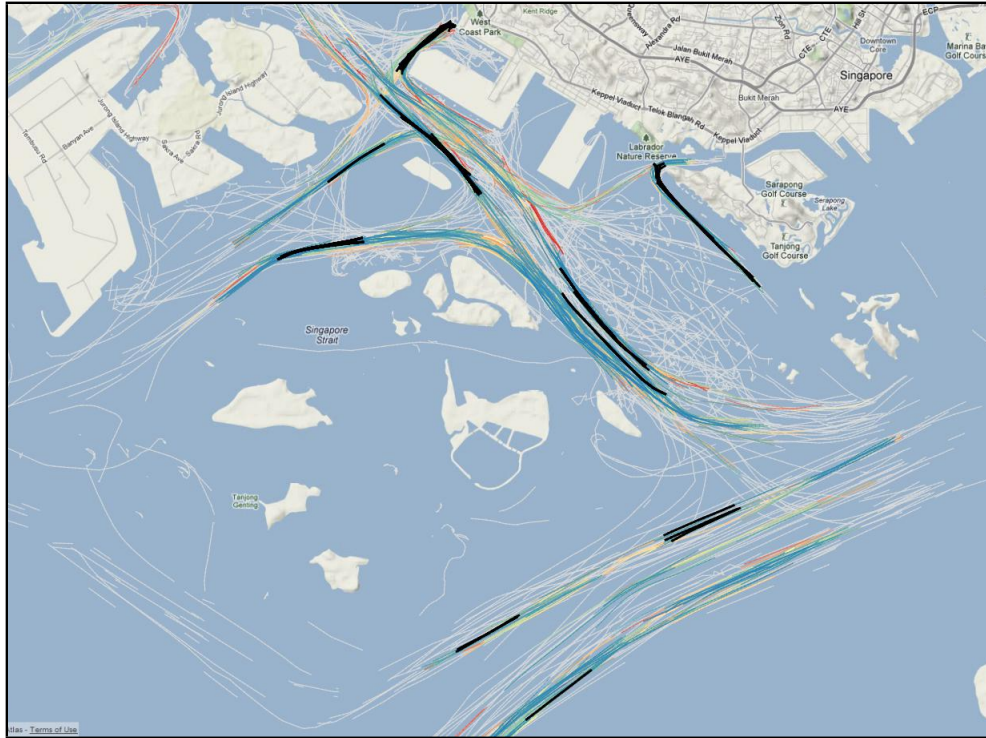


Figure 5.8: Frequent (Used at least three times and five times) Routes of the Ships Discovered by Algorithm A-1 (in colors and in black) Superimposed on all Trajectories (in grey) in the Dataset.

trajectories in the dataset (in light pink). It clearly shows the major road sections in Athens, which the trucks in the dataset used multiple times.

To summarize the comparison of the algorithms' outputs, we learnt that our proposed algorithms produced results, which agrees to human intuitions and which contains fewer false positives than Sweep. On the other hand, Traclus, using suggested parameters, failed to find some frequent routes, which our proposed algorithms found. Since Traclus failed to deliver intuitive output and it took longer to process for each dataset, we omitted Traclus from further sets of experiments.

Table 5.6 shows a summary of memory footprints we collected from the garbage collector logs of algorithms A-1 and A-2 to assess the memory saving A-2 brought for some datasets. We found that for most datasets (except NCSU), despite the similar peak memory allocation, the garbage collector could free up (and reuse) more memory when running algorithm A-2 than when running A-1. We hypothesized that



(a)



(b)

Figure 5.9: (a) Trajectory Clusters of the Trucks Discovered by Traclus and (b) Frequent (Used at least five times) Routes of the Trucks Discovered by Algorithm A-1 in the Same Area.

in such instances, although it is possible to reuse the garbage collected memory, Java Virtual Machine (JVM) decided to allocate more memory instead of waiting for garbage collection making the peaked allocation of A-1 and A-2 similar. Should the JVM decides not to allocate more memory and wait for the garbage collection, the peaked allocation of A-2 would be much less than that of A-1. This claim is supported by a closer inspection of memory footprints for NCSU, in which, although A-2 seemingly allocated more memory (at peak) and freed up less than A-1 did, its peak (actual) memory usage was only 0.5GB (it did not use a large portion of the the allocated memory) compared to actual usage of A-1, over 1.0GB.

We conducted a set of experiments in order to assess the quality of false-positive reduction mechanism employed in A-1 (and, by extension, also in A-2, which uses



Figure 5.10: (a) A Trajectory Cluster of the Trucks Discovered by Traclus and (b) Frequent (Used at least five times) Routes of the Trucks Discovered by Algorithm A-1 in the Same Area.

Table 5.6: Memory Footprint of Algorithms A-1 and A-2.

Dataset	A-1		A-2	
	Peaked Allocated	Total Freed	Peak Allocated	Total Freed
NCSU	3.4GB	3.0GB	2.4GB	0.9GB
New York	2.4GB	3.0GB	3.4GB	6.0GB
KAIST	6.4GB	72.4GB	6.4GB	102.6GB
Ships	1.9GB	2.5GB	2.0GB	7.2GB

A-1 as a sub-routine). We developed A-1 (FP), which is essentially A-1 without the mechanism to remove the false-positives and compared its results and run-time with those of A-1. Table 5.7 shows detailed comparison of the results and run-time of A-1 and A-1 (FP). In all but NCSU and New York datasets, A-1 (FP) reported a significantly higher number of closed-TRAJCLIQS, all of which we confirmed as false-positives. In NCSU and New York datasets, some false-positives A-1 (FP) could not prune would connect two TRAJCLIQS A-1, reducing the total number of closed-TRAJCLIQS A-1 (FP) reported. Nonetheless, even in such cases, we were able to confirm that the total length of all sub-trajectories (without counting any portion twice) in the TRAJCLIQS A-1 (FP) reported is longer than that of A-1. Although A-1(FP) finished the smaller datasets (Statefair, NCSU, New York, and Orlando) around at the same time as A-1 does, it took longer time for the larger

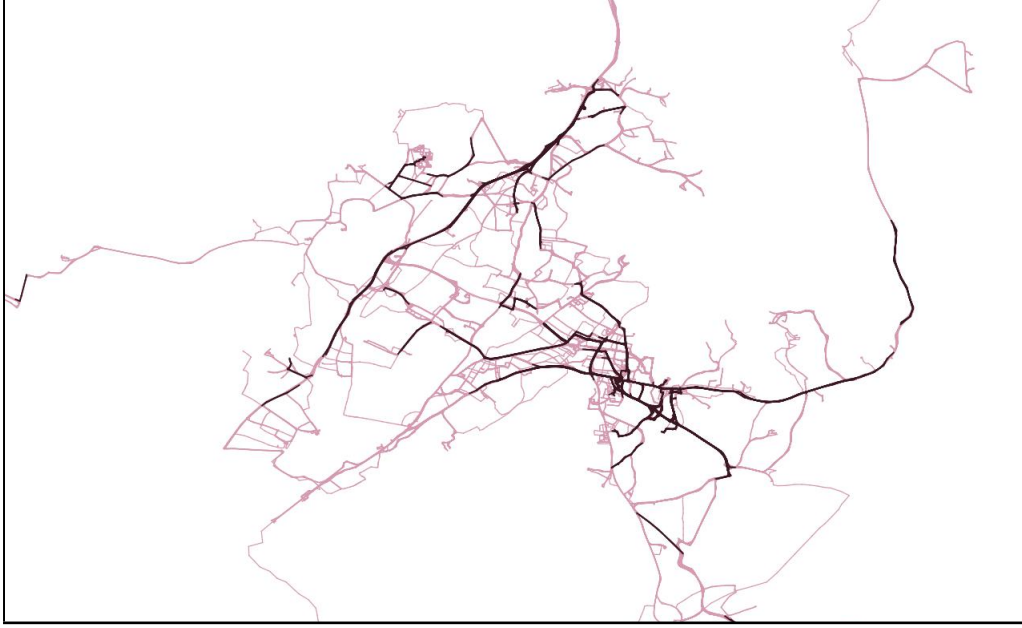


Figure 5.11: All Frequent (Used at least five times) Routes of the Trucks Discovered by Algorithm A-1 (in black) Superimposed on All Trajectories (in light pink) in the Dataset.

dataset (KAIST) because A-1 (FP) had to process the false-positives, which A-1 efficiently pruned off.

Table 5.7: Results and Performance of Algorithms A-1 and A-1 (FP).

Dataset	A-1		A-1 (FP)	
	Run-time (seconds)	Number of Frequent Routes	Run-time (seconds)	Number of Frequent Routes
Statefair	9	22	12	21
NCSU	63	445	64	268
New York	103	212	102	111
Orlando	96	306	99	419
KAIST	1,237	3,953	1,4,37	16,216

In the subsequent sets of experiments, we will assess the impact of the parameters on the algorithms in New York and KAIST datasets. The impact of parameters are also similar to Orlando and NCSU datasets.

Figure 5.12(a) and Figure 5.12(b) shows the total running time of algorithms for datasets, New York and KAIST, using default l and r while varying m . For both New York and KAIST, A-1 and A-2 finished faster than Sweep, with A-1 being the fastest. The changes in the value of m did not significantly affect the run-time of

all algorithms although the run-times were slightly reduced when $m = 4$ as larger m values result in fewer frequent routes required to be outputted.

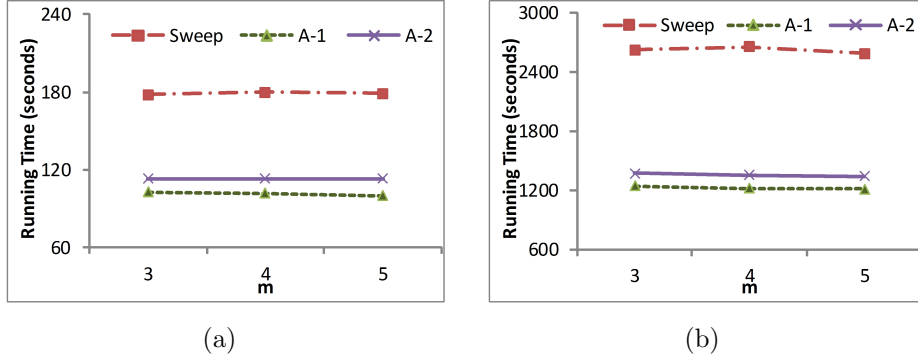


Figure 5.12: Impact of the Parameter m on the Performance of the Frequent Route Mining Algorithms Using (a) New York and (b) KAIST with Parameters $l = 500$ Metre and $r = 30$ Metre.

Figure 5.13(a) and Fig. 5.13(b) show the total running time for datasets, New York and KAIST, with default m and r while varying l . For both datasets, as l increases, the total running time for both A-1 and A-2 decreases, which is more pronounced for A-2 on KAIST. It is because as l increases, the number of TRAJCLIQS and frequent routes decreases and A-1 and A-2 are output sensitive algorithms. We observed A-1 always performed faster than the others and A-2 performed faster than Sweep except in KAIST when $l = 300m$. A-2 took longer time in KAIST for $l = 300m$ because KAIST contains longer trajectories, which A-2 cannot prune early, forming using a large set of shorter frequent routes A-2 has to wastefully process multiple time.

Figure 5.14(a) and 5.14(b) compare the performance of the algorithms for New York and KAIST using default m and l while using different values for r . We see that the running times for all algorithms increase when r increases because increasing r also increases the number of TRAJCLIQS and, hence, the number of frequent routes in the results. A-1 and A-2 still outperformed Sweep and the run-time of Sweep increases faster than A-1 and A-2 when r increases. Regardless of the value of r

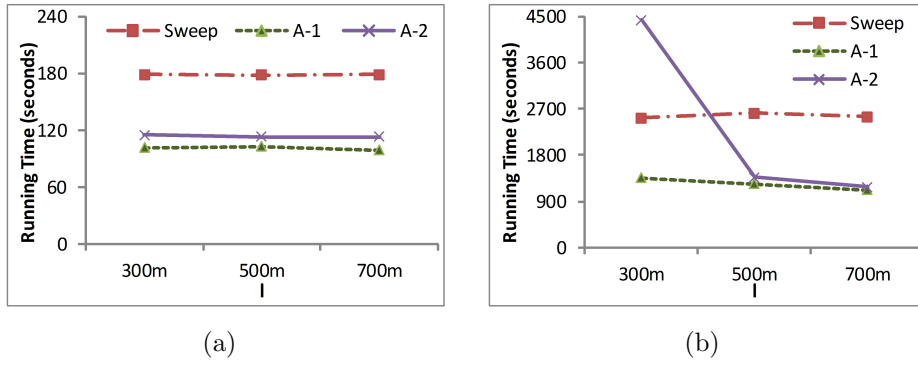


Figure 5.13: Impact of the Parameter l on the Performance of the Frequent Route Mining Algorithms Using (a) New York and (b) KAIST with Parameters $m = 3$ and $r = 30$ Metre.

given, A-1 finished faster than A-2.

