# Moving Objects Management for Location-Based Services


Guo Long

School of Computing

Computer Science Department

National University of Singapore

Supervisor: Kian-Lee TAN

A Thesis Submitted for the Degree of

*Doctor of Philosophy*

August 2015

I would like to dedicate this thesis to my beloved parents and wife for their endless love and encouragement.

# Declaration

I hereby declare that the 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.

GUO Long
5 August 2015

# Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Tan Kian-Lee, for his excellent guidance, patience and support throughout my study, and especially for providing me with an excellent atmosphere for doing research. I would also like to thank Prof. Li Mong-Lee and Prof. Roger Zimmermann for serving as members on my thesis committee.

I would like to express my very sincere gratitude to Dr. Zhang Dongxiang, for his guidance and support in my research, and for all the philosophy of life that I have learned from him. I am also thankful to my co-authors, Prof. Li Guoliang, Prof. Cong Gao, Prof. Shao Jie, Prof. Bao zhifeng, Dr. Wu Wei, Dr. Htoo Htet Aung and Chen Lu. They contribute a lot of work to improve our papers. It has been great to work together with them.

I would like to thank all my friends, for their understanding and encouragement in my moments of crisis. Their friendship makes my life a wonderful experience. I cannot list all the names here, but they are always on my mind.

I especially thank my parents and sister. My hard-working parents have sacrificed their lives for my sister and myself and provided unconditional love and care. They never put pressure on us and their only wish is that we can be happy. My elder sister have done her best to take care of me and shared my parents' burden for me. I love them so much, and I would not have made it this far without them.

The best outcome from these past five years is finding my best friend, soul-mate, and wife. I fell in love with her at first sight, and she lives in my heart from then. She was always there cheering me up and stood by me through the good times and bad. There are no words to convey how much I love her. I wish to send this thesis as a gift to her, which is only a beginning of our journey.

# Abstract

With the rapid development of GPS-enabled devices and wireless communication technologies, location-based services have attracted significant attention from both academic and industry communities. In this thesis, we focus on the management of moving objects data to make location-based services more intelligent in order to improve people's quality of life. Many interesting applications that target moving objects have great potential to bring revolutionary changes to people's life. However, there is an urgent call for efficient algorithms to support these applications, due to the explosion of geo-tagged information collected from various channels in this era. In this thesis, we have identified problems that are related to moving objects and have many interesting applications in location-based services, and proposed frameworks with efficient algorithms to solve these problems.

In particular, this thesis proposes three novel problems that deal with three types of spatial queries, respectively. First, we study the efficient processing of moving spatial keyword queries on road networks. Such queries consider both spatial locations and textual descriptions to find top $k$ best objects of interest. Most of the existing studies on this topic are restricted to the Euclidean space only. In many applications, position and accessibility of spatial-textual objects are constrained by network connectivity, and spatial proximity should be determined by the shortest path distance rather than the Euclidean distance. However, there is still no research effort on continuous monitoring of spatial keyword queries on road networks. We propose two frameworks that traverse the network in an incremental manner. Meanwhile, different techniques are designed for these two frameworks to avoid unnecessary traversing of some network edges.

Second, we study the efficient processing of moving spatial queries against dynamic event streams. In this problem, the server continuously monitors moving users subscribing to dynamic event streams, and notifies users

instantly when there is a matching event nearby. Past research on such problems either focused on how to handle incoming event streams efficiently by assuming users' locations are static or attempted to process continuous moving subscriptions against a static event dataset. Thus, we propose a novel location-aware pub/sub system named Elaps to support moving spatial queries against dynamic event streams for the first time. Elaps employs the concept of safe region to reduce the communication overhead. In Elaps, we develop a new concept called impact region that allows us to identify whether a safe region is affected by newly arrived events. Moreover, we propose a novel cost model to optimize the safe region size to keep the communication overhead low. Based on the cost model, we design two incremental methods for safe region construction. In addition, Elaps uses boolean expression, which is more expressive than keywords, to model user intent and we propose a novel index to handle spatial boolean expression matching.

Third, we study the efficient processing of optimal trajectories queries for influence maximization. Such queries find $k$ best trajectories to be attached with a given advertisement and maximizes the expected influence of the advertisement among a large group of audience. We are the first to extend the classic influence maximization problem to the location-aware advertising field. We show that the problem is NP-hard and propose both exact and approximate solutions to find the best set of trajectories. In the exact solution, we devise an expansion-based framework that enumerates combinations of trajectories in a best-first manner. In addition, we propose two effective methods for the upper bound estimation to facilitate early termination. To support large $k$, we propose three approximate methods with performance guarantees. In particular, we propose a greedy method and an improved cluster-based approach, both with an approximation ratio of $(1 - 1/e)$. We also propose a threshold method that can support any approximation ratio in $(0, 1]$. In addition, we extend our problem to support influence maximization for a group of advertisements.

For each framework proposed in the thesis, we conduct extensive experiments in realistic settings with both real and synthetic datasets. These experiments reveal the effectiveness and efficiency of the proposed frameworks.

**Keywords:** Moving objects, location-based services, spatial trajectory, location-based queries, moving spatial keyword queries, road network, moving range queries, dynamic event streams, optimal trajectories queries, influence maximization, location-aware advertising, performance evaluation.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

## 1.1 Motivations

Advances in location positioning and wireless communication technologies have given rise to the prevalence of location-based services (LBS), leading to a myriad of location data representing the mobility of a variety of moving objects, such as people and vehicles. Such moving objects data has two characteristics. On the one hand, it is real time. The location of an object can be obtained either by using the Global Positioning System (GPS) or by using the mobile communication network in real time. An application scenario is the moving spatial queries, where people want to be notified of interesting events nearby by providing his/her locations in real time in a certain period. On the other hand, it is traceable. In this case, the mobility of moving objects is typically represented in the form of spatial trajectories. For instance, in Flickr[1], a series of geo-tagged photos can form a spatial trajectory as each photo has a location tag and a time-stamp corresponding to where and when the photo was taken. Likewise, the "check-ins" of a user in Foursquare[2] can be regarded as a trajectory, when sorted chronologically. In addition, many taxis in major cities have been equipped with a GPS sensor, which enables them to report a time-stamped location to a data center with a certain frequency. Such reports contribute to a large amount of spatial trajectories. Overall, the ability to record the moving objects have offered us unprecedented information to understand the value hidden in data generated by the moving objects and great opportunity to explore its broad applications. As such, lots of location-based services (LBS) have been studied to improve people's life quality.

[1]https://www.flickr.com/
[2]https://foursquare.com/

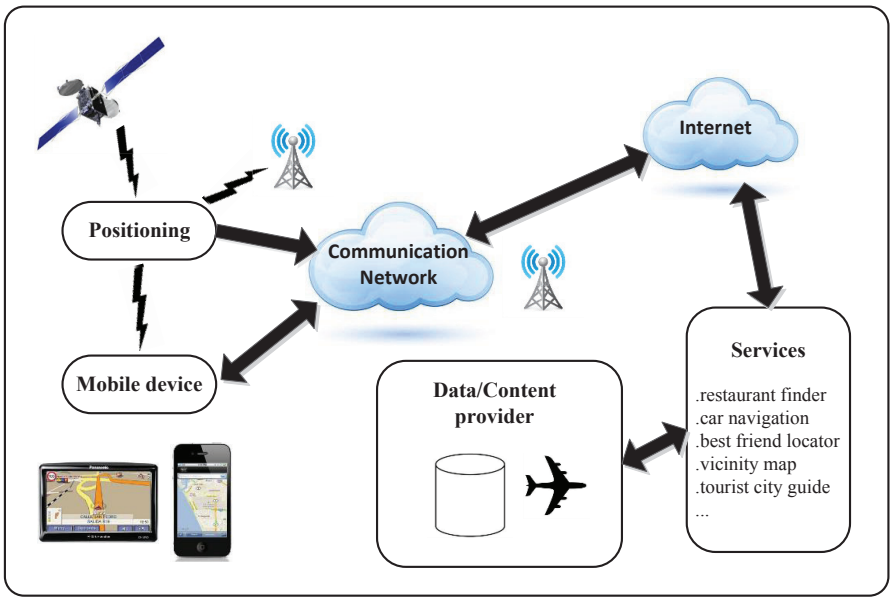



Figure 1.1: LBS components and information flow.

Location-based services are an important class of context-aware applications, which answer location-related queries, where a location is either explicit or implicit. Figure 1.1 presents the basic infrastructures required in the location-based services and their connections. The mobile clients (e.g., pedestrians or vehicles) receive their current GPS locations from the satellites and update their locations to the server via the communication network (e.g., 3G network or WiFi). The server can provide a variety of services to the user and is responsible for the service request processing. Such services offer the functionality of recommending a tourist attraction, finding a route, or searching specified information based on the user interest and so forth. Take the nearest neighbor queries for example, this service can provide the user the nearest object/person around him. In this thesis, we focus on the management of the moving objects data to make the location-based services more intelligent.

From the server's perspective, moving objects data can be classified into two categories: real-time data and historical data, corresponding to its two characteristics. For some applications, moving objects data continuously stream into the server that in turn uses the data to process real-time queries, such as moving range queries or nearest neighbour queries. For some other applications, the increasing number of location-aware devices has resulted in the accumulation of a large amount of trajectory data that capture the movement histories of a variety of moving objects, which enable the server to answer even more sophisticated queries, such as route recommendation. In this thesis, we study three different kinds of location-based queries from these two

aspects, introduced as follows.

The first query addressed in this thesis is the efficient processing of moving spatial keyword queries on road networks. With the rapid development of GPS-enabled mobile devices, there is a prevalent trend that web content is increasingly being geotagged, such as messages on Facebook and Twitter. Spatial keyword queries, which find best points of interest in terms of both spatial proximity to query location and textual relevance to query keywords, adapt well to this trend, and thus, become a new interest of the research community. Spatial keyword queries are also being supported in real-life applications, such as Google Maps where points of interest can be retrieved, Foursquare where geo-tagged documents can be retrieved, and Twitter where tweets can be retrieved. On the one hand, most of the existing studies on this topic [83, 14, 86, 27, 23, 82, 74, 75, 61, 80] are restricted to Euclidean space only. In many applications, position and accessibility of spatial-textual objects are constrained by network connectivity, and spatial proximity should be determined by shortest path distance (rather than Euclidean distance, which may be inappropriate especially in urban areas). On the other hand, Rocha-Junior and Nørvåg [62] studied static top-$k$ spatial keyword queries on road networks and proposed algorithms conducted in the spatial and textual domains alternately to constrain the search region. However, until now there has been no research effort on continuous monitoring of spatial keyword queries on road networks. In location-based services, besides snapshot queries that are evaluated only once, continuous moving queries are also very appealing, since queries can be evaluated on dynamic settings more realistically. Motivated by these two observations, we design a framework to efficiently process moving top-$k$ spatial keyword queries when query and objects of interest are on a road network.

The second query addressed in this thesis is the efficient processing of moving spatial queries against dynamic event streams. The prevalence of social networks and mobile devices has facilitated the real-time dissemination of local events such as sales, shows and exhibitions. To notify users interesting events nearby, various commercial location-aware pub/sub systems have been proposed, such as Foursquare's push notifications and Apple's iBeacon. Location-aware pub/sub is also receiving increasing interest in the research community. These fall into two categories: they either focused on how to handle incoming event streams efficiently by assuming users' locations are static [6, 50, 13, 24]; or they attempted to process continuous moving subscriptions against a static event dataset [84, 35, 5, 75, 37, 32]. None of them can really support subscriptions from mobile users moving all the time against spatial events that are continuously published by local businesses. Therefore, we propose a

new location-aware pub/sub system, Elaps, that continuously monitors moving users subscribing to dynamic event streams from social media and E-commerce applications and notifies them instantly when there is a matching event nearby.

The third query addressed in this thesis is the efficient processing of optimal trajectories queries for influence maximization. Influence Maximization is a key algorithmic problem behind online viral marketing. By word-of-mouth propagation effect among friends, it finds a seed set of $k$ users to maximize the expected influence among all the users in a social network. It has attracted significant attention from both academic and industry communities due to its potential commercial value, such as viral marketing [9, 30, 39], rumor control and information monitoring [25, 40, 60]. In this work, we study a novel problem of influence maximization in trajectory databases to explore the applications of advertising on moving objects. Given a trajectory database and a group of audience attached with spatial-temporal patterns, for an advertisement, our goal is to find $k$ trajectories for moving vehicles to maximize the influence of the advertisement among the audience. Several applications can benefit from the trajectory influence maximization problem. For instance, the trajectory influence maximization problem is useful in moving advertisement management. Nowadays, mobile advertisement (e.g., bus or subway) offers advertisers the opportunity to reach consumers seamlessly as they spend more time commuting out of home. With the help of the trajectory influence maximization problem, mobile advertisement can be well placed to reach consumers on the go, delivering advertising messages to the right audience at the right time. Besides that, the trajectory influence maximization problem also have applications in route recommendation systems. Consider $k$ vehicles which carry some advertisement for a promotion activity (e.g., a presidential election or a beauty pageant) waiting for route navigation. Given a trajectory database, we can issue a top-$k$ query and get $k$ trajectories for the vehicles, which can maximize the influence of the promotion activity. With the help of the trajectory influence problem, location-aware advertising can be made more intelligent.

There is great linkage among the three pieces of works. The first work only deals with moving spatial queries to provide moving users with static events nearby. The second work extends the first work by considering moving spatial queries against dynamic event streams, where users can receive much more rich information nearby. While the first two works focus on the management of real-time locations of moving users to improve the location-based services, the third work further improves the location-based services from another level by utilizing the trajectories of moving

vehicles and the past movement histories of users to provide better location-aware advertising.

## 1.2 Challenges

Location based queries (LBQ) provide support for location based services. In general, LBQ faces two main challenges, which are described as follows.

- How to satisfy the real time requirement. The increasing number of geo-tagged information means that a naive execution of LBQ can be inefficient. This translates to long response time which may not be acceptable for real time applications. In order to solve this challenge, efficient indexing techniques has to be proposed to save the query time.

- How to reduce the communication cost for the moving LBQ. The difficulties arise from the need of keeping the answer to moving queries up to date while optimizing the wireless communications. If we consider a moving query as just a sequence of instantaneous queries that are periodically evaluated, we will probably not be able to refresh the answer with the required frequency due to the high communication cost. Thus, some new techniques are needed to process continuous moving LBQ efficiently.

In the following, we present the challenges in each of the thesis research work in order to reduce the computation cost and the communication cost.

### 1.2.1 Moving Spatial Keyword Queries on Road Networks

Although continuous moving queries with network distance can provide up-to-date and accurate results as a query point moves, it is often costly to monitor such moving queries. A straightforward solution is to traverse the road network to find the top-$k$ results every time the query is evaluated. However, such a scheme will lead to lots of unnecessary repetitive traversing of network edges and lots of unnecessary communication with the server, resulting in high computation cost and communication cost. Many proposals for moving spatial keyword queries in Euclidean space such as [55, 12, **?**, 37] unitize the concept of safe region. In response to a query, its result can be computed together with a safe region within which the result remains valid. Only when the query exits its safe region will a new query (additional processing) be invoked, which repeats the above process. In this way, both the computation cost and

communication cost can be reduced significantly. However, query processing on road networks is fundamentally different compared with the query processing in Euclidean space due to the constraint of the network connectivity. Thus, a different scheme should be proposed to tackle the challenge induced by the road network. In addition, compared with previously proposed continuous moving spatial queries on road networks [22, 20, 54] where only the spatial information of the objects of interest are considered, how to exploit the opportunities of simultaneously using spatial proximity and textual relevancy for joint pruning brings new challenges to query optimization of our problem. Chapter 3 shows how these challenges are tackled.

### 1.2.2 Moving Spatial Queries against Dynamic Event Streams

There are two types of roles in this problem, subscribers and publishers. Subscribers can submit his/her subscription in the form of boolean expressions and specify a notification region. When a subscriber moves, the notification region moves along. On the publisher side, an event is published at a location. The subscribers are moving all the time and the events are arriving continuously. The moving subscribers should be continuously monitored and notified once there is a matching event in their notification regions. In order to meet these desiderata, the following two main challenges need to be tackled:

- *how to effectively process continuous moving subscriptions against dynamic event streams.* Since we are dealing with continuous moving spatial queries, we can utilize the safe region to reduce the communication cost and computation cost. If there are no matching events nearby, the users are safe to disconnect from the server and do not need to periodically update their locations. However, we observe that the safe region techniques used in [84, 35, 5, 75, 37, 32] fail to work effectively when dynamic events are considered. This is because all these techniques assume the events are static. However, in our new scenario, events are arriving continuously and newly arrived events can trigger new communication needed to update the safe region, besides the communication incurred by location updates. Moreover, these two types of communication have conflicting requirements on the size of the safe region. Therefore, there is a need to reconsider how best to exploit the safe region.

- *how to support symmetric boolean expression matching with spatial constraints between subscribers and publishers.* In other words, when a subscriber arrives, we need to search in the event database for matching events within the specified

notification radius; when a new event arrives, we need to search in the subscriber database to find matching subscribers and notify them. In order to satisfy the real time requirement, efficient indexes should be proposed to handle the challenge.

Chapter 4 present techniques to solve these two challenges.

### 1.2.3 Optimal Trajectories Queries for Influence Maximization

Unlike traditional influence maximization problem, trajectory influence maximization problem aims to select top-$k$ trajectories for moving vehicles to maximize the influence of an advertisement among a large group of audience. This problem is formulated for the first time and shown to be NP-hard. All the previous methods focus on the improvement based on the IC model or linear threshold model in social networks and can not be applied to our problem. A naive solution for the trajectory influence maximization problem is to exhaustively examine all the possible size-$k$ combinations and return the one with the maximum influence. However, this method is intractable because the number of candidate sets grows exponentially with the number of trajectories and the cost of influence calculation for each candidate set is expensive. In order to reduce the number of candidates examined and find the best-$k$ trajectory set as early as possible, an exact algorithm with efficient pruning techniques are devised in Chapter 5. In addition, when $k$ is large, the computation overhead is expected to be high. Thus, several approximation methods are also provided in Chapter 5 to support the scenario when $k$ is large.

## 1.3 Contributions

The main contribution of this thesis is the efficient management of the moving objects data in order to make the location-based services more intelligent. Based on the two characteristics (i.e., real time and traceable) of the moving objects data, we propose three different kinds of location-based queries. The first query recommends objects of interest to moving users based on the network distance instead of the Euclidean distance, which makes the results more realistic. The second query further improves users' experience by allowing users subscribing to dynamic event streams. The third query utilizes the historical movement pattern of the audience and can provide right advertisement to the right audience at the right time, which can benefit not only consumers but also companies.

### 1.3.1 Moving Spatial Keyword Queries on Road Networks

We formalize the problem of *moving top-k spatial keyword* (MkSK) queries in the domain of road networks. Given a set of spatial-textual objects and a moving query on a road network, MkSK queries continuously return $k$ best objects. To the best of our knowledge, this is the first attempt on this important problem with real-world applications.

We propose two methods that traverse the network in an incremental manner, namely, *query-centric algorithm* (QCA) and *object-centric algorithm* (OCA). In principle, both methods benefit from the reduction of the problem from finding the objects of interest for moving queries to examining static network nodes. QCA starts traversing the network from an end node of the edge on which the query location lies, until it finds the top-$k$ results. Meanwhile, it maintains an expansion tree to avoid unnecessary traversing of some network edges for subsequent processing. OCA takes a different approach and starts the traversing from objects which are relevant to the query keywords. In this way, after the initial processing, an order-$k$ shortest path tree is constructed and subsequent processing can use this tree to significantly reduce the number of edges traversed.

We report an extensive set of experiments conducted with real road network datasets to compare our algorithm performance with three baseline methods. In some settings, the cost saving can be as much as one order of magnitude. Results reveal that each of the proposed methods may perform best under different parameter settings.

### 1.3.2 Moving Spatial Queries against Dynamic Event Streams

To the best of our knowledge, Elaps is the first location-aware pub/sub system that takes into account continuous moving spatial queries as well as dynamic event streams.

To reduce the communication cost, we optimize the design and processing of safe regions in several ways. First, given a safe region, we derive its *impact region*. The impact region is a novel concept used to identify if its corresponding safe region is affected by newly arrived events. Second, we propose a cost-based approach to determine the optimal safe region size to keep the communication overhead low. Our cost model considers the communication cost incurred by location updates as well as that incurred by event arrival. Third, based on the cost model, we design two new schemes, iGM and idGM, to incrementally construct safe regions. iGM partitions the space into $N \times N$ cells and a safe region is represented by the set of cells that it covers.

Then iGM starts from the cell containing the user's current location and iteratively expands it to cover nearby cells. In each expansion, a cell is selected based on certain criteria and added into the safe region. In the meantime, the impact region is updated accordingly. The algorithm terminates when any further expansion violates our cost model. To reduce the computation cost, we propose a new index named BEQ-Tree which integrates Quadtree [29] with boolean expressions seamlessly to support spatial boolean expression matching and safe region construction efficiently.

Moreover, we conduct comprehensive experiments using real datasets to evaluate the system performance. We use geo-tweets from Twitter and venues from Foursquare to simulate publishers and boolean expressions generated from AOL search log to represent users intentions. We test user movement in both synthetic trajectories and real taxi trajectories. The results show that our proposed iGM and idGM can reduce the communication overhead by 10 times. Also, our proposed index handles spatial boolean expression matching significantly faster than the competing methods.

### 1.3.3 Optimal Trajectories Queries for Influence Maximization

We formulate the influence maximization problem in trajectory databases and show it is NP-hard. Given a trajectory database and a group of audience attached with spatial-temporal patterns, for an advertisement, our goal is to find $k$ trajectories to maximize the influence of the advertisement.