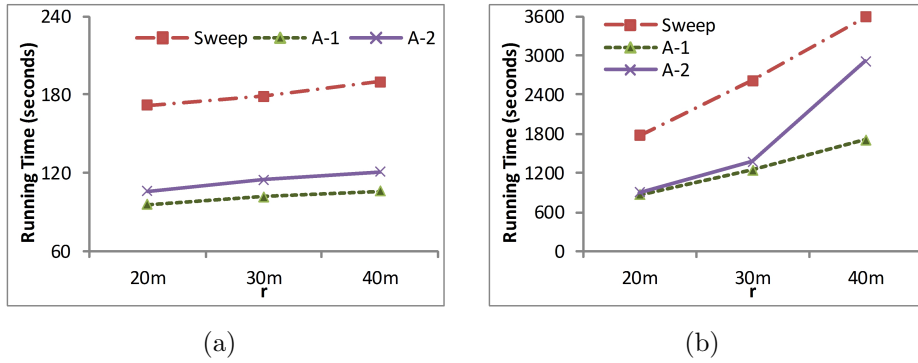


Figure 5.14: Impact of the Parameter r on the Performance of the Frequent Route Mining Algorithms Using (a) New York and (b) KAIST with Parameters $m = 3$ and $l = 500$ Metre.

To summarize our experiment results, we conclude that, for real-life settings (real-life datasets and intuitive parameter values), our proposed algorithms (A-1 and A-2) provide more intuitive results and perform faster than Traclus using its default parameters. They also perform faster than the polynomial time approximation algorithm Sweep due to their pruning power based on *a priori*-properties of TRAJCLIQS. Although A-1 performed faster than A-2 in our experiments, with larger Trajectory Databases and limited amount of memory, A-2 would give a reasonable trade-off between run-time and memory requirements in finding sub-trajectory cliques and frequent routes.

5.5 Summary

In this chapter, we studied techniques to find frequent routes in Trajectory Databases without any prior knowledge of the underlying spatial space. We proposed to mine sub-trajectory cliques (sub-trajectory clusters) called TRAJCLIQS using Fréchet distance as the similarity measure and extract frequent routes from the resulting TRAJCLIQS. Since mining all TRAJCLIQS is an NP-Complete problem and exact algorithms even from data-driven perspective are not feasible, we proposed two approximate algorithms based on the *Apriori* algorithm. Empirical results show that both of our proposed algorithms can run faster than the existing polynomial time approximation algorithm and provide a tighter results, although the theoretical approximation factor is still the same. The second algorithm we proposed is a divide and conquer algorithm, which sub-divides the input Trajectory Database into subsets so that each subset can be processed by instances of the first one. The divide and conquer algorithm runs slower than the first one yet provides opportunities for parallelism and/or memory efficiency. Our experiment results showed that the frequent routes reported by our algorithms are more intuitive (longer, contain turns and crosses junctions) than those reported by existing clustering algorithm like Traclus using its default parameters.

Discovery of Evolving Convoys

6.1 Introduction

From Trajectory Databases obtained from GPS data and other sensor data, many new applications such as monitoring of convoys and the stages of their life-time evolution thread — forming, gaining (losing) members and disbanding — become a reality in addition to simple discovery of the existence of convoys. A traditional convoy is routinely defined as a group of moving objects that are close to each other for a period of time. Existing techniques, following this traditional model (definition), cannot find evolving convoys with dynamic members and do not have any monitoring aspect in their design. In this chapter, we will propose new concepts called dynamic convoys and evolving convoys, which reflect real-life scenarios, and develop algorithms to discover evolving convoys in an incremental manner.

Object identification and tracking technologies as well as triangulation techniques enable monitoring and archiving movement data of objects. For example, in an urban setting, pedestrian and vehicle movement can be recorded using a combination of GPS, cellular networks, Wi-Fi hotspots, and other radio frequency (RF) sensor networks. Recent developments in such technologies make it possible to obtain movement data at a high temporal resolution (high update-rate). These data can be used to find interesting Multi-object Movement Patterns called convoys.

Discovering and monitoring convoys have many practical applications ranging

from traffic planning to wild-life research and even on-line games. Traffic planners can benefit from knowledge of trucks moving in convoys (moving in groups) between factories, warehouses, and stores. Convoy discovery can be used to extract complex herding behaviour of wild animals from GPS-collar data. In on-line games, where players can form allies, monitoring them and providing a different difficulty level for each group can lead to a more enjoyable game-play.

Definition. 6.1. *Traditional Convoy* — *A traditional convoy can be defined as a set of m or more objects, which are within proximity of each other for a duration $dur \geq w$ or longer period of time, where $m > 1$ and $w \geq 1$ are user defined parameters.*

Definition 6.1 serves as a template to describe how the existing works [10, 23, 26, 30, 32, 54] defined a group of tracked objects, which move together. The only difference between the existing definitions of a convoy is the togetherness criteria (i.e. how to determine the spatial proximity in question) — definition of a flock [10, 23] determines it as being covered by a circle of the given size, that of a moving group [26, 54] determines it as being within a given distance from each other (form a clique) and that of a convoy [30, 32] determines it as being in the same density-connected cluster. We will collectively term all the models, which fit Def. 6.1 as “traditional convoys”.

The concept of dynamic convoys and evolving convoys we will discuss in this chapter applies to traditional notion of convoys defined using any togetherness criteria (spatial proximity). However, we focus our studies on convoys formed from density-connected objects because defining spatial-proximity as density-connection is more relevant for real-world objects. In reality, many moving objects, such as trucks, pedestrians etc, occupy a spatial-space, which is not shared with other objects. Thus, a set of moving objects may not fit into a circle (form a clique) of

given size (with given maximum distance between members) if it has a large number of members. In contrast, a convoy formed from density-connected objects can occupy a spatial region of arbitrary size and shape in its lifetime rather than a fixed geometric shapes such as a circle. We will use the following running example to illustrate the novel concepts in this chapter.

Example 6.1. *Figure 6.1 shows movement of five commuters. Alice (a), Bob (b), and Cathy (c) were heading to works (work1 and work2), when Elvis (e) joined them. Later, Bob took a de-tour to purchase some petrol for his car from a petrol station (gas). David (d) followed a different route. Circles with time-stamps show spatial proximity.*

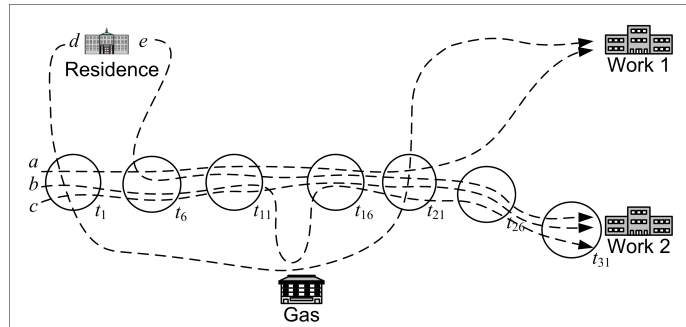


Figure 6.1: Trajectory Database Containing Five Commuters' Movements.

Definition. 6.2. *A convoy C is closed if and only if there is no convoy C' that contains all members of C and has a life-span completely covering that of C , i.e. C is a closed convoy if and only if $\nexists C'$ such that $C \subseteq C'$, $t_{\text{start}}(C') \leq t_{\text{start}}(C)$ and $t_{\text{end}}(C) \leq t_{\text{end}}(C')$.*

Under Def. 6.1, a subset of a convoy is often a convoy. Since subsets can be derived from the larger convoy that contains them, only *closed* convoys defined as in Def. 6.2 are of interest in many applications. For example, in Fig. 6.2, which illustrates the events of Example 6.1 between t_6 and t_{14} in details, $\{a, b, c\}$ forms

a convoy during $[t_6, t_{13}]$. However, it is not a closed convoy in the sense that its superset, $\{a, b, c, e\}$ forms a convoy during the same time-frame, $[t_6, t_{13}]$.

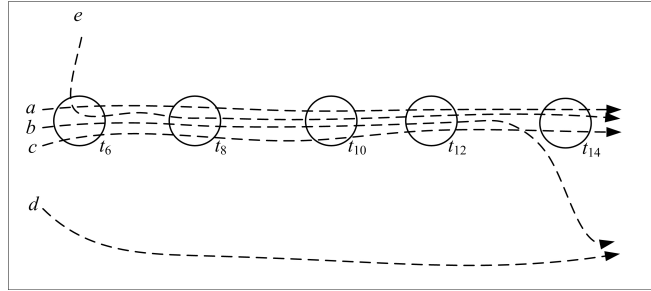


Figure 6.2: Detailed Movements of the Five Commuters from Fig. 6.1 during $[t_6, t_{14}]$.

Although Def. 6.1 is intuitive, the notion of convoys having only persistent members does not reflect real-life scenarios and, hence, it cannot be directly applied. For the movement scenario described in the example 6.1, when a car-pool administrator, who wants to know whether car-pooling is possible for the commuters, tries to find *closed* convoys of size $m = 2$ and duration $w = 5$, he will be overwhelmed by a result containing seven closed convoys as listed in the first column of Tab. 6.1.

Table 6.1: Maximal Convoys Formed by Five Commuters Depicted in Fig. 6.1.

Convoys	Life-span	Note
$C_1 = \{a, c\}$	t_1-t_{22}	Not covered by C_3 or C_4 (life-span not fully covered)
$C_2 = \{c, e\}$	t_6-t_{31}	Not covered by C_4 or C_5 (life-span not fully covered)
$C_3 = \{a, b, c\}$	t_1-t_{13}	Not covered by C_6 or C_7 (life-span not fully covered)
$C_4 = \{a, c, e\}$	t_6-t_{22}	Not covered by C_6 or C_7 (life-span not fully covered)
$C_5 = \{b, c, e\}$	$t_{15}-t_{31}$	Not covered by C_6 or C_7 (life-span not fully covered)
$C_6 = \{a, b, c, e\}$	t_6-t_{13}	Different from C_7 ; different life-span.
$C_7 = \{a, b, c, e\}$	$t_{15}-t_{22}$	Different from C_6 ; different life-span.

From the above example, we made the following observations on nature of real-life convoys that Def. 6.1 (and Def. 6.2) cannot cope with :

1. Some members of the convoy may temporarily leave the group. Actually a, b, c , and e formed a convoy from t_6 to t_{22} as they were literally moving together. However, according to Def. 6.1, there were three convoys (C_4, C_6 , and C_7) for

their movement in Tab. 6.1 because b was not detected together with the rest (a , c , and e) at a single time-stamp t_{14} resulting in a short gap that eventually creates three overlapping convoys. Hence, we observed that the definition of convoy needs to be more flexible to allow some dynamic members to move away occasionally. We refer to a convoy that allows dynamic members a *dynamic convoy*.

2. Some members may join (leave) the convoy later (earlier) than the convoy's starting (ending) time — resulting in the convoys evolve into a larger (smaller) convoy. In our example, the convoy $\{a, b, c\}$ was joined by e and evolve into a larger convoy $\{a, b, c, e\}$ at t_6 , which, in turn, evolves into a smaller convoy $\{b, c, e\}$ at t_{23} . While this is intuitively a single group of moving objects that had a new member joining and an existing member leaving in different stages of its life-span, there are seven maximal convoys in the result reported to the administrator. Representing this single evolving convoy as seven overlapping convoys is not intuitive and hard to comprehend for human users. It is also difficult to order or establish relationship among the overlapping convoys. Hence, we noted that the definition of convoy needs to take account of objects joining (leaving) the existing convoy since it will be more useful if the moving group is represented as three stages of a single *evolving convoy*.

Many existing works do not have a satisfactory mechanism to handle the convoys' behavior we observed above. For the scenario in Fig. 6.1, the algorithms proposed by an existing work [32] report only two convoys – $\{a, c\}$ from t_1 to t_{22} and $\{b, c, e\}$ from t_{23} to t_{31} . Start-time of $\{b, c, e\}$ is wrongly reported as t_{23} instead of actual t_{15} . This error is introduced because c was in many convoys, whose life-spans were overlapped, and the earlier $C_1 = \{a, c\}$ is favoured over the rest (including $C_5 = \{b, c, e\}$ and $C_2 = \{c, e\}$), pushing their start-times until it ended

at t_{22} . As a result, start-time of $\{b, c, e\}$ is erroneously reported and the longest-duration convoy, $C_2 = \{c, e\}$, is not reported at all¹. The technique in [54] (and [26]) reports all convoys (all seven closed convoys) without any information to establish links between the results. Algorithms in another work [33] report different number of cluster sequences for different values of similarity threshold, θ . For example, two cluster sequences will be reported for similarity threshold $\theta = 0.70$.

It is a difficult challenge to report all closed convoys from a given Trajectory Database, i.e. for Fig. 6.1, listing all seven closed convoys. A brute force solution would be to check each subset of O , which contains two or more members (in this case there are 27 such subsets), throughout all possible time-partitions of five or more consecutive time-stamps (in this case there are 378 such partitions). However, this may also result in many convoys that are not closed and, hence, need another expensive post-processing step.

Chapter Contributions

The contributions of this chapter are:

1. **Introducing novel concepts of dynamic convoys and evolving convoys** — in contrasts to traditional persistent-members-only definitions, the new definition of **Dynamic Convoy (DYCO)** allows dynamic members under constraints imposed by user-defined parameters. An **Evolving Convoy (EC)** captures the relationship between different stages of convoys such that a convoy in a stage has more (fewer) members than its previous stage.
2. **Development of three algorithms that can be used to incrementally discover evolving convoys** — all proposed algorithms are incremental in nature and can be used in both off-line and streaming data.

¹Personal communications with the authors confirmed this claim.

To the best of our knowledge, this is the first work that addresses the Dynamic Convoys and Evolving Convoys². Convoys with dynamic members are natural and common in real-life. Information on membership is equally important to the mere existence of convoys as many applications need to examine the members of a convoy for further processing. In the above example, which commuters form convoys is a natural question after discovering some convoys so that car-pooling suggestions can be dispatched to appropriate persons.

6.2 Dynamic Convoys and Evolving Convoys

Definition. 6.3. *Dynamic Convoy* – For given parameters: m , k , and w ($m > 1$, $1 \leq k \leq w$), a set of moving objects D forms a **Dynamic Convoy (DYCO)** from $t_{\text{start}}(D)$ to $t_{\text{end}}(D)$ if it :

- Contains at least m persistent members (denoted by PM_D), all of which are in the same density-connected cluster in each time-stamp t in $[t_{\text{start}}(D), t_{\text{end}}(D)]$, i.e. $PM_D \subseteq D$ and for $t_{\text{start}}(D) \leq t_i \leq t_{\text{end}}(D)$, $|PM_D| \geq m$, and $PM_D \subseteq L_i$, where L_i is a maximal density-connected cluster found in time-stamp t_i and
- Contains zero or more dynamic members (denoted by DM_D), each of which must be in the same density-connected cluster with the persistent-members at least k times for any w sliding window in $[t_{\text{start}}(D), t_{\text{end}}(D)]$, i.e. $D = PM_D \cup DM_D$ and if $PM_D \subseteq L_j$, which is a density-connected cluster found at time-stamp t_j , then, for any dynamic member $o \in DM_D$, $t_{\text{start}}(D) \leq t_i \leq t_{\text{end}}(D) - w + 1$, and $i \leq j \leq i + w - 1$, the number of times $o \in L_j \geq k$.

Definition 6.3 introduces flexibility to Def. 6.1 by defining a discipline for some members (dynamic members in DM_D) concerning leaving and returning their parent

²For detailed comparison between different convoy models, refer to Sect. 3.2.2 through 3.2.6

convoy. The first condition ensures that, for any given w consecutive time-stamps (called w period) in a convoy's life-span, a fixed set of the persistent members (PM_D) form its main body. The second condition requires each dynamic member ($o \in DM_D$) to stay close with the set of persistent members frequently enough – at least k times in any w sliding window in the convoy's life-span. This constraint filters out occasional by-passers from being reported as a (dynamic) member of a dynamic convoy. For smaller k values, a dynamic-member can move away from the convoy for a longer period while larger k values prohibit a dynamic-member from being away for a long time ($k = w$ means no dynamic-member is allowed and, hence, the convoy becomes a traditional convoy.) For example, in Fig. 6.1, with $m = 2$, $w = 5$, and $k = 4$, $\{a, b, c, e\}$ forms a dynamic convoy from t_6 to t_{22} .

It is clear that for a dynamic convoy D , PM_D forms a traditional convoy. Therefore, for a given Trajectory Database, we can have as many dynamic convoys as we have traditional convoys. Although dynamic convoys allow dynamic members to move away (under discipline) from the parent convoys, they cannot handle the case(s) of a new (existing) member entering (leaving) the convoy well. For instance, for $m = 2$, $k = 4$, and $w = 5$, in the scenario described in Example 6.1, $D_1 = \{a, b, c\}$ in $[t_1, t_{22}]$, $D_2 = \{b, c, e\}$ in $[t_6, t_{31}]$, $D_3 = \{a, b, c, e\}$ in $[t_6, t_{22}]$ and many of their subsets are all dynamic convoys. From usability point of view, reporting all (dynamic) convoys, whose members and life-spans are overlapped, may be confusing. It is also difficult for a human user to establish relationship between overlapping dynamic convoys. Selecting a representative from overlapping convoys is, however, application-dependent. For example, some administrators may be interested in longer-duration convoys (like D_2) while others may be interested in larger convoys (like D_3). A more comprehensive approach is to report each set of overlapping convoys as an evolving entity with stages.

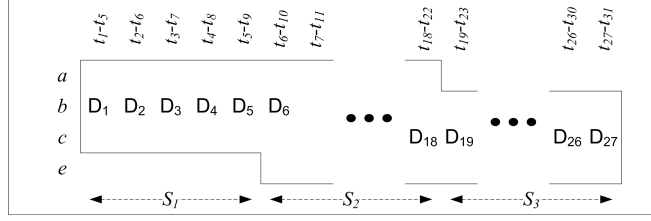


Figure 6.3: The Concept of Convoy Evolution.

Definition. 6.4. w -convoy – A dynamic convoy of duration $dur = w$ is called a w -convoy.

For given m , k , and w , a dynamic convoy D of duration $dur \geq w$ has $dur - w + 1$ w -convoys $D_1, D_2, \dots, D_{(dur-w+1)}$ of duration w , each of which has the same persistent-members and dynamic-members as D . For example, the convoy $D_1 = \{a, b, c\}$ (see above paragraph) that exists from t_1 to t_{22} has 18 convoys of duration w , each having the same set of members as D_1 .

Definition. 6.5. Evolution of w -convoy – A w -convoy D that exists from t to $t + w - 1$ evolves into another w -convoy D' that exists from $t + 1$ to $t + w$ if they have at least m common persistent-members, i.e $|PM_D \cap PM_{D'}| \geq m$.

Definition 6.5 defines how a w -convoy can evolve into the next w -convoy of duration w . It ensures that a convoy evolves only into a related convoy (not to a convoy with totally different members). It also prohibits a convoy from evolving to an earlier convoy or back-ward evolving. The w -convoys formed by a , b , c , and e in the scenario in Fig. 6.1, for parameters $m = 2$, $k = 4$, and $w = 5$, is shown in Fig. 6.3. $D_1 = \{a, b, c\}$ that exists from t_1 to t_5 evolves into $D_2 = \{a, b, c\}$ that exists from t_2 to t_6 , which in turn evolves into D_3 , D_4 , and D_5 . Then, $D_5 = \{a, b, c\}$ evolves into $D_6 = \{a, b, c, e\}$ as they share $\{a, b, c\}$ as persistent members. Then, the evolution continues until D_{18} , which evolves into smaller D_{19} and so on.

Definition. 6.6. Closed Sequence of w -convoys – A sequence of w -convoy D_1, D_2, \dots, D_z such that each D_i evolves into $D_{(i+1)}$ for $1 \leq i < z$ is closed if there

is no w -convoy D' , which evolves into D_1 or into which D_z evolves into.

Following Def. 6.6, we can see that in Fig. 6.3, there is a single closed sequence of 27 w -convoys, from $D_1 = \{a, b, c\}$ to $D_{27} = \{b, c, e\}$. Informally, for each closed sequence of w -convoys in a Trajectory Database, there is a corresponding evolving convoy (defined below) that covers all related convoys (even convoys, whose duration are longer than w). For example, in Fig 6.3, the evolving convoy corresponding to the closed sequence of w -convoys has three stages — stage $S_1 = \{a, b, c\}$, stage $S_2 = \{a, b, c, e\}$, and stage $S_3 = \{b, c, e\}$. In this way, information of convoys become more comprehensive.

Definition. 6.7. Evolving Convoys – For given parameters m , k , w and a closed sequence of w -convoys, D_1, D_2, \dots, D_z such that each D_i evolves into $D_{(i+1)}$ for $1 \leq i < z$, the corresponding **Evolving Convoy (EVOCO)** V contains $z' \leq z$ stages. Each stage $S_V(j)$, for $1 \leq j \leq z'$, is defined as a continuous sequence of w -convoys $[D_s, D_e]$ having the same set of members.

Definition 6.7 ensures a sequence of related dynamic convoys to be covered in an evolving convoy with stages. The members at each stage covering w -convoys $[D_s, D_e]$ are the members of D_s . The start-time and end-time of a stage $S_{[s,e]}$ that covers w -convoys $[D_s, D_e]$ can be derived from the start-time and end-time of participating w convoys. For example the period of the first stage S_1 of the evolving convoy V is from t_1 to t_5 , while the period of the last stage S_3 of V is from t_{22} to t_{31} . Each stage corresponds to at least a dynamic convoy and, from an evolving convoy, the dynamic convoys it covers can be derived by a brute-force approach.

Evolving convoys have interesting properties. By Def. 6.7, an evolving convoy can gain (lose) new (existing) members throughout its life, i.e. any non-member object can become a member (and vice versa). Moreover, evolving convoys allow a persistent member in the current stage to become a dynamic member in subsequent

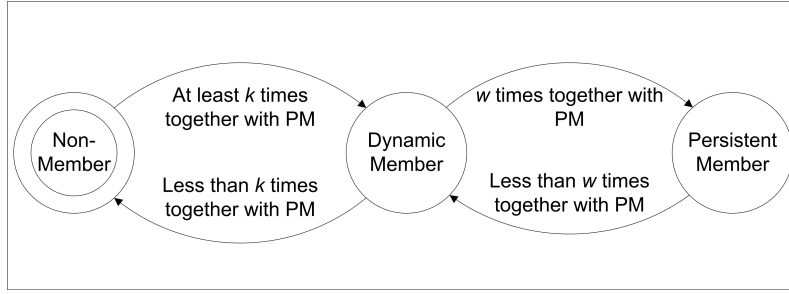


Figure 6.4: Transition between Membership in an Evolving Convoy.

stages (and vice versa) and impose no hard differentiation on a member's role, i.e. Any member can become a dynamic member (persistent member) and leave (form) the main body of the convoy. Figure 6.4 depicts a state-transition diagram outlining the memberships and transition between each pair of memberships in an evolving convoy. A non-member can become a dynamic member by staying at least k times together with the set of persistent members of the convoy during the previous w period. A dynamic member, if it stays together with PM w times during the previous w period, will become a persistent member. A persistent (dynamic) member becomes a dynamic member (non-member) if it does not stay together w (k) times with the persistent members during the previous w period.

The fact that a member in an evolving convoy changes his role (dynamic membership/persistent membership) agrees with real-life scenarios. Consider a group of five soldiers each, in turn, taking a point-duty, i.e to walk ten meter ahead of the group looking for anomalies, depicted in Fig. 6.5. Although this is a convoy moving for 50 time-stamps, there is no fixed set of persistent-members defining the main body from t_1 to t_{50} for any $m > 1$. However, with parameters $w = 20$, $k = 10$, and $m = 2$, we can detect this particular collective movements as an evolving convoy as the soldiers returning from point-duty will take the role of persistent-member and form the main body of the convoy. For example, b is a dynamic-member earlier and, eventually, a persistent-member at t_{21} onwards. Thus, a and b together formed the main body of the evolving convoy during $[t_{41}, t_{50}]$.

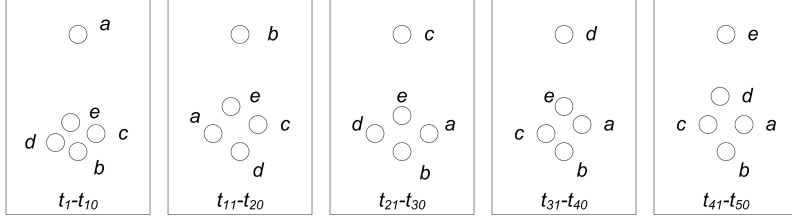


Figure 6.5: A Visualization of the Example of Five Soldiers' Movements.

To summarize this section, following the traditional notions of convoys, there are many overlapping convoys, the relationship among which is hard to establish, in a Trajectory Database. Therefore, the new concept of evolving convoys, which allow a convoy to evolve from a stage to the next, provides better picture of the real-life groups of moving objects than the traditional definitions.

Definition. 6.8. Discovery of Evolving Convoys – Given a Trajectory Database \mathcal{R} , DBSCAN parameters ε and min_pts , and constraints m , k , and w , Discovery of Evolving Convoys (DEC) is defined as finding evolving convoys.

6.3 Algorithms to Discover of Evolving Convoys

We developed three algorithms to discover evolving convoys from a Trajectory Database. The first algorithm is a straight-forward implementation while the next two algorithms mitigate the performance bottlenecks of their predecessors.

6.3.1 Simple Slice-by-slice Algorithm

The Simple Slice-by-Slice algorithm (S^3) is directly obtained from the problem definition and is similar to MC2 in [33] and CMC in [32]. It obtains density-connected clusters in each time-stamp in the Trajectory Database. Each cluster is treated as a potential candidate and S^3 tries to verify if it actually forms a convoy by checking clusters in subsequent time-stamps.

Details of S^3 is shown in Algorithm 6.1. If the Trajectory Database \mathcal{R} has

missing records, the function SNAP performs linear interpolation to fill the gaps in \mathcal{R}' — it gives a snapshot of \mathcal{R} at time-stamp t by reporting the location of each tracked objects at t . EXTEND is a function that tracks *count* or the number of times an object $o \in O$ is found with the PM_V for each convoy V and assigns different role to each member o for a convoy, effectively implementing the state-transition diagram in Fig. 6.4. In this way, it maintains a log of stages for each convoys so that the stages of the evolving convoys can be returned. The internals of the sub-routine EXTEND is shown in Algorithm 6.2.

Algorithm 6.1 Simple Slice-by-Slice Algorithm (S^3) to Discover Evolving Convoys.

Input: \mathcal{R} , ε , *min_pts*, m , k and w .