To efficiently find top-$k$ trajectories, we propose an expansion-based method that enumerates the trajectory combination in a best-first manner. The algorithm starts by calculating the influence score of each trajectory w.r.t to the query advertisement. The trajectories are then sorted by the influence and accessed in order. In each iteration, combinations with the new trajectory are enumerated. If a combination contains less than $k$ trajectories, it is considered incomplete and we estimate its upper bound influence. Otherwise, we have found a candidate with $k$ trajectories and can calculate its exact influence score. The algorithm terminates when the upper bound influence scores of all the incomplete combinations and unvisited trajectories are smaller than the best result ever found. To improve the efficiency, we propose two effective estimations of the upper bound based on a new concept named incremental influence. However, the expansion-based method is not scalable when $k$ is large. The number of candidates grows exponentially with $k$ and the computation becomes intractable and the memory cost is not affordable. To address the issue, we propose three approximate methods with performance guarantees to solve the problem. The

first is a baseline greedy algorithm which finds the trajectory with the maximum incremental influence at each iteration until $k$ trajectories are found and achieves a $(1 - 1/e)$ approximation ratio. The second is a cluster-based algorithm that further improves the efficiency of greedy algorithm and guarantees the same approximation ratio. It partitions the trajectory database into clusters and allows us to access the clusters in an order such that promising trajectories will be found earlier. Our third approximate solution, named threshold-based method, provides a flexible means to adjust the tradeoff between efficiency and accuracy using a threshold $\epsilon$. It guarantees a $\epsilon$ approximation ratio for any $\epsilon \in (0, 1]$. In addition, we propose a group greedy method to support the influence maximization for a group of advertisements, which selects the trajectories by considering all the advertisements simultaneously and can guarantee a $(1 - 1/e)$ approximation ratio.

We use three real Singapore datasets to model user motion patterns and construct trajectory databases. The results show that our proposed methods can solve the trajectory influence maximization problem efficiently.

## 1.4 Organization

The rest of the thesis is organized as follows:

- Chapter 2 reviews related topics. The surveyed topics include background knowledge about location based services and trajectory databases.
- Chapter 3 presents our study on moving spatial keyword queries on road networks.
- Chapter 4 describes our location-aware pub/sub system named Elaps to handle moving spatial keyword queries against dynamic event streams efficiently.
- Chapter 5 presents our framework that maximizes the influence of an advertisement in trajectory databases.
- Chapter 6 concludes our thesis and discusses several possible directions for future work.

## 1.5 Published Material

- The work in Chapter 3 has been published as a journal paper [32] in Geoinformatica 2015:

**Long Guo**, Jie Shao, Htoo Htet Aung and Kian-Lee Tan, "Efficient Continuous Top-$k$ Spatial Keyword Queries on Road Networks", Geoinformatica, 19(1):29-60, Jan 2015, Springer.

- The work in Chapter 4 has been published as a conference paper [33] in SIGMOD 2015 and a demo paper [31] in ICDE 2015:
  **Long Guo**, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, Zhifeng Bao, "Location-Aware Pub/Sub System: When Continuous Moving Queries Meet Dynamic Event Streams", Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data.
  **Long Guo**, Lu Chen, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, Zhifeng Bao, "Elaps: An Efficient Location-Aware Pub/Sub System", 2015 IEEE 31st International Conference on Data Engineering.

- The work in Chapter 5 is ready for submission:
  **Long Guo**, Dongxiang Zhang, Gao Cong, Wei Wu, Kian-Lee Tan, "Efficient Influence Maximization in Trajectory Databases".

# Chapter 2

# Background and Related Work

In this chapter, we review some background knowledge and existing works that are related to this thesis. As the three pieces of work in this thesis deal with spatial queries related to moving objects and their trajectories to improve location-based services, we first introduce some background knowledge about location-based services and trajectory databases, and then describe existing works related to each of the spatial queries studied in this thesis.

## 2.1 Location Based Services

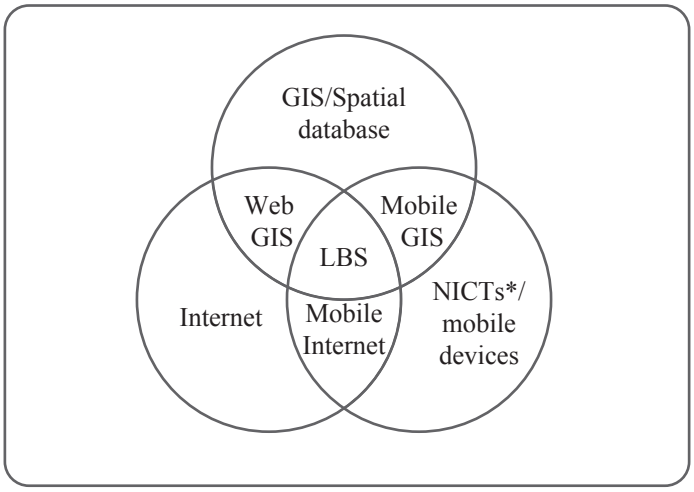



Figure 2.1: LBS as an intersection of technologies.

Location-based services (LBSs) are an important class of context-aware applications. They answer location-based queries, where a location is either explicit or implicit. As shown in Fig. 2.1, location-based services are an intersection of three

technologies, which are created from New Information and Communication Technologies (NICTS) such as the mobile telecommunication system and GPS-enabled devices, from Internet and from Geographic Information Systems (GIS) with spatial databases [67, 66].

The service and application providers provide location based services by using queries called location based queries (LBQ). The result of these queries is based on the location of the mobile device. Some of the most popular LBQs are listed as follows.

- **Range Queries (RQ)** retrieve the objects located within a specific region. If the region is rectangular, the range queries are known as window queries. For range queries, the objects of the query can be either stationary or moving, corresponding to several different query type. The problem addressed in Chapter 4 dealing with moving range queries falls in this category.

- **Nearest neighbor (NN)** queries are used to get the objects closest to a specific location. If they are capable of getting top k objects other than the nearest one, they are called kNN queries [63]. Reverse kNN queries [44] can fetch the objects that have a specified location among their $k$ nearest neighbors. Constrained NN queries [28] are the queries with a range constraint for the objects retrieved. Group NN queries [58] retrieve the objects with the smallest sum of distances to all locations in a group.

- **Spatial keyword queries (SKQ)** retrieve $k$ objects based on a ranking function that takes into account both textual relevance and spatial relevance. Actually, SKQ is a variant of kNN, which consider not only spatial locations but also textual descriptions to find objects of interest. Recently, a considerable number of studies have focused on this interesting problem [86, 27, 23, 82, 74, 61, 80, 83]. The problem addressed in Chapter 3 dealing with moving spatial keyword queries falls in this category.

- **Optimal routes queries (ORQ)** aim to find the optimal routes based on some predefined preference score [68, 70, 46]. The preference score can be defined based on the spatial-textual attributes of some POIs (e.g., restaurants or tourist attractions) or the past movement historical data of some users. The problem addressed in Chapter 5 dealing with trajectories recommendation based on the motion patterns of the users falls in this category.

Location-based queries can also be classified on the basis of the query movement. The queries corresponding to a static query are called snapshot queries, where the answer is computed only once and forwarded to the user at once, such as the optimal routes queries in Chapter 5. The queries corresponding to a moving query are called continuous queries, which are evaluated continuously until the user decides to terminate them, such as the moving spatial keyword queries in Chapter 3 and the moving range queries in Chapter 4.

## 2.2 Trajectory Databases

A spatial trajectory is a trace generated by a moving object in geographical spaces, usually represented by a series of chronologically ordered points, e.g., $p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n$, where each point consists of a geospatial coordinate set and a timestamp such as $p = (x, y, t)$. Spatial trajectories have offered us unprecedented information to understand moving objects and locations, calling for systematic research and development of new computing technologies for the processing, retrieving, and mining of trajectory data and exploring its broad applications. Therefore, trajectory databases has become an important research area and attracted a great deal of research interest during the last decade.

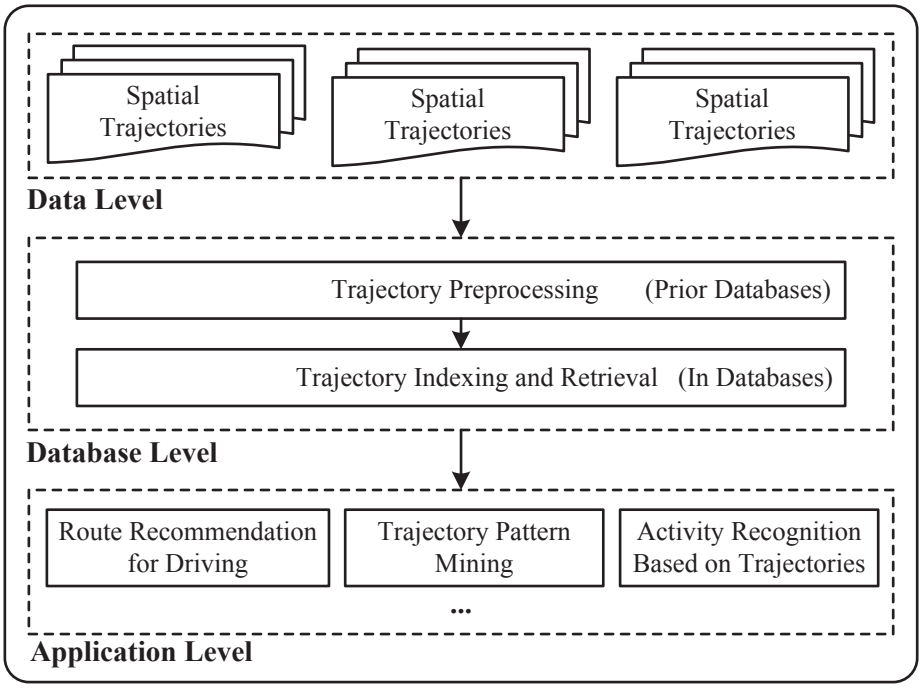



Figure 2.2: Framework of Trajectory Databases.

Fig. 2.2 presents the framework of the trajectory databases. We introduce the trajectory databases from the following three levels.

- **Data Level.** With the help of location positioning and wireless communication technologies, more and more spatial trajectories representing the mobility of a variety of moving objects, such as people, vehicles, animals, and natural phenomena, in both indoor and outdoor environments can be collected. For example, travelers log their travel routes with GPS trajectories for the purpose of memorizing a journey and sharing experiences with friends; a user carrying a bus smart card unintentionally generates many spatial trajectories represented by a sequence of bus station locations with corresponding boarding times; many taxis in major cities equipped with a GPS sensor report a time-stamped location to a data center with a certain frequency; biologists solicit the moving trajectories of animals like migratory birds for research projects.

- **Database Level.** While spatial trajectories carry rich information that is valuable in a variety of applications, we have to deal with a number of issues before using them. First, the continuous movement of an object is usually recorded in an approximate form as discrete samples of location points. On the one hand, a high sampling rate of location points leads to high overhead in data storage, communications and processing due to the large amount of data generated. In this case, it is vital to design data reduction techniques that compress the size of a trajectory while maintaining the utility of the trajectory. On the other hand, a low sampling rate of locations results in some uncertainty between the two updated locations of a moving object. In this case, some techniques need to be proposed to remedy the uncertainty between two locations to improve the accuracy of a trajectory. Second, a trajectory is usually generated with occasional outliers or some noisy points caused by the poor signal of location positioning systems. As a result, techniques for filtering the noisy points are needed for preprocessing spatial trajectories. After the preprocessing work, the spatial trajectories needs to be stored in trajectory databases in the form of trajectory indexes to facilitate and accelerate the retrieval of the trajectories based on some search conditions.

- **Application Level.** After preprocessing and organizing spatial trajectories with corresponding techniques, we can start using them in a variety of applications, such as route recommendation for driving, trajectory pattern mining and activity recognition based on trajectories.

## 2.3 Moving Spatial Keyword Queries on Road Networks

In this section, we discuss some related studies of spatial keyword query, and continuous monitoring of moving queries.

### 2.3.1 Spatial Keyword Queries

Spatial keyword queries have drawn lots of attention in recent years. Zhou et al. [86] evaluate three different hybrid indexing structures of integrating inverted files and $R^*$-trees. Their experiment shows that building an inverted index on top of R*-tree is the best scheme. The IR-tree [23] also incorporates the inverted files and R*-trees to answer a top-$k$ spatial keyword query. Unlike the method in [86] which first uses one index to filter web documents and then uses the other index to process the query, the IR-tree combines these two indexes to jointly prune the search space. In [74], the IR-tree is further extended with multiple query optimization for a group of top-$k$ spatial keyword queries to simultaneously prune the search space. Felipe et al. [27] propose an index called IR$^2$-tree which integrates an R-tree and signature files to answer a top-$k$ spatial keyword query. Rocha-Junior et al. [61] propose the S2I index which uses textual-first partition and splits the database into inverted lists. If a keyword is frequent, an aggregated R-tree is built. Otherwise, the infrequent keywords are stored in a flat file. Zhang et al. [83] propose the $I^3$ index which also uses textual-first partition. Unlike the S2I in [61], $I^3$ maps a keyword to a list of keyword cells which are generated using Quadtree. Zhang et al. [82] propose an $m$-closest keywords (mCK) query that retrieves the spatially closest objects which match $m$ user-specified keywords. An index called bR*-tree is devised to augment each node with a bitmap for query processing. These studies all assume Euclidean space and are not applicable to road networks.

The only work that supports spatial keyword query on road networks is reported in [62], where the authors describe the indexing structure and utilize overlay network for efficient query processing. However, the problem of processing continuous queries has not been addressed in the literature, which arises naturally in a travel environment.

### 2.3.2 Continuous Monitoring

We consider the setting of moving query and stationary objects. In the following, we review related work under this setting. Kolahdouzan and Shahabi [43] propose

an upper bound algorithm for continuous $k$ nearest neighbor queries in spatial networks. Their algorithm retrieves ($k$+1) objects given a query location and calculates an upper bound, which is used to eliminate the computation of $k$NN queries between locations that the result does not change. Cho and Chung [22] propose a continuous $k$NN technique that performs snapshot $k$NN queries at intersections along the query trajectory. Conceptually, the basic idea of our two methods is similar to this. However, in addition to performing snapshot queries at the intersections, we maintain an expansion tree to further reduce the number of edges traversed. Moreover, none of the studies on continuous spatial queries on road network consider textual relevancy of objects.

Recently, Wu et al. [75] and Huang et al. [37] both study continuously moving top-$k$ spatial keyword queries. They propose different algorithms for computing safe zones that guarantee correct results. However, their algorithms are restricted to Euclidean space, and thus, cannot be applied to our problem where query and objects of interest are constrained on a road network.

## 2.4 Moving Spatial Keyword Queries Against Dynamic Event Streams

In Elaps, our primary goal is to build a pub/sub system that caters for both continuous moving subscribers and dynamic event streams from publishers, and our secondary goal is to support a more expressive event matching semantic than pure keywords. Thus, in this section, we first highlight the novelty of Elaps as compared with existing location-aware pub/sub systems and then present the related work about boolean expression matching.

### 2.4.1 Location-aware Pub/sub

To clearly distinguish Elaps from other works in the literature, we first present a taxonomy of the existing location-aware pub/sub systems and our proposed Elaps in three aspects: support for continuous moving queries, support for dynamic event streams and event matching semantic (see Table 2.1).

**Location-aware pub/sub system.** A location-aware pub/sub system sends matching geo-tagged events to the corresponding subscribers. Compared to existing location-aware pub/sub systems [50, 13, 24], Elaps has two distinguishing features, as shown in Table 2.1. First, it continuously monitors users' locations and sends nearby notifications in real time, while [50, 13, 24] assume users' locations are static. Second,

| | Location-aware news feed | | Location-aware pub/sub | | | Continuous spatial queries | | |
|---|---|---|---|---|---|---|---|---|
| | GeoFeed[6] | MobiFeed[78] | R$^t$-tree[50] | IQ-tree[13] | CLCB[24] | KNN/Range | Spatial keyword | Elaps |
| CMQ[a] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| DES[b] | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ |
| MS[c] | - | - | keyword | keyword | BE | - | keyword | BE |

Table 2.1: Comparison of existing location-aware pub/sub system.

[a]continuous moving queries
[b]dynamic event streams
[c]matching semantic

it allows users to specify their interests with boolean expressions, which provides better flexibility and expressiveness in shaping an interest than keyword subscription in [50, 13].

**Continuous spatial queries.** Another problem that is closely related to our research is the processing of continuous spatial queries such as continuous KNN/range queries [84, 35, 5, 48] and continuous spatial keyword queries [75, 37, 32]. Given a moving query, existing works have developed techniques to continuously return a set of objects satisfying the spatial constraint [84, 35, 5] or the combined spatial and textual constraints [75, 37, 32]. To reduce the communication overhead, these methods typically require users to update their locations only when they move out of their safe regions. Moreover, users within the safe regions can safely disconnect from the server as long as there is no matching event in their neighborhood. Since we focus on range query, the safe region construction methods for spatial keyword query in [75, 37, 32] cannot be applied. Besides, those proposed for continuous knn/range queries [84, 35, 5, 48] assume the publisher events are static, so they fail to solve our problem either (Please refer to Section 4.2 for a detailed justification). Lastly, these methods allow users to search for relevant events by keyword subscriptions while Elaps uses a more expressive boolean expression to model user intent.

**Location-aware news feed system.** We also note that location-aware news feed systems enable mobile users to share geo-tagged user-generated messages. In GeoFeed [6], users retrieve geo-related message updates from either their social friends or social media. It differs from Elaps in that: (1) GeoFeed is pull-based, which poses a high chance of missing interesting events, (2) users are static objects, (3) users cannot customize the messages they are interested in explicitly. MobiFeed [78] extends GeoFeed to support mobile users. Instead of monitoring the location of moving users from time to time like Elaps, it predicts the potential next locations for a moving user in advance and pushes the messages around these locations to the user. Thus,

it cannot support continuous moving queries; moreover, there is no guarantee that users will not miss any matching event.

Therefore, to our knowledge, Elaps is the first location-aware pub/sub system that takes into account continuous moving queries as well as dynamic event streams.

### 2.4.2 Boolean expression matching

Unlike existing pub/sub systems, Elaps uses boolean expressions, which are more expressive than keywords, to model user intent. The advantage of using boolean expressions is that users can subscribe to structured, semi-structured and unstructured data. For instance, the pub/sub systems based on keyword subscription cannot support numeric attribute matching such as price $<$ \$1000, while the numeric attribute matching can be easily supported by boolean expressions. In the following, we introduce some related work about boolean expression matching.

There have been several studies on efficient *event matching* over a large quantity of subscriptions [26, 71, 64]. Whang et al. [71] propose $k$-index which partitions the subscriptions into inverted lists, whose key is a triple of subscription size, attribute name and attribute value. To further improve efficiency and expressiveness, Sadoghi and Jacobsen propose the BE-Tree [64] and develop a two-stage partition mechanism to facilitate pruning. Zhang et al. [81] propose a scalable and extensible index named OpIndex to support high-dimensional and sparse database effectively. OpIndex adopts a two-layer partition scheme and can be extended to support more expressive subscriptions.

In Elaps, we require not only *event matching* but also *subscription matching* over a large quantity of events. Among the above methods, only $k$-index and OpIndex can be extended to support *subscription matching*. Both indexes adopt a two-layer partitioning scheme and use the inverted list to group the attributes in the second layer. Their difference is the way they partition the events in the first layer. $k$-index partitions the events based on the event size, while OpIndex partitions the events based on the pivot attribute selected for each event. However, both partitioning schemes are not efficient in supporting subscription matching, especially when the spatial matching is taken into consideration. In Chapter 4, we propose a more efficient index BEQ-Tree to support spatial subscription matching.

## 2.5 Optimal Trajectories Queries for Influence Maximization

In this section, we introduce several topics related to the influence maximization on moving objects.

### 2.5.1 Influence Maximization

**Influence Maximization in Social Networks.** The influence maximization problem is proposed in [25, 60] and has attracted great attention ever since. Initially, the proposed methods are probabilistic and have no bounded influence spread guarantee. To fix the issue, Kempe et al. [40] propose two discrete influence spread models: Independent Cascade (IC) model and Linear Threshold model. They prove the influence maximization problem is NP-hard based on the spread models and propose a greedy framework with $(1-1/e)$ approximation ratio guarantee. There are many subsequent studies aiming at improving the efficiency of the greedy framework. When Independent Cascade (IC) model is considered, Kimura et al. [41] use the shortest path to approximate the actual spread process. Leskovec et al. [47] propose a "lazy-forward" algorithm. Chen et al. [18] propose a degree-discount heuristics for an IC model where all propagation probabilities are the same. Chen et al. [17] propose the PMIA algorithm to solve the influence spread maximization problem. Similar ideas [19] have also been applied to support Linear Threshold model. The latest work comes from Tang et al. [69] who propose an algorithm with near-optimal time complexity and novel heuristics for improving empirical efficiency.

**Topic-aware Influence Maximization.** Barbieri et al. [7] propose the Topic-Aware Influence Cascade (TIC) model. In the TIC model, the relationship strength between two vertices is computed by their topic preference learned from history activities on a social network. Based on the TIC model, Barbieri et al. [4] propose a similarity-based method, INFLEX, to support topic-aware influence maximization. Chen et al. [16] develop a preprocessing based strategy, MIS, for topic-aware influence maximization. However, both INFLEX and MIS have no influence spread guarantee. Chen et al. [15] propose a best effort method which has an influence spread guarantee while keeping high performance.

**Location-aware Influence Maximization.** Li et al. [49] extend the influence maximization problem by considering the spatial context. In particular, the problem finds top-$k$ users in a location-aware social network that have the highest influence

upon a group of audience in a specified region. They use IC model for influence propagation and proposed several efficient algorithms.

In Chapter 5, we consider a novel influence maximization problem in trajectory databases. We are the first to formulate the problem and exhibit its usefulness in location-aware advertising. All the above three categories are focused on the improvement based on the IC model or linear threshold model in social networks and cannot be applied to our problem. Besides that, we are the first to propose an accurate method to solve the influence maximization problem while previous methods only provide approximate methods.

## 2.6 Facility location problem

The facility location problem, also known as the MaxBRNN problem, is first introduced by Cabello et al. [11]. It finds a location for a new facility which can attract the maximal number of audiences. The literature falls into two categories. In the first category, the consumers are associated with static locations. Wong et al. [73] propose the first polynomial-time complexity algorithm, *MaxOverlap*. Some variations, such as the extension of the *MaxOverlap* algorithm in a three-dimensional space and other $L_p$-norm metric spaces, are studied in [72]. Liu et al. [52] develop an improved algorithm *MaxSegment*. Zhou et al. [87] study a generalized MaxBR$k$NN problem in which a client may have different probabilities to visit different servers. Recently, Chen et al. [21] propose an efficient algorithm, MinMax-Alg, for the optimal location query. In the second category [1, 8, 51], the consumers are modeled as moving objects. The goal is to identify a location or segment that intercepts the most flow from moving customers. Flows are made up by pre-planned customer trips and the idea is that customers can choose to interrupt their trip to receive a service from a facility at a nearby location.