Output: A set of **Evolving Convoys** \mathcal{V} .

```

1:  $\mathcal{V} \leftarrow \phi$  and  $\mathcal{V}_{\text{cur}} \leftarrow \phi$ 
2: for  $t = 1$  to  $\tau$  do
3:   Snapshot  $\mathcal{R}' \leftarrow \text{SNAP}(\mathcal{R}, t)$ 
4:   Set of Clusters  $\mathcal{L} \leftarrow \text{DBSCAN}(\mathcal{R}', \varepsilon, \text{min\_pts})$ 
5:    $\text{match}(L) \leftarrow \text{false}$  for all  $L \in \mathcal{L}$ 
6:   for all  $V \in \mathcal{V}_{\text{cur}}$  do
7:      $\text{extended} \leftarrow \text{false}$ 
8:     for all  $L \in \mathcal{L}$  such that  $|L \cap PM_V| \geq m$  do
9:        $\text{EXTEND}(V, L)$ ,  $\text{match}(L) \leftarrow \text{true}$ , and  $\text{extended} \leftarrow \text{true}$ 
10:    if  $\text{extended} = \text{false}$  then
11:       $t_{\text{end}}(V) \leftarrow (t - 1)$  and  $\mathcal{V}_{\text{cur}} \leftarrow \mathcal{V}_{\text{cur}} - \{V\}$ 
12:      if  $t_{\text{end}}(V) - t_{\text{start}}(V) + 1 \geq w$  then
13:         $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$ 
14:    for all  $L \in \mathcal{L}$  such that  $\text{match}(L) = \text{false}$  and  $|L| \geq m$  do
15:      Create new convoy  $V$  with  $PM_V \leftarrow L$  and  $t_{\text{start}}(V) \leftarrow t$ 
16:       $\mathcal{V}_{\text{cur}} \leftarrow \mathcal{V}_{\text{cur}} \cup \{V\}$ 
17: for all  $V \in \mathcal{V}_{\text{cur}}$  such that  $\tau - t_{\text{start}}(V) + 1 \geq w$  do
18:   Set  $t_{\text{end}}(V) \leftarrow \tau$  and  $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$ 

```

For each time-stamp t , S^3 obtains a complete snapshot \mathcal{R}' using SNAP (line 3) and all the objects in \mathcal{R}' are clustered using DBSCAN (line 4). S^3 tries to match each of the current convoys maintained in \mathcal{V}_{cur} with the clusters found in the current timestamp (lines 6 - 9). If m or more persistent-members of a convoy V is found in a cluster C , V is matched to C — we say V “*extends*” to C . For each evolving convoy V matched to a cluster L , its member objects are tracked by $\text{EXTEND}(V, L)$ (line

Algorithm 6.2 Sub-routine EXTEND Used in Simple Slice-by-Slice Algorithm.

Input: V, L, m, k and w .

```
1: for all  $o \in O$  do
2:    $count \leftarrow$  no. of times  $o$  is density-connected to
      $PM_V$  in  $[t - w + 1, t]$ 
3:   if  $count = w$  then
4:      $PM_V \leftarrow PM_V \cup \{o\}$  and  $DM_V \leftarrow DM_V - \{o\}$ 
5:   else
6:     if  $count \geq k$  then
7:        $PM_V \leftarrow PM_V - \{o\}$  and  $DM_V \leftarrow DM_V \cup \{o\}$ 
8:     else
9:        $PM_V \leftarrow PM_V - \{o\}$  and  $DM_V \leftarrow DM_V - \{o\}$ 
```

9). When a matching cluster cannot be found, the convoy is put in the results if its life-span is at least w (lines 12-13). Those clusters L , into which no convoy has extended, are made potential convoys and placed in \mathcal{V}_{cur} (lines 14-16).

The S^3 algorithm reports a split when it detects a convoy V extends to more than one clusters. It can also detect merges using a test – conducted in each time-stamp — which checks whether two convoys, V and V' , have the same set of persistent-members.

Figure 6.3.1 shows movement of eight objects in nine time-stamps. For $min_pts = 2$, $m = 2$, $k = 3$ and $w = 4$, this Trajectory Database contains two evolving convoys, shown in shaded areas, namely V_1 and V_2 . A partial trace following convoy V_1 is listed in Tab. 6.2. At $t = t_1$, S^3 found a cluster $C_{1,1} = \{a, b, c, d\}$, which is made as a potential convoy V_1 and put into \mathcal{V}_{cur} . In subsequent time-stamp $t = t_2$, DBSCAN returns a cluster $C_{2,1} = \{a, b, c\}$, into which the potential convoy V_1 extends to because $C_{2,1}$ has $3 \geq m$ objects common with PM_{V_1} and PM_{V_1} becomes $\{a, b, c\}$. For each object o , EXTEND tracks $count$ for or the number of times it appeared with PM_{V_1} in $[t - w + 1, t]$, thus, for example, at t_2 , $count$ for $a = 2$ and $count$ for $d = 1$. In this way, at $t = t_4$, existence of convoy V_1 starting from t_1 is confirmed (d and e are not included as $count$ for $d = 1 < k$ and $count$ for $e = 2 < k$). In time-stamp $t = t_5$, DBSCAN returns two clusters $C_{5,1} = \{a, c, e\}$

and $C_{5,2} = \{d, f, g\}$. $V1$ extends to cluster $C_{5,1}$, and, since $count$ for $e = 3 \geq k$, e is noted to be joining the convoy. Since no convoy extends to cluster $C_{5,2}$, a potential convoy $V2 = \{d, f, g\}$ is put into \mathcal{V}_{cur} .

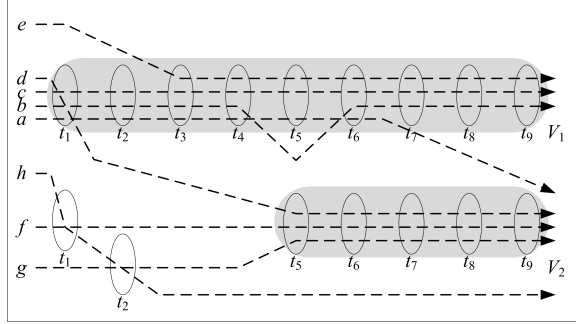


Figure 6.6: A Trajectory Database of Eight Objects' Movements.

Table 6.2: A Partial Trace of the Simple Slice-by-Slice (S^3) Algorithm Following the Convoy V_1 in Fig. 6.3.1.

t	PM_{V_1}	DM_{V_1}	$count$ of						$log(V_1)$
			a	b	c	d	e	f	
t_1	$\{a, b, c, d\}$	-	1	1	1	1	0	0	
t_2	$\{a, b, c\}$	-	2	2	2	1	0	0	
t_3	$\{a, b, c\}$	-	3	3	3	1	1	0	
t_4	$\{a, b, c\}$	-	4	4	4	1	2	0	Stage $S_1 = \{a, b, c\}$.
t_5	$\{a, c\}$	$\{b, e\}$	4	3	4	0	3	0	Stage $S_2 = \{a, b, c, e\}$. b became DM.
t_6	$\{a, c, e\}$	$\{b\}$	4	3	4	0	4	0	e became PM.
t_7	$\{c, e\}$	$\{a, b\}$	3	3	4	0	4	0	a became DM.
t_8	$\{c, e\}$	$\{b\}$	2	3	4	0	4	0	Stage $S_3 = \{b, c, e\}$
t_9	$\{b, c, e\}$	-	1	4	4	0	4	0	b became PM.

6.3.2 Interleaved DEC Algorithms

In the Simple Slice-by-Slice algorithm (S^3), all objects in each time-stamp are clustered using DBSCAN, which is an expensive operation. Therefore, for better performance, we need to minimize DBSCAN calls. TRAJ-DBSCAN (DBSCAN for trajectories) proposed in [32] uses the closest distance between each trajectories as their distance. It has a property that if objects o_1 and o_2 are density-connected at t ($t \leq t \leq t + \lambda - 1$), their trajectories j_1 and j_2 for $[t, t + \lambda - 1]$ are in the

same trajectory-cluster returned by TRAJ-DBSCAN. Therefore, in order to prune objects which may not form a cluster in a given time-stamp t ($t \leq t \leq t + \lambda - 1$), we borrowed TRAJ-DBSCAN to check the trajectory-clusters for trajectories in $[t, t + \lambda - 1]$.

For better performance, TRAJ-DBSCAN works on trajectories simplified by DP-simplification [32]. DP-simplification uses a parameter δ to reduce the number of points to represent a trajectory by allowing an error less than δ .

Our proposed Interleaved-DEC (ID) algorithms divide the Trajectory Database into partitions, each containing λ consecutive time-stamps. For each partition, Interleaved-DEC (ID) algorithms operates in two steps — the first is to get the set of objects which likely to form convoys while the second is actual clustering of objects and extending of the convoys. ID algorithms, therefore, interleave the two steps (hence their names) as they progress. We will collectively call the ID algorithms as “ID-Family” The length of each partition (λ) and trajectory simplification parameter (δ) can be set independently.

The First Interleave DEC Algorithm (ID-1)

The first interleaving algorithm, ID-1, is a simple extension of S^3 . A sketch of ID-1 is shown in Algorithm 6.3. The function $\text{PARTITION}(\mathcal{R}, p, \lambda)$ returns the p^{th} λ -length partition from Trajectory Database \mathcal{R} . Selective SNAP – $\text{S_SNAP}(\mathcal{P}, J, t)$ – returns the data of the given set of objects J at t . For each partition \mathcal{P} , the trajectories are clustered using TRAJ-DBSCAN (lines 3-5). For each trajectory-cluster J found in current partition, only its members are clustered in each-timestamp t (lines 6-8) saving clustering efforts. Matching the clusters found in each time-stamp against the set of current convoys and initiating un-matched clusters as potential convoys are same as S^3 (lines 9-20).

ID-1 brings performance improvement over S^3 by clustering a handful of objects

Algorithm 6.3 The First Interleave DEC Algorithm (ID-1) to Discover Evolving Convoys.

Input: \mathcal{R} , ϵ , min_pts , m , k and w .

Output: A set of **Evolving Convoys** \mathcal{V} .

```

1:  $\mathcal{V} \leftarrow \phi$  and  $\mathcal{V}_{cur} \leftarrow \phi$ 
2: for  $t = 1$  to  $\tau$  do
3:   if  $mod(t - 1, \lambda) = 0$  then
4:      $p \leftarrow t/\lambda + 1$  and  $\mathcal{P} \leftarrow \text{PARTITION}(\mathcal{R}, p, \lambda)$ 
5:     Set of Trajectory Clusters  $\mathcal{J} \leftarrow \text{TRAJ-DBSCAN}(\mathcal{P}, \epsilon, min\_pts\delta)$ 
6:     for  $J \in \mathcal{J}$  do
7:       Snapshot  $\mathcal{R}' \leftarrow \text{S\_SNAP}(\mathcal{P}, J, t)$ 
8:       Set of Clusters  $\mathcal{L} \leftarrow \mathcal{L} \cup \text{DBSCAN}(\mathcal{R}', \epsilon, min\_pts)$ 
9:        $match(L) \leftarrow \text{false}$  for all  $L \in \mathcal{L}$ 
10:      for all  $V \in \mathcal{V}_{cur}$  do
11:         $extended \leftarrow \text{false}$ 
12:        for all  $L \in \mathcal{L}$  such that  $|L \cap PM_V| \geq m$  do
13:           $\text{EXTEND}(V, L)$ ,  $match(L) \leftarrow \text{true}$ , and  $extended \leftarrow \text{true}$ 
14:          if  $extended = \text{false}$  then
15:             $t_{end}(V) \leftarrow (t - 1)$  and  $\mathcal{V}_{cur} \leftarrow \mathcal{V}_{cur} - \{V\}$ 
16:            if  $t_{end}(V) - t_{start}(V) + 1 \geq w$  then
17:               $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$ 
18:          for all  $L \in \mathcal{L}$  such that  $match(L) = \text{false}$  and  $|L| \geq m$  do
19:            Create new convoy  $V$  with  $PM_V \leftarrow L$  and  $t_{start}(V) \leftarrow t$ 
20:             $\mathcal{V}_{cur} \leftarrow \mathcal{V}_{cur} \cup \{V\}$ 
21:          for  $V \in \mathcal{V}_{cur}$  such that  $\tau - t_{start}(V) + 1 \geq w$  do
22:             $t_{end}(V) \leftarrow \tau$  and  $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$ 

```

each time-stamp. For example, in Fig. 6.3.1, if we set $\lambda = 2$, in partition $[t_3, t_4]$, only trajectories of a , b , c and e will form trajectory-clusters while those of d , f , g and h will not. Therefore, in time-stamps t_3 and t_4 , ID-2 needs to cluster only four objects in contrast to S^3 clustering eight objects each time-stamp.

The Second Interleave DEC Algorithm (ID-2)

Although ID-1 is expected to have better performance than S^3 , it is still costly because the pruning is not tight enough and have false positives, which must be checked and removed by the slice-by-slice loop that follows. For instance, in the scenario shown in Fig. 6.3.1, trajectories of objects f , g and h form a trajectory cluster in partition $[t_1, t_2]$ as the closest distance between h and f (g) was small as

they meet in t_1 (t_2). ID-1 must, therefore, try to cluster them for 2 time-stamps (t_1 and t_2) without finding a single cluster (and convoy) they form. We also noted that if a convoy is verified to exist from t_i to $t_{i+\lambda-1}$, its members can be excluded from TRAJ-DBSCAN, i.e. clustering the trajectories of a , c and e in partition $[t_5, t_6]$ is a waste if their convoy is verified to exist up to t_6 .