The facility location problem aims to minimize the overall distance from the given users to the new facility while our problem aims to maximize the influence of the advertisement in the given users. In addition, in our problem, the advertisement is considered to be moving with the trajectory, which is more challenging than a fixed location in the facility location problem.

# Chapter 3

# Efficient Moving Spatial Keyword Queries on Road Networks

## 3.1 Introduction

In this chapter, we study how to efficiently process continuous top-$k$ spatial keyword queries when query and objects of interest are on a road network. Consider a scenario that Alice and Bob are visiting a foreign city. When Bob intends to find a buffet seafood restaurant for lunch, a spatial keyword query can be submitted to obtain information about some buffet seafood restaurants opening for lunch nearest to them. However, Alice is not satisfied with these results. With continuous spatial keyword queries, they can just keep traveling and up-to-date results will be reported to let them choose from, until an attractive one appears.

Although continuous queries with network distance can provide up-to-date and accurate results as the query point moves, it is often costly to monitor such moving queries. A straightforward solution is to traverse the road network to find the top-$k$ results every time the query is evaluated. However, such a scheme will lead to lots of unnecessary repetitive traversing of network edges. Many proposals for moving queries in Euclidean space such as [55, 12, **?**, 37] unitize the concept of safe region. In response to a query, its result can be computed together with a safe region within which the result remains valid. Only when the query exits its safe region will a new query (additional processing) be invoked, which repeats the above process. However, query processing on road networks is fundamentally different. Compared with previously proposed continuous *spatial* queries on road networks such as [22, 20, 54], how to exploit the opportunities of simultaneously using spatial proximity and textual relevancy for joint pruning brings new challenges to query optimization of our problem.

We propose two methods that traverse the network in an incremental manner, namely, *query-centric algorithm* (QCA) and *object-centric algorithm* (OCA). In principle, both methods benefit from the reduction of the problem from finding the objects of interest for moving queries to examining static network nodes. QCA starts traversing the network from an end node of the edge on which the query location lies and incrementally expands the network from a query node which is similar in spirit to Dijkstra's algorithm, but further uses a pair of upper and lower bounds to exploit both spatial and textual domains for joint pruning. The expansion terminates until it finds the top-$k$ results. Meanwhile, it maintains an expansion tree to avoid unnecessary traversing of some network edges for subsequent processing.

OCA takes a different approach and starts the traversing from objects which are relevant to the query keywords. At the beginning, OCA loads relevant objects that match at least one of the query keywords in their descriptions. Instead of traversing from the end nodes of the edge on which the query location lies, OCA starts traversing the network from a relevant object and constructs a shortest path tree. We can get the results of the two end nodes of the edge on which the query location lies using the shortest path tree. Moreover, the incremental shortest path tree construction is characterized by allowing the construction process to halt when the top-$k$ results of the requested node are found and to resume when more results are required. OCA does not suffer from repetitive result evaluation, since query results of each node are obtained via object-centric expansion, where the shortest path tree remain valid. In addition, when constructing the shortest path tree, OCA also obtains results or partial results of surrounding nodes, which makes it suitable for the processing of moving spatial keyword queries on road networks.

Our contributions are summarized as follows.

- We formalize the problem of continuous top-$k$ spatial keyword queries in the domain of road networks. To the best of our knowledge, this is the first attempt on this important problem with real-world applications.

- We propose two algorithms that incrementally expand the network from the query location and objects relevant to the query keywords, respectively. Both of them can reduce lots of unnecessary repetitive traversing of network edges for continuous monitoring.

- We report an extensive set of experiments conducted with real road network datasets to compare our algorithm performance with three baseline methods. In

some settings, the cost saving can be as much as one order of magnitude. Results reveal that each of the proposed methods may perform best under different parameter settings.

The rest of this chapter is organized as follows. First, we present formal definition of the problem in Section 3.2. Then, we describe the query-centric algorithm in Section 3.3, followed by the object-centric algorithm in Section 3.4. We report the experimental evaluation in Section 3.5. Finally, we conclude this chapter in Section 3.6.

## 3.2 Problem Statement

| Symbol | Description |
|---|---|
| $G(N, E)$ | the graph model of road network |
| $N$ | the set of nodes in $G$ |
| $E$ | the set of edges in $G$ |
| $(n_i,n_j)$ | the edge that connects $n_i$ to $n_j$ |
| $n_i$ | an intersection node or terminal node |
| $\mathcal{T}$ | the expansion tree |
| $O$ | the set of spatial-textual objects on $G$ |
| $O_r$ | the set of objects relevant with the query keywords |
| $O_{re}$ | the set of relevant objects in the expansion tree |
| $O_{(n_i,n_j)}$ | the set of objects lying on the edge$(n_i,n_j)$ |
| $o$ | a spatial-textual object |
| $q$ | a top-$k$ spatial-keyword query |
| $q.l$ | the query location |
| $q.d$ | the set of query keywords |
| $q.k$ | the number of requested objects of interest |
| $q.r$ | the moving direction of the query |
| $\tau$ | the score which is used to rank the objects |
| $\delta$ | spatial proximity |
| $\theta$ | textual relevance |
| $R_q$ | the top-$k$ results of query $q$ |
| $R_{n_i}$ | the top-$k$ results of node $n_i$ |
| $O_{d1}$, $O_{d2}$ | the set of objects in $R_q$ that can be replaced |
| $O_{r1}$, $O_{r2}$ | the set of objects that can replace the objects in $R_q$ |
| $d(p_1, p_2)$ | the shortest path distance between two points $p_1$ and $p_2$ |
| $\|p_1, p_2\|$ | the length of the segment between $p_1$ and $p_2$ |

Table 3.1: Frequently used notations.

In this section, we introduce the continuous moving spatial keyword queries on road networks formally. Table 3.1 summarizes the notations frequently used in the

chapter.

**Road network.** A road network is generally represented by a connected and undirected planar graph $G(N, E)$[1], where $N$ is the set of nodes and $E$ is the set of edges. In this work, we define the edge as follows.

**Definition 3.1 (Edge)** *An edge is defined as one or multiple road segments (i) two endpoints are intersection nodes (with degree above 2) or terminal nodes (with degree 1) and (ii) intermediate nodes all have degree 2.*

In the following context when we use *edge*, it indicates that this edge has two endpoints, which are either intersections or terminal nodes.

**Object set.** Let $O$ represent a set of spatial-textual objects on the edges $E$ of $G$, where each object $o \in O$ has a spatial location $o.l$ and a textual description (or called *document*) $o.d$. Denote $|o, n|$ and $|o, n'|$ as the distances between an object $o$ and two end nodes of the edge $(n, n')$ on which it lies. The shortest path distance between two objects $o$ and $o'$ on $G$ is defined as $d(o, o')$.

**Top-$k$ spatial keyword query on a road network.** Define $q =< q.l, q.d, q.k, q.r >$ to be a top-$k$ spatial keyword query on a road network $G$, where $q.l$ is the query location, $q.d$ is the set of query keywords, $q.k$ is the number of requested results and $q.r$ is the moving direction of $q$. Given a set $O$ of spatial-textual objects on $G$, a top-$k$ spatial keyword query $q$ returns $q.k$ objects from $O$ in ascending order of score $\tau$, which is defined as

$$\tau(q, o) = \alpha \cdot \delta(q.l, o.l) + (1 - \alpha) \cdot [1 - \theta(q.d, o.d)], \tag{3.1}$$

where $\delta(q.l, o.l)$ reflects spatial proximity of $o.l$ from the query location $q.l$, and $\theta(q.d, o.d)$ reflects textual relevance of $o.d$ with respect to the query keywords $q.d$. Like other typical work of spatial keyword queries such as [23], a preference parameter $\alpha \in (0, 1)$ is used to define relative importance of one measure over the other. For example, $\alpha < 0.5$ increases the weight of textual relevance over spatial proximity.

*Spatial proximity ($\delta$).* Spatial proximity measure can be defined as

$$\delta(q.l, o.l) = \frac{d(q.l, o.l)}{d_{max}}, \tag{3.2}$$

where $d_{max}$ is the largest network distance between any object and any location in $G$. $d_{max}$ can be obtained by traversing the network from each object until the entire network is expanded and keeping the maximum distance. $\delta$ is in the range of [0,1].

[1]Note that the methods proposed in this chapter can also be applied to the directed road network. The only difference is that now the connectivity between edges depends on not only the adjacency between edges but also the direction of edges.

*Textual relevance ($\theta$).* Textual relevance measure can be captured by any information retrieval model. In this work, cosine similarity [88, 62] is used to evaluate the similarity between $q.d$ and $o.d$, which is defined as

$$\theta(q.d, o.d) = \frac{\sum_{t \in q.d} \omega_{t,q.d} \cdot \omega_{t,o.d}}{\sqrt{\sum_{t \in q.d} (\omega_{t,q.d})^2 \cdot \sum_{t \in o.d} (\omega_{t,o.d})^2}}, \tag{3.3}$$

the weight $\omega_{t,q.d} = \ln(1 + \frac{|O|}{df_t})$, where $|O|$ is the number of objects in $O$ and $df_t$ is the number of objects with $t$ in their descriptions (document frequency); and the weight $\omega_{t,o.d} = 1 + \ln(f_{t,o.d})$, where $f_{t,o.d}$ is the number of occurrences (frequency) of term $t$ in $o.d$. $\theta$ is in the range of [0,1] (property of cosine).

For simplicity of computation, Equation 3.3 can be rewritten in the *impact* form as

$$\theta(q.d, o.d) = \sum_{t \in q.d} \lambda_{t,q.d} \cdot \lambda_{t,o.d}, \tag{3.4}$$

where the impact $\lambda_{t,d} = \frac{\omega_{t,d}}{\sqrt{\sum_{t \in d} (\omega_{t,d})^2}}$ is normalized weight of the term in the document, by taking document length into account [65, 2].

A lower score $\tau$ means the object is better. In this chapter, we study the efficient processing of *moving top-k spatial keyword* (MkSK) queries on road networks which is defined as follows.

**Definition 3.2 (MkSK queries on road networks)** *Given a set $O$ of spatial-textual objects and a moving query $q$ on a road network $G$, $MkSK$ queries continuously return $k$ ranked objects in ascending order of score $\tau$.*

## 3.3 Query-centric Algorithm

In this section, we present our first method named *query-centric algorithm* (QCA). We start with the basic idea of our query processing in Section 3.3.1, followed by the snapshot query algorithm in Section 3.3.2. We present the usage of expansion tree in Section 3.3.3 and show how the top-$k$ results can be derived and safe segment can be computed in Section 3.3.4. Finally, we give complete QCA monitoring algorithm in Section 3.3.5.