We developed another interleaving algorithm ID-2 to have tighter pruning than ID-1 and exploit the fact we noted to save trajectory-clustering efforts. The skeleton of the second interleaving algorithm, ID-2, is given in Algorithm 6.4. For each partition \mathcal{P}_p , ID-2 first tries to extend the current convoys in \mathcal{V}_{cur} and those which failed to extend until end of \mathcal{P}_p are put into results (line 4). Objects which are not persistent-members of any current convoy (verified up to end of \mathcal{P}_p) can form new convoys. Therefore, new convoys are formed out of them and put into the list of current evolving convoys \mathcal{V}_{cur} (line 5).

Algorithm 6.4 The Second Interleave DEC Algorithm (ID-2) to Discover Evolving Convoys.

Input: \mathcal{R} , ε , min_pts , m , k and w .

Output: A set of **Evolving Convoys** \mathcal{V} .

- 1: $\mathcal{V} \leftarrow \phi$, $\mathcal{V}_{\text{cur}} \leftarrow \phi$ and $N \leftarrow w/\lambda$
 - 2: **for** $p = 1$ to $\tau/\lambda + 1$ **do**
 - 3: Partition $\mathcal{P}_p \leftarrow \text{PARTITION}(\mathcal{R}, p, \lambda)$
 - 4: $\mathcal{V} \leftarrow \mathcal{V} \cup \text{S_VERIFY}(\mathcal{P}_p, \mathcal{V}_{\text{cur}}, \varepsilon, min_pts, m, k, w)$
 - 5: $\mathcal{V}_{\text{cur}} \leftarrow \mathcal{V}_{\text{cur}} \cup \text{NEW_CONVOY}(p, \mathcal{V}_{\text{cur}}, \varepsilon, min_pts, m, k, w)$
 - 6: **for all** $V \in \mathcal{V}_{\text{cur}}$ such that $\tau - t_{\text{start}}(V) + 1 \geq w$ **do**
 - 7: Set $t_{\text{end}}(V) \leftarrow \tau$ and $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$
-

Since, a current convoy V can only extend to a cluster L containing at least m of its persistent members (PM_V), S_VERIFY, shown in Algorithm 6.5, uses the persistent-members of the convoy PM_V as a guide to build the clusters it can extend. In each time-stamp t in the given partition \mathcal{P}_p , clusters containing persistent-members of evolving convoy $V \in \mathcal{V}_{\text{cur}}$ are formed by S_DBSCAN (lines 3 - 4). Then, the convoys are extended to the clusters found (lines 6-9) and those convoys, which

cannot extend anymore are returned.

Algorithm 6.5 Sub-routine S_VERIFY Used in the Second Interleave DEC Algorithm.

Input: \mathcal{P}_p , \mathcal{V}_{cur} , ε , min_pts , m , k and w .

Output: A set of **Evolving Convoys** \mathcal{V} .

```

1: for  $t = t_{\text{start}}(\mathcal{P}_p)$  to  $t_{\text{end}}(\mathcal{P}_p)$  do
2:    $\mathcal{L}_t \leftarrow \phi$ 
3:   for all  $V \in \mathcal{V}_{\text{cur}}$  do
4:     Set of Clusters  $\mathcal{L}_t \leftarrow \mathcal{L}_t \cup \text{S\_DBSCAN}(\mathcal{R}, PM_V, t, \varepsilon, \text{min\_pts})$ 
5:     Set  $\text{match}(L) \leftarrow \text{false}$  for all  $L \in \mathcal{L}_t$ 
6:     for all  $V \in \mathcal{V}_{\text{cur}}$  do
7:       Set  $\text{extended} \leftarrow \text{false}$ 
8:       for all  $L \in \mathcal{L}_t$  such that  $|L \cap PM_V| \geq m$  do
9:          $\text{EXTEND}(V, L)$  and  $\text{match}(L) \leftarrow \text{true}$  and  $\text{extended} \leftarrow \text{true}$ 
10:      if  $\text{extended} = \text{false}$  then
11:         $t_{\text{end}}(V) \leftarrow t - 1$ ,  $\mathcal{V}_{\text{cur}} \leftarrow \mathcal{V}_{\text{cur}} - \{V\}$  and  $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$ 

```

S_DBSCAN is a modified version of DBSCAN that returns all the valid clusters in \mathcal{R} at t containing all objects $o \in PM_V$ according to DBSCAN parameters ε and min_pts . S_DBSCAN is built around the fact that any object o can be either a core point or a border point in a density cluster (or a noise). Since a density-cluster can be built starting with any of its core points, the given object will be used as the seed to recursively expand its cluster if it is a core point. On the other hand, if it is a border point, one of its ε -neighbor must be a core point. Therefore, its ε -neighbors are used as seeds to build the clusters containing them. For example, if we ask S_DBSCAN to find clusters containing $\{b, c\}$, in Fig. 3.1, it will return L_1 and L_3 (but not L_2). While c can be directly used as the seed to build L_1 , a , a neighbor of b is used to construct L_3 as b is not a core-point.

Since it is possible that new convoy formed from the clusters that S_DBSCAN leaves to build (for example L_2 from Fig. 3.1), NEW_CONVOY (see Algorithm 6.6) uses a filter-refinement scheme to initiate new convoys. In order to avoid finding convoys already existed, only trajectories, which are not persistent-members of any existing convoys, are clustered (lines 1-2). Trajectory-clusters are verified if they

can form a set of persistent members across $N = w/\lambda$ partitions (or w time-stamps) to get convoy-candidates (lines 3 - 9). The set of convoy candidates \mathcal{D} is used as a guide to find the set of convoys $\mathcal{V}'_{\text{cur}}$, whose start-time is in partition $\mathcal{P}_{(p-N+1)}$ (line 10). VERIFY is similar to S_VERIFY except it uses \mathcal{D} to find cluster instead of \mathcal{V}_{cur} to find density-connected cluster (see line 3 of S_VERIFY) and it initiates the un-matched clusters as potential convoys (which S_VERIFY does not). Convoys in $\mathcal{V}'_{\text{cur}}$ are extended by S_VERIFY until end of partition $\mathcal{P}_{(p)}$ (lines 11 - 12).

Algorithm 6.6 Sub-routine NEW_CONVOY Used in the Second Interleave DEC Algorithm.

Input: $p, \mathcal{V}_{\text{cur}}, \varepsilon, \text{min_pts}, m, k$ and w .

Output: A set of **Evolving Convoys** V'

- 1: Partition $\mathcal{P}'_p \leftarrow \mathcal{P}_p - \{\langle o, t, loc \rangle \mid \text{there is } V \in \mathcal{V}_{\text{cur}} \text{ such that } o \in PM_V\}$
 - 2: Set of Trajectory Clusters $\mathcal{J}_p \leftarrow \text{TRAJ-DBSCAN}(\mathcal{P}'_p, \varepsilon, \text{min_pts}, \delta)$
 - 3: $\mathcal{D} \leftarrow \mathcal{J}_{(p-N+1)}$
 - 4: **for** $T = p - N + 1$ to p **do**
 - 5: $\mathcal{D}' \leftarrow \phi$
 - 6: **for all** $D \in \mathcal{D}$ **do**
 - 7: **for all** $J \in \mathcal{J}_T$ such that $|J \cap D| \geq m$ **do**
 - 8: $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{J \cap D\}$
 - 9: $\mathcal{D} \leftarrow \mathcal{D}'$
 - 10: $\mathcal{V}'_{\text{cur}} \leftarrow \phi$ and Call VERIFY($\mathcal{P}_{(p-N+1)}, \mathcal{D}, \mathcal{V}'_{\text{cur}}, \varepsilon, \text{min_pts}, m, k, w$)
 - 11: **for** $T = p - N + 2$ to p **do**
 - 12: Call S_VERIFY($\mathcal{P}_T, \mathcal{V}'_{\text{cur}}, \varepsilon, \text{min_pts}, m, k, w$)
-

Here is an illustration of ID-2 on Trajectory Database in Fig. 6.3.1. For the first two partitions $[t_1, t_2]$ and $[t_3, t_4]$, there is no existing convoys in \mathcal{V}_{cur} . To find new convoys of at least $w = 4$ period, trajectory-clusters from those partitions are examined. In the first partition, there are two trajectory-clusters, $\{a, b, c\}$ and $\{f, g, h\}$. However, the second partition has only one trajectory cluster $\{a, b, c, e\}$. Thus, slice-by-slice verification is done only for the convoy containing persistent-members $\{a, b, c\}$ and put it into the set of existing convoys. In the third partition $[t_5, t_6]$, the existing convoy $\{a, b, c\}$ is extended until t_6 (e became a persistent-member, b became dynamic-member). Since the convoy containing $\{a, b, c, e\}$ is verified to exist up to t_6 , trajectories of its members are left in TRAJ-DBSCAN

operation and a trajectory cluster containing d , f and g is formed. In the fourth partition $[t_7, t_8]$, existing convoy is extended up to t_8 and a new convoy $\{d, f, g\}$ is formed.

tab

6.4 Experimental Evaluations

6.4.1 Preliminary Experiments

Before we present our experiments on convoy discovery algorithms, we will give a summary of the preliminary investigation we conducted in order to assess the performance of convoy discovery algorithms in the absence of high performance spatial-temporal index. We have the preliminary experiment results published in [6] and reproduce the full report in Appendix A.

In a streaming Trajectory Database setting, when only rudimentary spatial-temporal index built in ad hoc manner is available and location update rate is predictable/fast, S^3 cannot scale well with the size of dataset compared to our proposed ID-Family. ID-1 can be used when we want convoys of short-duration or when few false positives are expected — a false positive means a trajectory cluster forms for a time-partition but no density-connected cluster is found in any time-stamp during the time-partition. ID-2 is suitable for many scenarios.

We also learnt good settings for parameters δ and λ as they have impact on performance (but not correctness) of the ID-Family. Although δ can be set independently, we recommend to set δ low (our experiences suggest lower than half of ε) to have tighter bound since higher δ values will increase the false-positives and, hence, running time. Yet, λ is not an independent variable as it determines how often the user gets the reports as all information of convoys in a λ -partitions \mathcal{P} are reported in bulk only after \mathcal{P} has been read in. Therefore, users would want to

set λ as low as possible. We observed that the lower the value of λ , the better ID algorithms perform. However, the running time rises when λ is set such that each partition includes only 1-2 movement record for each object, putting more overheads in TRAJ-DBSCAN operations. Therefore, from our studies, we recommend setting a low λ value as long as each object reports its location 3 or more times in a given length of partition (λ).

6.4.2 Experiment Setup

We implemented our proposed algorithms in Java along with X-CuTS, a naive extension of CuTS [32]. X-CuTS, like original CuTS³, essentially divides the Trajectory Database into partitions (of λ time-stamps each), performs TRAJ-DBSCAN to filter objects, which will not form convoys, and performs verification. However, λ values for X-CuTS must be greater than $w - k$ in order to prevent its pruning mechanism from pruning dynamic members (by joining trajectory clusters across partitions), which can be away from the convoy up to $w - k$ time-stamps. We ran X-CuTS with $\lambda = w/2$ when $k = 0.60 \times w$.

For all sets of experiments, the interval between each consecutive time-stamps was set at 10 seconds but no pre-processing was done for missing records. The parameters used for the experiments were selected intuitively. For example, the distance between walking/commuting humans (ε) in the same convoy was 10/15 meter while that of moving taxis was 100 metre. Details of the parameters used for each dataset is given in Tab. 6.3.

The main distinguishing feature of these sets of experiments is that they used permanently-built indexes to support range queries (described immediately below) while the preliminary experiments in Sect. 6.4.1 used only rudimentary low-performance ad-hoc indexes.

³Refer to Sect. 3.2.4 for more information of CuTS

Table 6.3: Parameters Used to Assess Convoy Discovery Algorithms.

Dataset	m	w	k	ε	min_pts	λ	δ
Statefair	2	90	60	10	2	12	3
NCSU	3	90	60	10	3	12	3
New York	3	90	60	15	3	12	3
Orlando	3	90	60	15	3	12	3
KAIST	3	90	60	10	3	12	3
SF-Cabs21	3	90	60	100	3	12	10
SF-Cabs21	3	90	60	100	3	12	10

Index Used to Support Range Queries

The range query operation heavily used in DBSCAN was supported by a 3D grid index built in the following manner. First, we divided the spatial-space of the Trajectory Database (\mathcal{R}) into $p \times q$ equal-sized squares and, for each square, we created a cube for each t_{grid} time-range, i.e. the spatial-temporal space of the \mathcal{R} was divided into p rows, q columns, and $\lceil \tau/t_{\text{grid}} \rceil$ cubes, where τ is the number of time-stamps in \mathcal{R} . Then, we associated each cube with a unique number (cid) in such a way that cubes having same time-range have consecutive numbers. Finally, for each line-segment $\langle loc, loc_{\text{next}} \rangle$ and each of the cube it passes, an entry $\langle pkey, cid \rangle$ was inserted into the index, where loc_{next} is defined as the next loc an object o is going to visit after a particular record in \mathcal{R} and $pkey$ is the primary key of the record in \mathcal{R} . In essence, the 3D grid index is similar to B^x -tree [27], which is proven to be suitable for large Trajectory Databases.