### 3.3.1 Basic Idea

To solve the MkSK queries, we examine result updates for intersections along the trajectory on which the query point moves. Specifically, our method is based on the following lemma:

**Lemma 3.1** *The top-k results $R_q$ of any spatial keyword query q whose query location q.l lies on an edge$(n_i, n_j)$ are in the union of (i) the set of relevant objects $O_{(n_i,n_j)}$ on the edge, and (ii) the top-k results $R_{n_i}$ and $R_{n_j}$ of the end nodes of the edge, which can be formulated as:*

$$R_q \subseteq (O_{(n_i,n_j)} \cup R_{n_i} \cup R_{n_j}),$$

*where relevant object refers to object that matches at least one of the query keywords in its description.*

**Proof 3.1** *We prove this lemma by contradiction. First assume that there exists an object o satisfying $o \in R_q$ and $o \notin (O_{(n_i,n_j)} \cup R_{n_i} \cup R_{n_j})$. Since $o \notin O_{(n_i,n_j)}$, o is not on the edge. The shortest path from the query location q.l to o then must go through either $n_i$ or $n_j$. Without loss of generality, assume o is closer to $n_i$. Let $o'$ be any one of the objects in $R_{n_i}$. Since $o \notin R_{n_i}$, we know that $\tau(n_i, o) > \tau(n_i, o')$. Because $|q.l, n_i|$ can be added to the spatial proximity part of both sides of the above inequality and the textual relevance part remains constant, we have $\tau(q, o) > \tau(q, o')$. This means that all of the top-k results in $R_{n_i}$ have a lower score than o with respect to q.l. Thus, o should not be in the top-k results $R_q$. This is contradictory to the initial assumption that $o \in R_q$.*

We adopt a client-server architecture along with safe segment. When a top-$k$ spatial keyword query $q$ is submitted by a client, the edge on which it lies can be located and the result sets of the two end nodes of this edge are computed first. Then, the top-$k$ results of $q$ can be derived from the result sets of the end nodes, which are sent back to the client. Only when the client exits the safe segment will it send a location update to the server, which repeats the above process.

### 3.3.2 Snapshot Query Algorithm

Our work focuses on continuous monitoring, and in QCA, we employ a method similar to the algorithm named *enhanced* presented in [62] as our underlying snapshot query algorithm for a single spatial keyword query on a road network. It incrementally expands the network from a query node which is similar in spirit to Dijkstra's algorithm, but it further uses a pair of upper and lower bounds to exploit both spatial and textual domains for joint pruning. Besides the basic steps used in [62], our QCA maintains an expansion tree in $\mathcal{T}$ while traversing the network edges (these steps are underlined in Algorithm 3.1). The expansion tree is a critical component that is necessary to facilitate efficient processing of moving queries. We defer the discussion

**Algorithm 3.1:** SnapshotQueryResult(Node $s$, Edge $e$)

**environment**: Objects $O$, Query $q$, MinHeap $\mathcal{N}$, Tree $\mathcal{T}$
**input** : Node $s$, Edge $e$
**output** : Results $R_q$

update $\epsilon$ using $R_q$; $//k^{th}$ score in $R_q$; while $|R_q| < q.k, \epsilon \leftarrow 1$
$(s, n') \leftarrow e$;
set $n'$ as child of $s$ in $\mathcal{T}$;
insert $s$ and $n'$ into $\mathcal{N}$, mark $s, n'$ as visited;
$C \leftarrow FindCandidates(e.ID, s, \epsilon)$;
update $R_q$ and $\epsilon$ with $o \in C$;
$n \leftarrow \mathcal{N}.pop()$; //node $n$ in $\mathcal{N}$ with minimum $d(s, n)$
**while** $n \neq \phi$ **and** $\alpha \cdot \delta(s, n) < \epsilon$ **do**
    **foreach** *non-visited adjacent node* $n'$ *of* $n$ **do**
        set $n'$ as the child of $n$ in $\mathcal{T}$;
        insert $n'$ into $\mathcal{N}$, mark $n'$ as visited;
        $C \leftarrow FindCandidates((n, n').ID, n, \epsilon)$;
        update $R_q$ and $\epsilon$ with $o \in C$;
    $n \leftarrow \mathcal{N}.pop()$;
**return** $R_q$;

of the expansion tree and the related algorithms to the next subsection. First, we briefly introduce the indexing structure the snapshot algorithm used (refer to [62] for more details).

- **Spatial component.** This component is used to locate the edge on which the query lies.
- **Adjacency component.** This component points to adjacent nodes of a given node permitting traversing the network from node to node. We use a B-tree to point to the block in the adjacency file where the adjacent nodes of a given node are. The adjacency file stores for each node: (i) the id of each edge, and (ii) the length of the edge.
- **Mapping component.** We also use a B-tree that maps a key composed of the pair of *edge id* and *term id* to the inverted list that contains the objects lying on the edge with the term in their descriptions. This component contains also the *maximum impact* of a given term $t$ among the descriptions of the objects lying on a given edge. The maximum impact is an upper-bound impact for any object on the edge that contains $t$. Therefore, the inverted list of a term $t$ on an edge is accessed only if the lower-bound score derived by minimum distance

**Algorithm 3.2:** FindCandidates(EdgeID *eid*, NodeID *nid*, Threshold $\epsilon$)

**input**: EdgeID *eid*, NodeID *nid*, Threshold $\epsilon$
**output**: Candidates $C$

```
2  compute θ_max and τ_min;
4  if θ_max > 0 then
6      O_(n,n') ← objects on the edge;
8      foreach o ∈ O_(n,n') do
10         o.parent = nid;
12         insert o into O_re; //preserve relevant objects in expansion tree
15     if τ_min < ε then
17         foreach o ∈ O_(n,n') do
19             compute o.score;
21             if o.score < ε then
23                 insert o into C;
25 return C;
```

and maximum impact may turn an object, present on the edge, inside the top-$k$ objects found so far.

- **Inverted file component.** This component contains inverted lists and a vocabulary. Each inverted list stores the objects lying on an edge with a term in their descriptions. For each object, the inverted list stores: (i) the distance between the object and the reference node of the edge, and (ii) the impact of the term in the description of the object. The vocabulary file stores the document frequency $df_t$ of each term. This information is used to compute textual relevance of the object for a given query.

Algorithm 3.1 provides detailed steps of how snapshot query results can be obtained. The algorithm receives a query node $s$ whose result set is to be computed and the edge $e$ on which $s$ lies as input, and returns a result set $R_q$ of $k$ ranked objects in ascending order of score $\tau$. A priority queue (implemented as min-heap) $\mathcal{N}$, which is initially empty, is used to organize the encountered nodes in increasing order of distance from $s$. First, the algorithm updates $\epsilon$, which is the score of the current $k^{th}$ object in $R_q$ (line 1). Then, it locates the other end node $n'$ of $e$ and sets $s$ as the root of $\mathcal{T}$ with child $n'$ (lines 2-3). Next, $s$ and $n'$ are inserted into $\mathcal{N}$ and marked as visited (line 4). After that, it uses a *FindCandidates* procedure to retrieve candidate objects $C$ lying on the edge$(s, n')$ with score lower than $\epsilon$, and updates $R_q$ and $\epsilon$ using the objects in $C$ (lines 5-6). Subsequently, the algorithm dequeues the nearest node $n$

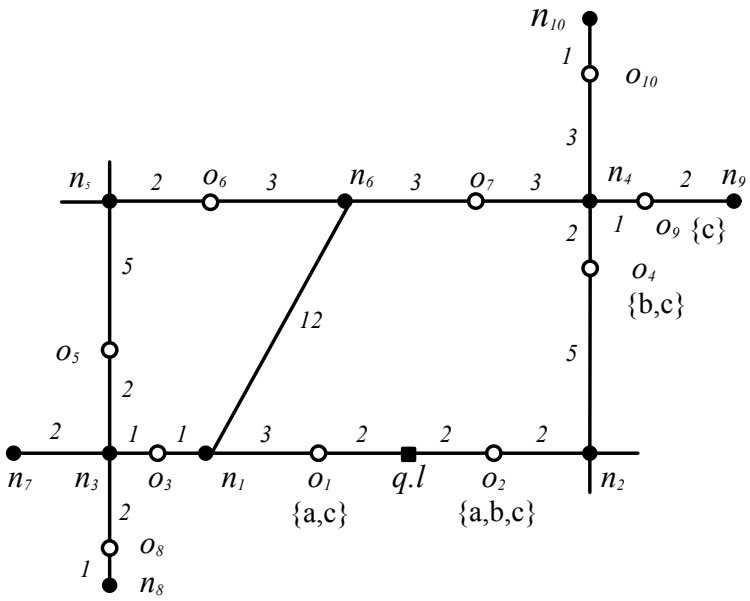



Figure 3.1: Graph representing a road network and objects.

from $s$ in $\mathcal{N}$ (line 7), and processes non-visited adjacent nodes of $n$ (lines 9-13). The algorithm terminates when the entire network is expanded, or the minimum network distance to any remaining object produces a *lower-bound score* higher or equal to the score of the $k^{th}$ object already found (line 8). The lower-bound score of a node $n$ is obtained through the network distance between $s$ and $n$ and the maximum textual relevance ($\theta = 1$). Therefore, if the lower-bound score of a node is higher than or equal to $\epsilon$, it means that even if there is a non-visited object $o$ matching all the query keywords with maximum textual relevance, its ranking cannot be better than the $k^{th}$ object in $R_q$, as $\delta(s, o.l) \geq \delta(s, n)$. This is guaranteed by the fact that the algorithm *strictly* expands the node with minimum distance from $s$.

Algorithm 3.2 provides detailed steps of the $FindCandidates$ procedure. It first computes a maximum textual relevance $\theta_{max}$ using the maximum impact $\lambda_{max}$ of each term $t \in q.d$ stored in the mapping component, and then a lower-bound score $\tau_{min}$ using the minimum network distance between the edge and the query node $s$ and the maximum textual relevance $\theta_{max}$ (line 1). Only if the lower-bound score $\tau_{min}$ is smaller than $\epsilon$ will it compute exact scores of the objects on the edge and returns a candidate set $C$ of objects with scores lower than $\epsilon$ (lines 7-12).

### 3.3.3 Using Expansion Tree

When we need to obtain the top-$k$ results of the end node of an edge, we can use Algorithm 3.1 described above. However, after getting the result set of one end node

$n$, if we retraverse the network from the other end point $n'$ for its result set, potentially there are many redundant operations here. This is the reason why we maintain the expansion tree. The sub-tree of $n'$ of the expansion tree rooted at $n$ is actually still valid when computing the result set of $n'$ and thus, we can reuse this information to avoid repetitive traversing of some network edges.

Fig. 3.1 shows an example of top-2 spatial keyword query on a road network at $q.l$ with $q.d = \{\text{c}\}$. For ease of description, we only mark the objects that contain the term 'c', and assume that the textual relevance $\theta$ is the number of occurrences of the query keywords in the description of an object $o.d$ divided by the number of keywords in the document. For example, textual relevance $\theta$ of $o_1$ whose $o_1.d = \{a, c\}$ is 0.5. In addition, we assume the maximum distance used to normalize spatial proximity $\delta$ is 30 units and the preference parameter $\alpha$ is 0.5.

First, we can get the result set of $n_1$ by using Algorithm 3.1, which is $\{o_9, o_1\}$. The score of $o_9$ is $0.5 \times \frac{17}{30} + 0.5 \times (1-1) = 0.28$ and the score of $o_1$ is $0.5 \times \frac{3}{30} + 0.5 \times (1 - \frac{1}{2}) = 0.3$. The expansion tree of $n_1$ is shown in Fig. 3.2, where the valid sub-tree of $n_2$ is shown in the ellipse. The part which is not included in the ellipse is invalid because there may be an optimal path from $n_2$ to the nodes in the invalid part. For example, path $\{n_2 \rightarrow n_4 \rightarrow n_6\}$ is shorter than path $\{n_2 \rightarrow n_1 \rightarrow n_6\}$. We use a tree structure $\mathcal{T}$ to maintain the expansion tree. For each node, if it is a non-leaf node, we preserve a child list for it (line 3 & line 10).