We determined the ideal values for p , q , and t_{grid} by minimizing a linear combination of the average number of cube per $pkey$ (it is directly proportional to the storage requirement of the index) and the average number of $pkey$ per cube (it is directly proportional to the number of data-access/computation-units each query makes after accessing the index):

$$cost(p, q, t_{\text{grid}}) = \alpha \cdot \frac{1}{n_{pkey}} \cdot \sum_{pkey \in \mathcal{R}} x_{pkey} + (1 - \alpha) \frac{1}{n_{cube}} \cdot \sum_{cube \in \text{index}} x_{cube},$$

where n_{pkey} is the number of records in the Trajectory Database, x_{pkey} is the

Table 6.4: Datasets and Index Settings Used by the Convoy Discovery Algorithms.

Dataset	Map-size	p	q	t_{grid} (in seconds)
Statefair	1.1 x 1.0 km ²	384	343	120
NCSU	14.6 x 9.7 km ²	384	255	120
New York	31.5 x 19.5 km ²	384	239	120
Orlando	15.4 x 17.9 km ²	331	384	120
KAIST	32.7 x 24.9 km ²	384	292	120
SF-Cabs21	47.2 x 85.9 km ²	141	256	150
SF-Cabs22	45.7 x 79.7 km ²	147	256	150

number of cubes the $pkey$ is associated, n_{cube} is the number of (non-empty) cubes in the index, x_{cube} is the number of $pkey$ in each (non-empty) cube, and $0 \leq \alpha \leq 1$. For our experiments, we set $\alpha = 0.5$. We sampled the values of $cost(p, q, t_{\text{grid}})$ by varying p and t_{grid} , if the length (East-West length) of the map is longer than its breadth (North-South length), and by varying q and t_{grid} otherwise. We used sample values of $cost(p, q, t_{\text{grid}})$ taken from NCSU and New York to determine index properties ($max(p, q$ and $t_{\text{grid}})$) for all five human datasets and those taken from SF-Cabs21rand100, for SF-Cabs21 and SF-Cabs22. The size of the spatial-map and the properties of indexed used are outlined in Tab. 6.4.

6.4.3 Results and Analysis

Table 6.5 shows a comparison of running time (in seconds) of the algorithms to find evolving convoys for each dataset. ID family (ID-1 and ID-2) always outperformed S³ as ID algorithms prune many of the objects from clustering, which S³ must inadvertently perform. Unlike the preliminary experiments, in which only rudimentary spatial-temporal index was used and ID-2 was consistently better than ID-1, in these sets of experiments, ID-2 was only slightly better than ID-1 since the permanent index we built benefited ID-1 much dwarfing ID-2’s tighter pruning, which saved clustering and verification efforts by filtering out objects coming near the convoy for a short period of time (during the filtering step in NEW_CONVOY

sub-routine). X-CuTS performed worse than S^3 in Statefair dataset. Although X-CuTS performed better than S^3 in other human-movement datasets, it did not return a complete answer in these datasets due to the accidentally linked convoys as described in Sect. 3.2.4. Therefore, we omitted it from experiments using taxi datasets and further sets of experiments.

Table 6.5: Running Time and Results of Convoy Discovery Algorithms.

Dataset	Run-time (seconds)				No. of Convoys
	S^3	ID-1	ID-2	X-CuTS	
Statefair	52	18	17	61	2
NCSU	258	158	144	218	13
New York	255	162	141	225	1
Orlando	229	135	127	226	1
KAIST	234	138	130	191	18
SF-Cabs21	101	50	46	–	36
SF-Cabs22	120	59	55	–	41

Figure 6.7 shows how the parameters w affected the performance of the algorithms in New York and KAIST datasets. S^3 took much longer to finish processing compared to the ID-Family. ID-2 slightly outperformed ID-1. All algorithms S^3 , ID-1, and ID-2 were not affected much by changing w value.

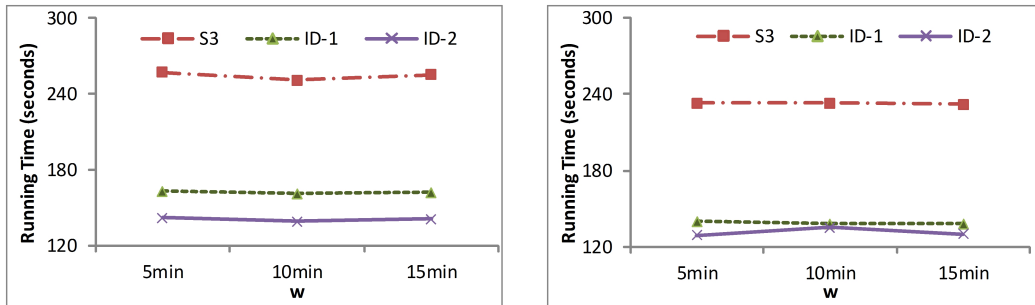


Figure 6.7: Impact of Parameter w on Performance of Convoy Discovery Algorithms in (a) New York Using Parameters, $min_pts = m = 3$, $k = \lceil 0.06 * w \rceil$, and $\varepsilon = 15$ Metre, and (b) KAIST Using Parameters, $min_pts = m = 3$, $k = \lceil 0.06 * w \rceil$, and $\varepsilon = 10$.

Figure 6.8 shows how the parameters k affected the performance of the algorithms in New York and KAIST datasets. Consistent to previous sets of experiments, S^3 took longer processing time than the members of the ID-Family did. ID-2

outperformed ID-1 although the difference is less prominent for KAIST dataset. ID-Family, ID-1 and ID-2, was not affected much by changing k value but S^3 took slightly longer for $k = 54$ in New York dataset compared to the amount it took for other k values in the same dataset. It was, we hypothesize, due to the random delays in I/O (such as disk seek etc) as more state-transition (from dynamic member to non-member and vice versa) happened for this k value — lower/higher k values make convoys have more/less dynamic members but they both have less dynamic member to non-member transitions and vice versa — prompting the algorithms to write more results to the disk (each transition is a part of the output).

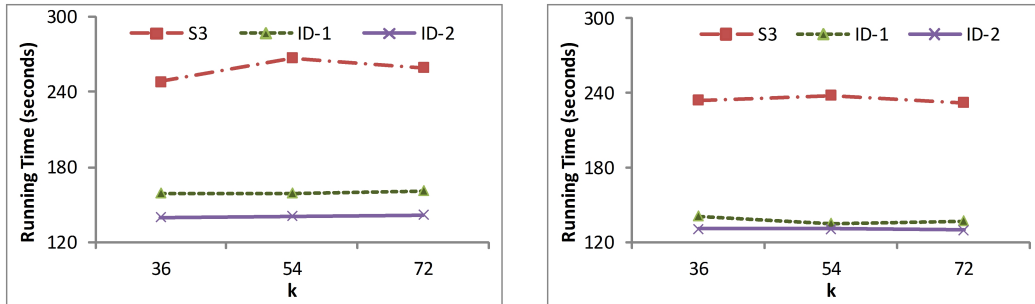


Figure 6.8: Impact of Parameter k on Performance of Convoy Discovery Algorithms in (a) New York Using Parameters, $min_pts = m = 3$, $w = 15$ minute (90 time-stamps), and $\varepsilon = 15$ Metre, and (b) KAIST Using Parameters, $min_pts = m = 3$, $w = 15$ minute (90 time-stamps), and $\varepsilon = 10$ Metre.

In the subsequent sets of experiments, we assess how the DBSCAN parameters min_pts and ε affected the performance of the algorithms using New York and KAIST datasets. Figure 6.9 shows the effect of the parameter min_pts had on the convoy discovery algorithms, where ID family consistently outperformed S^3 . ID-2 ran faster than ID-1 but the performance difference between ID-1 and ID-2 was not huge. Increasing min_pts means fewer and/or smaller clusters in each time-stamp and, hence, brought shorter running time for algorithms in the ID-Family although S^3 was not much affected by changing min_pts .

Figure 6.10 shows the effect of the parameter ε had on the convoy discovery

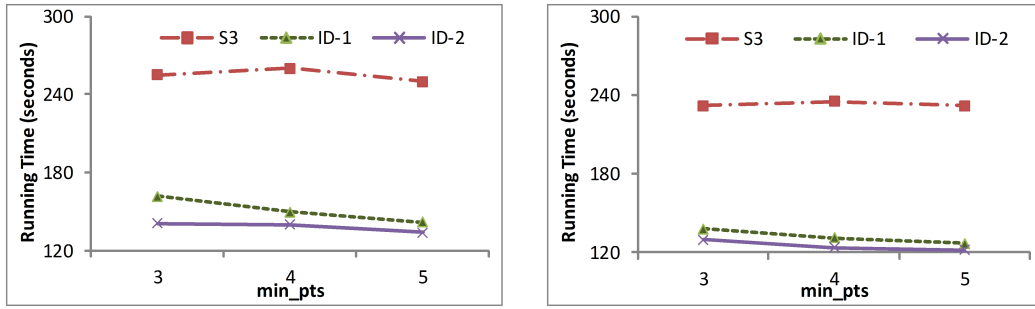


Figure 6.9: Impact of Parameter min_pts on Performance of Convoy Discovery Algorithms in (a) New York Using Parameters, $w = 15$ minute (90 time-stamps), $k = 60$, and $\varepsilon = 15$ Metre, and (b) KAIST Using Parameters, $w = 15$ minute (90 time-stamps), $k = 60$, and $\varepsilon = 10$ Metre.

algorithms. In these sets of experiments, ID family outperformed S^3 , with ID-2 being faster than ID-1. Increasing ε means more clusters and/or larger clusters are found in each time-stamps. This, in New York dataset, increases pruning time for ID-1, while decreases the overall running time of S^3 slightly as the pruning mechanism of ID-1 became less effective (could not reduce the clustering effort) while S^3 found fewer number of clusters in each time-stamp leading to less joining time. However, for KAIST dataset, changing ε values slightly increased the running time of all algorithms indicating that the number of convoys in the result changed slightly compared to the size of the dataset (8 convoys are found for $\varepsilon = 10$ and 18 are found for $\varepsilon = 10$).

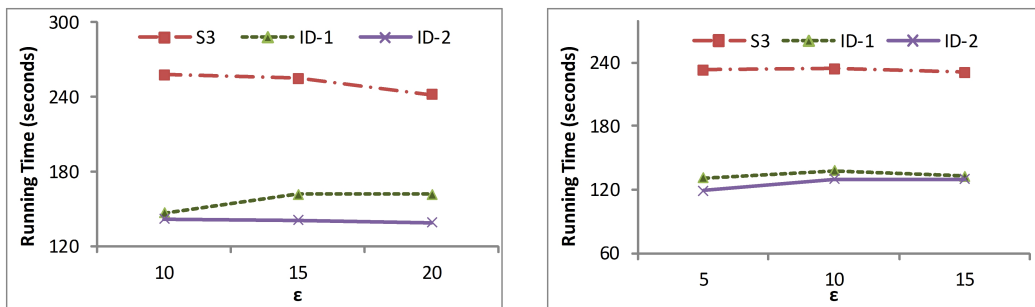


Figure 6.10: Impact of Parameter ε on Performance of Convoy Discovery Algorithms in (a) New York and (b) KAIST Using Parameters, $min_pts = m = 3$, $w = 15$ minute (90 time-stamps), and $k = 60$.

To summarize the experiment results, S^3 cannot scale well with the size of

dataset. In the absence of high-performance spatial-temporal index, i.e. for the streaming Trajectory Databases, we learnt from the preliminary experiments that, ID-1 can be used when we want convoys of short-duration or when fewer false positives are expected although ID-2 offers a greater performance gain regardless of the duration of convoys and the number of false positive. In the presence of a high-performance index, i.e. off-line Trajectory Database (a data warehouse), both ID-1 and ID-2 give reasonable increase in throughput compared to S^3 . Although ID-2's performance gain may not be as prominent in off-line setting as its performance in streaming setting, it still performs better than ID-1. Moreover, since ID-2 has three steps (verification step using S-VERIFY, filtering step in NEW_CONVOY, and verification step in NEW_CONVOY) while ID-1 only has two steps (filtering step using TRAJ-DBSCAN and the immediate verification step that follows), theoretically, running these steps in a pipe-line architecture, i.e. in parallel, would increase the overall throughput of ID-2 significantly more than that of ID-1.

6.5 Summary

In this chapter, we presented and proposed new and practical convoy definitions and proposed algorithms to find them. Dynamic convoys allow members to briefly move away based on user-defined constraints. Evolving convoys present the sequences of dynamic convoys evolving into one another in a more comprehensive result for human users. All three proposed algorithms can report details of evolving convoys. They work in an incremental manner suitable for both off-line discovery and on-line convoy discovery.

Conclusion

Along with the advances in GPS technology, users are able to maintain a highly accurate (up to a few meters) record of the locations the tracked objects visited in high temporal resolutions. Deployment of GPS technologies in various fields including shipping and port industries, public transport industries, sciences, and other personal usages has lead to exploding number and sizes of Trajectory Databases (TJDBs) available for mining instances of Multi-object Movement Patterns. Knowledge of such patterns has numerous applications in epidemiology, ecology, preservation of wild-life, traffic monitoring and control, Location-Based Services, marketing, social-studies, and even on-line game development.

We have noticed that there were many limitations in the existing data mining and knowledge discovery techniques to discover instances of Multi-object Movement Patterns. Experimental studies on the algorithms were scarce for finding meeting patterns in a TJDB. There were still limited amount of works for discovering frequent routes from Trajectory Databases without any prior knowledge of the underlying spatial regions. Previous works existed when we started our studies for this thesis on finding convoy patterns cannot handle real-life convoys, which have members occasionally moving away from their parent groups (and coming back) as well as new members joining and/or existing members leaving the convoy in different stages of the convoys' life-spans. In addition, a TJDB becomes larger as GPS and storage technologies advance. Therefore, efficient and effective mining of

TJDBs for the instances of the Multi-object Movement Patterns had been a new and interesting challenge.

7.1 Contributions

We had conducted research on three groups of patterns – (a) meeting patterns (b) sub-trajectory clique patterns to find frequent routes, and (c) dynamic convoy and evolving convoy patterns. For the first and second patterns, namely meeting patterns and sub-trajectory clique patterns, we studied how to extract their instances in off-line Trajectory Databases. For the third group of patterns, namely the dynamic convoy and evolving convoy patterns, we proposed techniques to discover them in an incremental manner — thus, our proposed techniques are applicable to both off-line Trajectory Databases and on-line Trajectory Data-streams, which are more widely available nowadays. A summary of the contribution our thesis made is as follows.

7.1.1 Finding Closed MEMOs

We defined a new model of meeting pattern called MEMO pattern and developed three algorithms to find its instances in Trajectory Database. We developed three novel data-driven algorithms to discover closed MEMOs. We had conducted experiments on real-life datasets, which showed that our proposed algorithms can perform better than the existing one in finding instances of MEMO patterns.

7.1.2 Mining Sub-trajectory Cliques to Extract Frequent Routes

We had proposed to find cliques of sub-trajectories and extract a frequent routes from each sub-trajectory cliques. To measure similarity between sub-trajectories, we used Fréchet distance, which follows the curvature of the trajectories, is invariant

in speed, and is direction-aware. Our proposed method to extract a frequent route through sub-trajectory cliques do not require any prior knowledge of the spatial space or road network.

We designed two pragmatic approximation algorithms. In the experiments we conducted using real-life datasets, both algorithms performed reasonably faster and provided better accuracy compared to existing polynomial time algorithm. They also provided more intuitive results than those provided by the existing partition and group framework, Traclus. The second algorithm we proposed is a divide and conquer algorithm, which sub-divides the input Trajectory Database into subsets so that an instance of the first algorithm can process each subset independently. It runs slower than the first one yet provides opportunities for parallelism and/or memory efficiency.

7.1.3 Discovery of Evolving Convoys

We had observed real-life nature of convoys and proposed new and practical convoy definitions called dynamic convoy pattern and evolving convoy. Dynamic convoys allow dynamic members, which temporarily move away from their parent convoy, while evolving convoys are able to capture different stages of a convoy's evolution using dynamic convoys (of fixed duration) as its building blocks. Therefore, evolving convoys present the convoys in a more comprehensive result for human users. We presented three algorithms to extract instances of evolving convoys from a Trajectory Database. All three proposed algorithms work in an incremental manner suitable for both off-line discovery and on-line monitoring.

7.2 Future Works

We conclude this thesis by identifying future research directions and a brief discussion on them.

7.2.1 Unified Framework for MOMO Patterns

This thesis presented techniques to discover instances of some Multi-object Movement Patterns from off-line and streaming Trajectory Databases. In addition, we can still identify other types of interesting movement patterns and devise techniques to find them. However, all the techniques developed (and will be developed for newer types of patterns) can discover instances of a single type of pattern (although parametrizable by users).

Along this reasoning, we question whether it is possible to develop a unified framework that can find a multitude of patterns at the same time. We believe it is a possible and pragmatic research direction as we had witnessed relationships between certain types of patterns — all meetings are convoys¹, which do not move; all convoys correspond to sub-trajectory cliques; and most importantly, existence of meeting instances slow down sub-trajectory clique mining.

Hence, we propose to continue identifying newer patterns and devise a unified framework for finding MOMO Patterns. The ultimate goal of this direction is, “given a Trajectory Database, the framework automatically discovers instances of all types of patterns in the data and present it in a human understandable form (or summarize the whole Trajectory Database in terms of pattern instances found) to the users.”

¹not necessarily defined using density-connection as proximity measure

7.2.2 Check-in and Social-network Data

During the recent years, we witnessed the advent of social networks and, more recently, location-based social networks. These social networks produce location-based data (check-in data), which users log as they visit interesting locations. These data are essentially incomplete trajectory data (without detailed routes). The availability of the check-in data-streams are increasing rapidly.

We also propose to identify and discover interesting patterns in such incomplete trajectory data — check-in data-streams. This direction of research has commercial applications such as targeted advertising and improving the services of social-networks. However, given the size and growth of the check-in data-streams and other available social network data, the challenge is not a trivial one.

REFERENCES

- [1] www.rtreeportal.org.
- [2] *Porcupine Caribou Herd Satellite Collar Project*.
<http://www.taiga.net/satellite/>, 2008.
- [3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.
- [4] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. In *In Proceedings of IPPS/SPDP Workshop on High Performance Data Mining*, 1998.
- [5] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.
- [6] H. H. Aung and K.-L. Tan. Discovery of evolving convoys. In M. Gertz and B. Ludäscher, editors, *SSDBM*, volume 6187 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 2010.
- [7] H. H. Aung and K. L. Tan. Finding closed MEMOs. In *Proceedings of the 23rd international conference on Scientific and statistical database management, SSDBM'11*, pages 369–386, Berlin, Heidelberg, 2011. Springer-Verlag.
- [8] H. H. Aung and K.-L. Tan. Mining multi-object spatial-temporal movement patterns. *SIGSPATIAL Special*, 4(3):14–19, 2012.
- [9] R. D. Balicer. Modeling infectious diseases dissemination through online role-playing games. *Epidemiology*, 18(2):260–261, Mar. 2007.
- [10] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Comput. Geom. Theory Appl.*, 41(3):111–125, 2008.

- [11] M. Booth. The AI system of left 4 dead. Artificial Intelligence and Interactive Digital Entertainment Conference 2009. Keynote.
- [12] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. In *Proceedings of the 19th International Symposium on Algorithms and Computation, ISAAC '08*, pages 644–655, Berlin, Heidelberg, 2008. Springer-Verlag.
- [13] M. Celik, S. Shekhar, J. P. Rogers, and J. A. Shine. Mixed-drove spatiotemporal co-occurrence pattern mining. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1322–1335, 2008.
- [14] J. Chen, C. Lai, X. Meng, J. Xu, and H. Hu. Clustering moving objects in spatial networks. In *DASFAA*, pages 611–623, 2007.
- [15] K.-T. Chen, A. Liao, H.-K. K. Pao, and H.-H. Chu. Game bot detection based on avatar trajectory entertainment computing - ICEC 2008. In S. M. Stevens and S. J. Saldamarco, editors, *Entertainment Computing - ICEC 2008*, volume 5309 of *Lecture Notes in Computer Science*, chapter 11, pages 94–105. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.
- [16] L. D. Drager, J. M. Lee, and C. F. Martin. On the geometry of the smallest circle enclosing a finite set of points. *Journal of the Franklin Institute*, 344(7):929 – 940, 2007.
- [17] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the frèchet distance for realistic curves in near linear time. In *Proceedings of the 2010 annual symposium on Computational geometry, SoCG '10*, pages 365–374, New York, NY, USA, 2010. ACM.
- [18] A. Dumitrescu and G. Rote. On the frèchet distance of a set of curves. In *CCCG*, pages 162–165, 2004.
- [19] J. Elzinga and D. W. Hearn. Geometrical Solutions for Some Minimax

- Location Problems. *TRANSPORTATION SCIENCE*, 6(4):379–394, 1972.
- [20] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *VLDB '98: Proceedings of the 24th International Conference on Very Large Data Bases*, pages 323–333, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [21] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.
- [22] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 330–339, New York, NY, USA, 2007. ACM.
- [23] 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*, pages 35–42, New York, NY, USA, 2006. ACM.
- [24] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. pages 73–84, 1998.
- [25] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM.
- [26] S.-Y. Hwang, Y.-H. Liu, J.-K. Chiu, and E.-P. Lim. Mining mobile group patterns: A trajectory-based approach. In T. B. Ho, D. W.-L. Cheung, and H. Liu, editors, *PAKDD*, volume 3518 of *Lecture Notes in Computer Science*,

- pages 713–718. Springer, 2005.
- [27] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient b+-tree based indexing of moving objects. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 768–779. Morgan Kaufmann, 2004.
- [28] J. G. Jetcheva, Y. chun Hu, S. Palchaudhuri, A. Kumar, S. David, and B. Johnson. Design and evaluation of a metropolitan area multitier wireless ad hoc network architecture. pages 32–43, 2003.
- [29] J. G. Jetcheva, Y.-C. Hu, S. PalChaudhuri, A. K. Saha, and D. B. Johnson. CRAWDAD data set rice/ad_hoc_city (v. 2003-09-11). Downloaded from http://crawdad.cs.dartmouth.edu/rice/ad_hoc_city, Sept. 2003.
- [30] H. Jeung, H. T. Shen, and X. Zhou. Convoy queries in spatio-temporal databases. In *ICDE*, pages 1457–1459, 2008.
- [31] H. Jeung, M. L. Yiu, and C. S. Jensen. Trajectory pattern mining. In Y. Zheng and X. Zhou, editors, *Computing with Spatial Trajectories*, pages 143–177. Springer, 2011.
- [32] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proc. VLDB Endow.*, 1(1):1068–1080, 2008.
- [33] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, pages 364–381, 2005.
- [34] G. Karypis, E.-H. S. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling, 1999.
- [35] S. Kashyap, S. Roy, M.-L. Lee, and W. Hsu. Farm : Feature-assisted aggregate route mining in trajectory data. In Y. Saygin, J. X. Yu, H. Kargupta, W. Wang, S. Ranka, P. S. Yu, and X. Wu, editors, *ICDM Workshops*, pages 604–609. IEEE Computer Society, 2009.

- [36] D. Kline. Bringing interactive storytelling to industry: Designing a reactive narrative encounter system. In C. Darken and G. M. Youngblood, editors, *AIIDE*. The AAAI Press, 2009.
- [37] J. G. Lee, J. Han, and K. Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 593–604, New York, NY, USA, 2007. ACM.
- [38] 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*, pages 617–622, New York, NY, USA, 2004. ACM.
- [39] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: mining relaxed temporal moving object clusters. *Proc. VLDB Endow.*, 3:723–734, Sept. 2010.
- [40] Z. Li, J. Han, M. Ji, L. A. Tang, Y. Yu, B. Ding, J. G. Lee, and R. Kays. MoveMine: Mining moving object data for discovery of animal movement patterns. *ACM Trans. Intell. Syst. Technol.*, 2, July 2011.
- [41] E. T. Lofgren and N. H. Fefferman. The untapped potential of virtual game worlds to shed light on real world epidemics. *The Lancet Infectious Diseases*, 7(9):625–629, Sept. 2007.
- [42] M. Morzy. Mining frequent trajectories of moving objects for location prediction. In P. Perner, editor, *MLDM*, volume 4571 of *Lecture Notes in Computer Science*, pages 667–680. Springer, 2007.
- [43] R. T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Trans. on Knowl. and Data Eng.*, 14(5):1003–1016, 2002.
- [44] H.-K. Pao, H.-Y. Lin, K.-T. Chen, and J. Fadlil. Trajectory based behavior analysis for user verification intelligent data engineering and automated

- learning IDEAL 2010. volume 6283 of *Lecture Notes in Computer Science*, chapter 39, pages 316–323. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.
- [45] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. A Parsimonious Model of Mobile Partitioned Networks with Clustering. In *The First International Conference on COMMunication Systems and NETWORKS (COMSNETS)*, January 2009.
- [46] H. Rademacher and O. Toeplitz. The spanning circle of a finite set of points. *The Enjoyment of Mathematics : Selection from Mathematics for the Amateur*, pages 103–110, 1957.
- [47] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong. On the levy-walk nature of human mobility. In *Proceedings of the 27th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Arizona, USA, April 2008. IEEE.
- [48] I. Rhee, M. Shin, S. Hong, K. Lee, S. Kim, and S. Chong. CRAWDAD data set ncsu/mobilitymodels (v. 2009-07-23). Downloaded from <http://crawdad.cs.dartmouth.edu/ncsu/mobilitymodels>, July 2009.
- [49] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov.*, 2(2):169–194, 1998.
- [50] G. Schofield, C. M. Bishop, G. MacLeon, P. Brown, M. Baker, K. A. Katselidis, P. Dimopoulos, J. D. Pantis, and G. C. Hays. Novel gps tracking of sea turtles as a tool for conservation management. *Journal of Experimental Marine Biology and Ecology*, Article in, 2007.
- [51] C. C. Schwartz and S. M. Arthur. Radiotracking large wilderness mammals: integration of gps and agro technology. *Ursus*, 11:261–274, 1999.

- [52] S. M. Tomkiewicz, M. R. Fuller, J. G. Kie, and K. K. Bates. Global positioning system and associated technologies in animal behaviour and ecological research. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1550):2163–2176, July 2010.
- [53] M. R. Vieira, P. Bakalov, and V. J. Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *GIS*, pages 286–295, 2009.
- [54] Y. Wang, E.-P. Lim, and S.-Y. Hwang. Efficient algorithms for mining maximal valid groups. *The VLDB Journal*, 17(3):515–535, 2008.
- [55] T. U. Wien, T. Eiter, T. Eiter, H. Mannila, and H. Mannila. Computing discrete frchet distance. Technical report, 1994.
- [56] H. Yang, S. Parthasarathy, and S. Mehta. A generalized framework for mining spatio-temporal patterns in scientific data. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 716–721, New York, NY, USA, 2005. ACM.
- [57] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 443–454, New York, NY, USA, 2004. ACM.
- [58] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12:372–390, 2000.
- [59] M. Zhang, W. Hsu, and M. L. Lee. Finding orientation-sensitive patterns in snapshot databases. In *ICTAI '07: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Vol.2 (ICTAI 2007)*, pages 171–178, Washington, DC, USA, 2007. IEEE Computer Society.
- [60] Z. Zhou, W. Wu, X. Li, M.-L. Lee, and W. Hsu. Maxfirst for maxbrknn. In S. Abiteboul, K. Böhm, C. Koch, and K.-L. Tan, editors, *ICDE*, pages

828–839. IEEE Computer Society, 2011.