In order to get the result set of $n_2$, we use the following steps.

- First, we update the expansion tree $\mathcal{T}$ by removing the invalid part. This can be achieved by removing the nodes which are not descendants of $n_2$. We first build a new tree root using $n_2$. Then we remove the ancestors of $n_2$ and their children.

- Second, we insert the objects lying on the valid sub-tree whose textual relevance $\theta$ is larger than zero into the current result set $R_{n_2}$ of $n_2$ (with updated scores). A key observation here is that we need to consider all the relevant objects in $O_{re}$ belonging to the valid sub-tree rather than only considering the results of $n_1$ that fall in the valid sub-tree. The reason is that the distances from the query node (which is $n_2$ now) to the objects in $O_{re}$ become smaller, which makes it possible for the objects to replace the top-$k$ results of $n_1$ which belong to the invalid part. For example, we need to insert $o_4$ and $o_9$ into $R_{n_2}$ rather than just $o_9$. We need to consider $o_4$ because it can replace $o_1$. The top-2 results of $n_2$ are $\{o_9, o_4\}$ with scores of 0.13 $(0.5 \times \frac{8}{30} + 0.5 \times 0)$ and 0.33 $(0.5 \times \frac{5}{30} + 0.5 \times \frac{1}{2})$ respectively. As

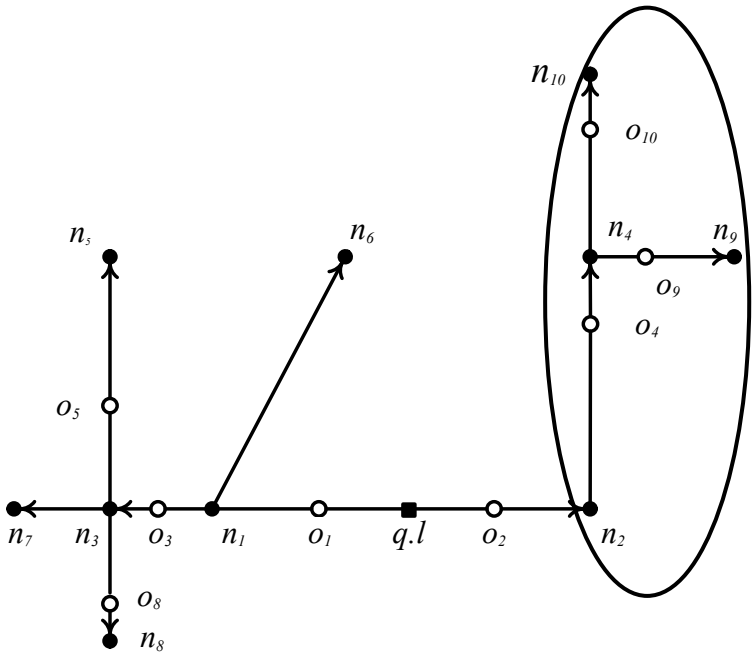



Figure 3.2: Valid part when computing the result set of $n_2$ (the sub-tree of $n_2$) of the expansion tree rooted at $n_1$.

can be seen, $o_4$ has a score of 0.33 lower than the score 0.35 ($0.5 \times \frac{6}{30} + 0.5 \times \frac{1}{2}$) of $o_1$ and replaces $o_1$, which demonstrates the observation. Although $o_4$ is not a result of $n_1$, it has a probability to be a result of $n_2$. Therefore, we need to preserve the qualifying objects ($\theta > 0$) in $O_{(n,n')}$ when the *FindCandidates* procedure is called, which can be found in Algorithm 3.2 (lines 4-6). *o.parent* is used to tell whether object $o$ belongs to the valid sub-tree.

- Finally, we compute the remaining top-$k$ results of $n_2$ with Algorithm 3.1 by initializing $\mathcal{N}$ to contain the leaves of the valid sub-tree and $n_2$ (with updated distances from $s$). Let us consider the leaves of the valid sub-tree. They consist of two node types: (i) the node that does not have adjacent nodes or whose adjacent nodes have all been visited, and (ii) the node $n$ with $\alpha \cdot \delta(n, s) \geq \epsilon$ that stops Algorithm 3.1. After adding these two types of nodes to $\mathcal{N}$, the algorithm can start from the leaves to further traverse the network, in order to get remaining results.

### 3.3.4 Deriving Top-$k$ Results and Safe Segment

We have shown that the top-$k$ results of any query location are contained in the union of the relevant objects on the edge on which the query location lies and the result

sets of two end nodes of this edge (Lemma 3.1). In this subsection, we present how to analytically derive the top-$k$ results of any location on an edge from this union.

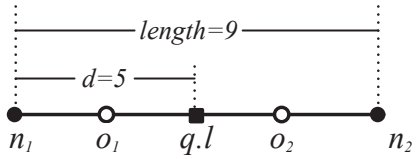



Figure 3.3: Derivation of top-$k$ results.

**Top-$k$ results.** We use Fig. 3.3 to illustrate how the top-$k$ results can be derived from the the union based on Lemma 3.1. Let $d$ denote the distance from $n_1$ to $q.l$ along the edge $(n_1, n_2)$. In the example, the query point is on edge $(n_1, n_2)$ with $d$ of 5 units, and $length = |n_1, n_2|$ of 9 units. We have known from the above subsection that the top-2 results of $n_1$ and $n_2$ are $\{o_9, o_1\}$ and $\{o_9, o_4\}$ respectively. For each object $o$ from the results of both nodes that does not lie on the edge on which the query location lies, we set $d(q.l, o.l)$ to the minimum distance of $(d(n_1, o.l) + d)$ and $(d(n_2, o.l) + length - d)$ and update $o.score$ using $\delta(q.l, o.l)$. For the objects lying on the edge, we compute their scores directly. Then, we select the top-$k$ objects with the lowest scores. Applying this principle to $o_1, o_2, o_4, o_9$, we can see that:

- $\delta(q.l, o_1.l) = \frac{2}{30}$; $\tau(q, o_1) = 0.5 \times \frac{2}{30} + 0.5 \times (1 - \frac{1}{2}) = 0.28$
- $\delta(q.l, o_2.l) = \frac{2}{30}$; $\tau(q, o_2) = 0.5 \times \frac{2}{30} + 0.5 \times (1 - \frac{1}{3}) = 0.37$
- $\delta(q.l, o_4.l) = \frac{min\{14+5,5+9-5\}}{30}$; $\tau(q, o_4) = 0.5 \times \frac{9}{30} + 0.5 \times (1 - \frac{1}{2}) = 0.4$
- $\delta(q.l, o_9.l) = \frac{min\{17+5,8+9-5\}}{30}$; $\tau(q, o_9) = 0.5 \times \frac{12}{30} + 0.5 \times (1 - 1) = 0.2$

Therefore, the top-2 results of $q$ should be $\{o_9, o_1\}$.

**Safe segment.** A straightforward solution to the continuous monitoring of spatial keyword queries on road networks is to periodically invoke the snapshot query algorithm (we use this method as a comparative method in the experiments). However, this method can yield excessive costs. In this work, we adopt a standard server-client architecture with *safe segment* [36, 77], which is defined as follows.

**Definition 3.3 (safe segment)** *A safe segment is a portion of an edge which can guarantee that as long as the client stays in it, its top-k results remain valid.*

In the following, we introduce how a safe segment can be computed. Assume we know the moving direction of the client, as depicted in Fig. 3.4. There are two key observations that support for the computation of the safe segment, which are introduced as follows.

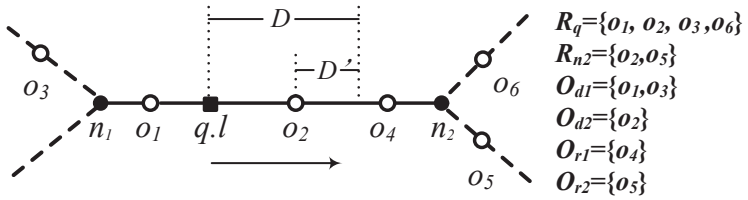



Figure 3.4: Derivation of safe segment.

**Observation 3.1 (the 'replaced' rule)** *When the client moves towards the direction within the edge($n_1$,$n_2$), only two parts of the top-k results $R_q$ can be replaced, which are (i) the objects lying on the left of q.l, and (ii) the objects lying between q.l and $n_2$. We use $O_{d1}$ to denote the first part of objects and $O_{d2}$ to denote the second part of objects.*

**Proof 3.2** *An example of $O_{d1}$ and $O_{d2}$ is shown in Fig. 3.4. When the client moves from q.l toward $n_2$, some objects in $R_q$ may be replaced by other objects. The object o $\in R_q$ may be replaced only when the distance from the new query location to o becomes larger. In this case, the score $\tau(q, o)$ becomes larger. Thus, o may be replaced by other objects whose score $\tau$ becomes smaller. The part of $R_q$ that satisfy the larger distance condition are $O_{d1}$ and $O_{d2}$. For other objects in $R_q$ lying on the right of $n_2$, such as $o_6$, the distance becomes smaller, resulting in a smaller score $\tau$. Thus, we only need to consider $O_{d1}$ and $O_{d2}$, which may be replaced.*

**Observation 3.2 (the 'replace' rule)** *When the client moves towards the direction within the edge($n_1$,$n_2$), only two parts of objects can replace some objects in the top-k results $R_q$, which are (i) the objects lying on the right of $n_2$ which belong to the top-k results of $n_2$, and (ii) the objects lying between q.l and $n_2$. We use $O_{r1}$ to denote the first part of objects and $O_{r2}$ to denote the second part of objects.*

**Proof 3.3** *An example of $O_{r1}$ and $O_{r2}$ is shown in Fig. 3.4. When the client moves from q.l toward $n_2$, some objects may replace the objects in $R_q$. The object o may replace some object in $R_q$ only when the distance from the new query location to o becomes smaller. In this case, the score $\tau(q, o)$ becomes smaller. Thus, o may replace some object in $R_q$ whose score $\tau$ becomes larger. The objects that satisfy the smaller distance condition are $O_{r1}$ and $O_{r2}$. According to Lemma 3.1, we do not need to consider the objects lying on the right of $n_2$ which do not belong to the top-k results of $n_2$.*

According to the two observations, we compute the safe segment in the following two steps.