- [61] H. Zhu, J. Luo, H. Yin, X. Zhou, J. Z. Huang, and F. B. Zhan. Mining trajectory corridors using frèchet distance and meshing grids. In *Proceedings of the 14th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining - Volume Part I*, PAKDD'10, pages 228–237, Berlin, Heidelberg, 2010. Springer-Verlag.

Preliminary Experiments on Convoy Discovery

A.1 Experiment Setup

We implemented the algorithms in *C/C++* and tested on a Windows XP-Professional workstation equipped with Intel Core 2 Duo E6500 processor and 4GB RAM. The distance unit used in the experiments is metre. Two real-life datasets and a synthetic dataset used to evaluate the performance of the algorithms are:

- **Mob** — contains human movement in five different sites [47, 48]. In order to obtain more moving objects, we merge data from five sites by aligning their reference points, i.e. the origins $(0, 0)$ of the two dimensional spatial planes. We further divide the dataset into five-hour-periods and merge them to obtain 559 trajectories. It is notable that the update rate of each trajectory is strictly 30 seconds.
- **Bus** — contains bus movements during peak hours (0800-1600 hours) in Settle from 30-Oct-2001 to 05-Oct-2001 [29]. We merged the data into a single day, i.e. we removed the date information, to obtain a large dataset of 4,471 bus movement.
- **Synth** — is a synthetic dataset that is used to test the scalability of the algorithms. We maintained a total of ten thousand moving objects at any

Table A.1: Datasets and Experiment Settings Used to Assess Convoy Discovery Algorithms in Preliminary Experiments.

Dataset	Time-stamps	Records	Map	m	w/k	ε	min_pts	λ/δ
Mob	1,800	267,459	(40km) ²	3	90/54	3	3	10/1
Bus	2,880	1,000,579	(100km) ²	5	90/54	10	5	10/5
Synth	720	2,046,112	(10km) ²	5	90/54	3	3	10/3

time by spawning a new object for each object going out of the map (number of unique objects is 13,635). The initial positions of the objects and their velocities are randomly determined. The mean location update rate of 30% of the objects is 3 while that of the rest is 5. Moreover, there is 1% missing records introduced randomly. Based on a random variable, new convoys of randomly determined durations are artificially built out of existing objects.

For all sets of experiments, the interval between each consecutive time-stamps is set at 10 seconds but no pre-processing is done for missing records. The parameters used for the experiments are selected intuitively. For example, the distance between walking humans in the same convoy is 3 metre while that of moving buses is 10 metre. The range query operation heavily used in DBSCAN is supported by dividing the map into 100 equal-sized grids, i.e. 10 rows and 10 columns. This is a fair assumption in real-time setting (like streaming data), where building a high-performance spatial-index is out of the question. More information of the datasets and experiment settings are summarized in Table A.1.

Since each evolving convoys starts with a dynamic convoy, for comparison, we extend CuTS [32] into X-CuTS to find dynamic-convoys and include it in the first set of experiments. However, to prevent its pruning mechanism from pruning dynamic members λ values for X-CuTS must be greater than $w - k$. We run X-CuTS with $\lambda = w/2$ when $k = 0.60 \times w$.

A.2 Results and Analysis

Table A.2 shows a comparison of running time (in seconds) of the algorithms to find evolving convoys for each dataset. ID family (ID-1 and ID-2) always outperforms S^3 and X-CuTS as ID algorithms prune many of the objects from clustering, which S^3 must inadvertently perform. In general, ID-2 is better than ID-1 since ID-2 has a tighter pruning and saves clustering and verification efforts for objects, which accidentally came close for a short period of time. X-CuTS performs worse than S^3 in Bus and Synth datasets. Although X-CuTS performs better than S^3 in Mob dataset, it does not return a complete answer in Bus and Synth datasets. Therefore, we omitted its results in further discussions.

Table A.2: Running Time Comparison of Convoy Discovery Algorithms for Different Datasets in Preliminary Experiments.

Dataset	No. of Convoys	S^3	ID-1	ID-2	X-CuTS
Mob	10	174.39	137.41	113.00	156.959
Bus	153	1843.88	1765.95	1423.24	2470.83
Synth	6	1932.61	1852.75	1607.44	5127.87

Figure A.1 shows how the parameters w and k affect the performance of the algorithms in Mob dataset. Algorithm S^3 is not affected by changing w and k values. ID family is not affected by changing k value but ID-2 performs better for larger w value since it can prune evolving convoys of short-duration while S^3 and ID-1 cannot.

Figure A.2 shows how the DBSCAN parameters ε and min_pts affect the performance of the algorithms in Mob dataset. All algorithms are affected by changing ε and min_pts values. Increasing ε means more clusters and/or larger clusters are found in each time-stamps. This, in turn, increases pruning, clustering, and joining time. However, ID family benefits from the pruning steps while S^3 does not. Increasing min_pts means fewer and/or smaller clusters and, hence, shorter running

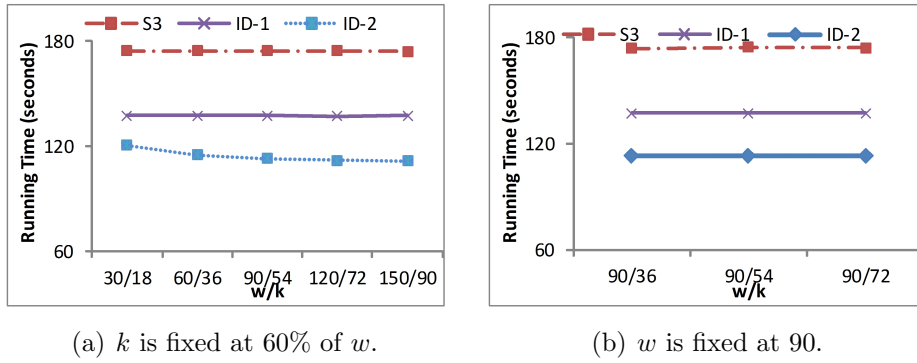


Figure A.1: Effect of Parameters w and k on Performance of Convoy Discovery Algorithms in Mob Dataset during Preliminary Experiments.

time. ID family out-performs S^3 , with ID-2 being the best.

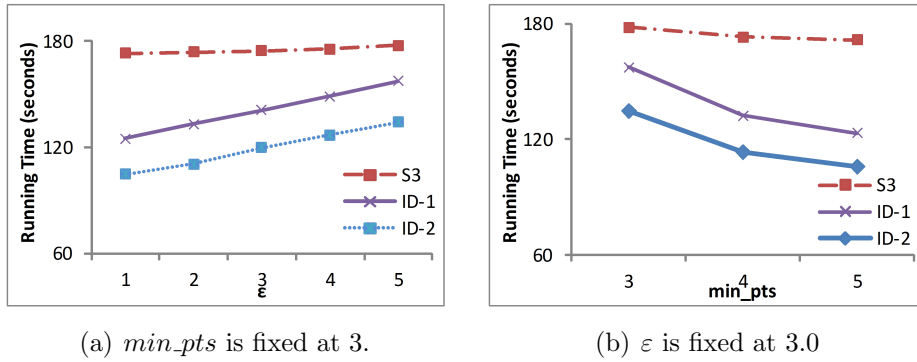


Figure A.2: Effect of DBSCAN Parameters ϵ and min_pts on Performance of Convoy Discovery Algorithms in Mob Dataset during Preliminary Experiments.

Parameters δ and λ do not affect the correctness of ID family but may have impact on performance. Although δ can be set independently, our preliminary studies showed that δ should be lower than half of ϵ to have tighter bound. Otherwise, higher δ values will increase the running time.

However, λ is not an independent variable as it determines how often the user get the reports as all information of convoys in a λ -partitions \mathcal{P} are reported in bulk only after \mathcal{P} has been read in. Therefore, user would want to set λ as low as possible. Figure A.3 shows the performance of the algorithms with different λ values. We observed that the lower the value of λ , the better ID algorithms perform. The running time rises when λ is set to 5 because the update rate for

Mob dataset is 3 (30 seconds), thus each partition includes 1-2 movement record for each object, putting more overheads in TRAJ-DBSCAN operations. Therefore, from our studies, we recommend setting a low λ value as long as each object reports its location 3 or more times in a given length of partition (λ).

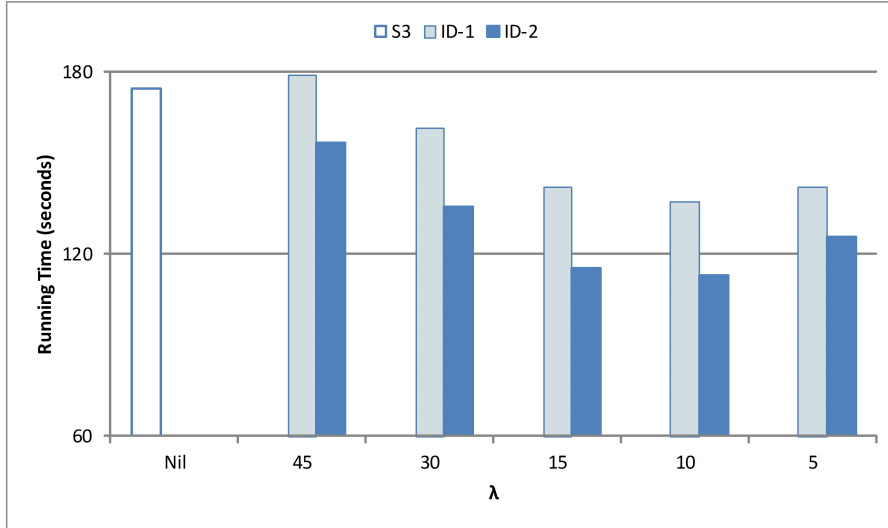


Figure A.3: Effect of Parameter λ on Performance of Convoy Discovery Algorithms in Mob Dataset during Preliminary Experiments.

In order to assess how the algorithms would perform when given more/less complete data, more experiments were conducted. Objects in Synth dataset is modified to have higher/lower update frequencies. Performance of S^3 , ID-1, and ID-2 are plotted in Fig. A.4(a). In general, lower update rates introduce lower I/O costs. However, this forces S^3 to perform more linear interpolations to predict locations of all the objects, reducing the saving in I/O. However, ID algorithms benefit as trajectory clustering time is reduced and they can prune much interpolation and clustering efforts.

More synthetic datasets (with 7,500 and 12,500 objects each) were generated to assess how the algorithms scale on different size of data. Figure A.4(b) shows the running time of each algorithm. ID algorithms outperform S^3 when the dataset contains more than 7,500 objects. ID-1 performs only slightly better than S^3 as its

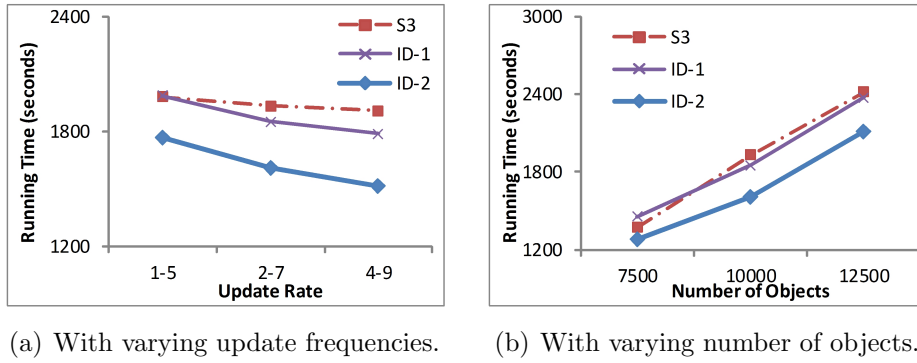


Figure A.4: Effect of the Nature of the Dataset on the Convoy Discovery Algorithms Assessed Using Synthetic Datasets during Preliminary Experiments.

pruning power is limited. It is found that ID-2 performs best and scales better than S^3 and ID-1.

Finally, we compared the moving clusters reported by MC2 [33] against the convoys our algorithms reported. MC2 often finds a set of shorter moving clusters instead of a single evolving convoy as the convoy’s members are often found in a cluster not similar to the one they were in the previous time-stamp (for example, a merge). In Mob and Bus data, 9 and 248 moving clusters (compared to 10 and 153 evolving convoys), which last for 90 time-stamps, are found respectively. Yet, they do not cover all the evolving convoys with the same duration because some convoys correspond to a set of disjoint moving clusters, some of whose duration are shorter than 90 time-stamps.

To summarize the experiment results, S^3 cannot scale well with the size of dataset. ID-1 can be used when we want convoys of short-duration or when few false positives are expected. ID-2 is suitable for many scenarios. By definition, evolving convoys are more compact and more expressive than moving clusters.