First, we consider the objects in $O_{d1}$ and the objects in $O_{r1}$ and $O_{r2}$. For every object $o_{d1}$ belonging to $O_{d1}$ and object $o_r$ belonging to $O_{r1}$ or $O_{r2}$, the distance from the new query location to $o_{d1}$ becomes larger and the distance from the new query location to $o_r$ becomes smaller. Thus, we can compute a candidate safe segment $D$ using Equation 3.5. Note that $D$ should be normalized when added to $\delta(q.l, o.l)$.

$$\begin{aligned} \alpha \cdot [\delta(q.l, o_{d1}.l) + \frac{D}{d_{max}}] + (1-\alpha) \cdot [1 - \theta(q.d, o_{d1}.d)] = \\ \alpha \cdot [\delta(q.l, o_r.l) - \frac{D}{d_{max}}] + (1-\alpha) \cdot [1 - \theta(q.d, o_r.d)] \end{aligned} \tag{3.5}$$

Second, we consider the objects in $O_{d2}$ and the objects in $O_{r1}$ and $O_{r2}$. We need to be more careful here. For every object $o_{d2}$ belonging to $O_{d2}$ and object $o_r$ belonging to $O_{r1}$ or $O_{r2}$, only when the client moves to the right of $o_{d2}$ will it be possible for $o_{d2}$ to be replaced, since the distance from $o_{d2}$ to the new query location will become larger from then on. Therefore, we first compute an offset $D'$ that indicates how far the client moves to the right of $o_{d2}$ when $o_{d2}$ is replaced using Equation 3.6. An example is shown in Fig. 3.4. We then get a candidate safe segment $D$ using Equation 3.7 which simply adds the offset $D'$ to the distance of $q$ and $o_{d2}$.

$$\alpha \cdot D' + (1-\alpha) \cdot [1 - \theta(q.d, o_{d2}.d)] =$$

$$\alpha \cdot [\delta(q.l, o_r.l) - \delta(q.l, o_{d2}.l) - D'] + (1-\alpha) \cdot [1 - \theta(q.d, o_r.d)] \tag{3.6}$$

$$D = [\delta(q.l, o_{d2}.l) + D'] \cdot d_{max} \tag{3.7}$$

If $o_r$ belongs to $O_{r2}$, only the positive $D$ value smaller than $|q.l, o_r.l|$ is valid. This is because after the client moves to the right of $o_r$, the distance between the new query location and $o_r$ will become larger. Thus, $o_r$ loses the ability to replace some object in $R_q$. If $o_r$ belongs to $O_{r1}$, only the positive $D$ value smaller than $|q.l, n_2|$ is valid. After the above two steps, we calculate the minimum distance $d_{min}$ from the valid candidate safe segments and $d_{min}$ is the safe segment obtained. What can happen is that there is no valid safe segment at last, which indicates that before the client reaches $n_2$, the top-$k$ results remain valid. In this case, the length of safe segment is $|q.l, n_2|$. Note that as we have reduced the problem to examining static network nodes (Lemma 3.1), the safe segment does not contribute much to computation cost reduction. However, it can greatly reduce the communication cost between the client and the server.

### 3.3.5 Complete QCA Monitoring

In this subsection, we describe the complete QCA monitoring. A standard server-client architecture is adopted to monitor the moving queries. Algorithm 3.3 provides the steps of how the server responses to different messages sent from the client. There are four types of messages: (i) $M_1$ that a client first submits a query (line 2), (ii) $M_2$ that the client exits a safe segment (line 15), (iii) $M_3$ that the client exits an edge (line 17) and (iv) $M_4$ that the client changes direction (line 30). In the following, we introduce how the server responds to each message type.

- For $M_1$, the server takes some initialization steps. First, the spatial component is used to find the edge on which $q.l$ lies (line 3). Then, the moving direction $q.r$ of the client is used to locate the next encountered node (lines 4-5). Finally, polyline of the edge is used to compute network distances between $q.l$ and the end nodes of the edge (line 6). After the initialization steps, the server computes the result sets of the two end nodes and then gets safe segment as described in the last subsection. The server sends the top-$k$ results and safe segment to the client (line 14).

- It is easy for the server to process $M_2$. As the client does not leave the edge, the server just recomputes the top-$k$ results and safe segment (line 16).

- When the client moves out of the edge, there are two cases. If the new edge is adjacent with the old edge (line 22), the result set of their intersecting node does not need to be recomputed. The server can just reuse the result set it computed last time. In order to guarantee the reuse, when processing $M_1$ and $M_3$, the server locates the intersecting node using the moving direction of the client (lines 4-5 and lines 19-20). Then the server reuses the result set of the intersecting node and computes the result set of the other node (lines 23-24). There is also a possibility that the client moves to a new edge which is not adjacent with the old edge due to some unpredictable factor, such as the fast speed of the client or the communication problem between the client and the sever. In this case, the server recomputes the result set of the two end nodes using the expansion tree $\mathcal{T}$ (lines 26-29).

- The algorithm can also process the situation when the client changes moving direction. The server first relocates the next encountered node (lines 31-32), and then recomputes the top-$k$ results and safe segment (line 33) due to the fact that the client still moves on the same edge.

**Algorithm 3.3:** QCA Monitoring

**environment**: Graph $G$, Objects $O$

1 **foreach** *moving client* $c$ **do**
2  **if** *receive* $M_1$ *with query* $q =< q.l, q.d, q.k, q.r >$ **then**
3  $e \leftarrow$ network edges on which $q.l$ lies;
4  $n_1 \leftarrow$ the next encountered node using $q.r$;
5  $n_2 \leftarrow$ the other node of $e$;
6  compute $|q.l, n_1|$ and $|q.l, n_2|$;
7  $R_{n_2} \leftarrow SnapshotQueryResult(n_2, (n_1, n_2))$;
8  delete the invalid part of $\mathcal{T}$;
9  insert the objects in $O_{re}$ falling in $\mathcal{T}$ to $R_{n_1}$;
10  insert the leaves of $\mathcal{T}$ to $\mathcal{N}$;
11  $R_{n_1} \leftarrow SnapshotQueryResult(n_1, (n_1, n_2))$;
12  $R_q \leftarrow$ get results using $R_{n_1}$, $R_{n_2}$ and $|q.l, n_1|$;
13  $S \leftarrow$ safe segment;
14  send $R_q$ and $S$ to the client;
15  **if** *receive* $M_2$ **then**
16  lines 12-14;
17  **if** *receive* $M_3$ **then**
18  $e' \leftarrow$ network edges on which $q.l_{new}$ lies;
19  $n_1 \leftarrow$ the next encountered node using $q.r$;
20  $n_2 \leftarrow$ the other node of $e$;
21  compute $|q.l_{new}, n_1|$ and $|q.l_{new}, n_2|$;
22  **if** $e'$ *is adjacent with* $e$ **then**
23  $R_{n_2} \leftarrow R_{n_1}$;
24  lines 8-14;
25  **else**
26  delete the invalid part of $\mathcal{T}$;
27  insert the objects in $O_{re}$ falling in $\mathcal{T}$ to $R_{n_2}$;
28  insert the leaves of $\mathcal{T}$ to $\mathcal{N}$;
29  lines 7-14;
30  **if** *receive* $M_4$ **then**
31  $n_1 \leftarrow$ the next encountered node using $q.r_{new}$;
32  $n_2 \leftarrow$ the other node of $e$;
33  lines 12-14;

## 3.4 Object-centric Algorithm

In this section, we present our second method named *object-centric algorithm* (OCA). We start with the basic idea of our query processing in Section 3.4.1, and then in Section 3.4.2 we introduce the network additively weighted Voronoi diagram which can be used for monitoring continuous spatial keyword queries. To support obtaining

top-$k$ results instead of a single result, we present the order-$k$ shortest path tree (kSPT) used in OCA in Section 3.4.3 and show how a kSPT can be incrementally constructed in Section 3.4.4. Finally, we give the complete OCA monitoring algorithm in Section 3.4.5.

### 3.4.1 Basic Idea

A shortcoming of using QCA for the MkSK queries is that it has to reevaluate the query results of each unvisited node encountered by the moving query point, since a new node may make some shortest paths invalid. Although it can avoid repetitive traversing of some network edges using expansion tree, QCA still has to traverse many edges that belong to the invalid part of the expansion tree. In this section, we introduce a different approach that applies a special property of the studied continuous monitoring problem: the textual relevance $\theta$ is independent of the query point movement. This motivates us to utilize the concept of *network additively weighted Voronoi diagram* (NAWVD) for monitoring spatial keyword queries.

At the beginning, OCA loads relevant objects that match at least one of the query keywords in their descriptions. Instead of traversing from the end nodes of the edge on which the query location lies, OCA starts traversing the network from a relevant object and constructs a shortest path tree. We get the results of the two end nodes of the edge on which the query location lies using the shortest path tree. Moreover, the incremental shortest path tree construction is characterized by allowing the construction process to halt when the top-$k$ results of the requested node are found and to resume when more results are required. OCA does not suffer from repetitive result evaluation, since query results of each node are obtained via object-centric expansion, where the shortest paths remain valid. Moreover, when constructing the shortest path tree, OCA also obtains results or partial results of surrounding nodes, which makes it suitable for the MkSK queries.

### 3.4.2 Network AW-Voronoi Diagram

Ordinary Voronoi diagram over a set of $n$ points (or called *generators*) is a partition of the space into $n$ disjoint *Voronoi cells*, where the nearest neighbor of any point inside a Voronoi cell is the generator of that Voronoi cell. Network Voronoi diagram [56, 42] can be analogously constructed to partition the space into network Voronoi cells (or *Voronoi edge sets*), by restricting objects on edges that connect nodes and considering network distance.

The weighted Voronoi diagram family (including multiplicatively, additively, compoundly, etc.) differs from the ordinary Voronoi diagram in that the generators do not have the same weight, reflecting their variable properties [56]. In our problem, in order to find the result for each individual query $q$, we consider not only the spatial proximity measure $\delta(q.l, o.l)$ but also the textual relevance measure $\theta(q.d, o.d)$, and the latter can be regarded as a *weight* to be considered in addition to the former. Thus, we can utilize *additively weighted Voronoi diagram* (or in short, AW-Voronoi diagram) for processing the MkSK queries. By regarding the set of objects $O_r = \{o_1, \ldots, o_n\}$ where $o_i \neq o_j$ for $i \neq j$, $i, j \in I_n = \{1, \ldots, n\}$ as weighted generators[2], *additively weighted distance* between a point $p$ and one of the generators $o_i$ can be written as

$$\begin{aligned} D_{AW}(p, o_i) &= D(p.l, o_i.l) + \omega_i \\ &= \delta(p.l, o_i.l) + \frac{1-\alpha}{\alpha}[1 - \theta(q.d, o_i.d)] \end{aligned} \tag{3.8}$$

where $\omega_i$ is the designated weight of the generator $o_i$, corresponding to $(1-\alpha)/\alpha[1-\theta(q.d, o_i.d)]$ (which is a non-spatial attribute associated with $o_i$, and independent of the query point movement).

Given a road network $G(N, E)$, a set of edges $E = \{e_1, \ldots, e_m\}$ connect the nodes $N$ in $G$. For $j \in I_n - \{i\}$, the dominance region of $o_i$ over $o_j$ on edges specifies all locations on the edges in $E$ that are closer to $o_i$ or of equal distance to $o_j$ with additively weighted network distance, which is

$$Dom(o_i, o_j) = \{p | p \in \bigcup_{l=1}^{m} e_l, D_{AW}(p, o_i) \leq D_{AW}(p, o_j)\}. \tag{3.9}$$

The network Voronoi cell generated by $o_i$ is the closures of these regions and can be defined as

$$V_{edge}(o_i) = \bigcap_{j \in I_n \setminus \{i\}} Dom(o_i, o_j). \tag{3.10}$$

The network AW-Voronoi diagram over a set of generators $O$ partitions $E$, and each network Voronoi cell $V_{edge}(o_i)$ includes edges or portions of edges. A network AW-Voronoi cell $V_{edge}(o_i)$ specifies all the locations on the network where $o_i$ should be the top ranked object of the spatial keyword query. This is because the generator $o_i$ of the network AW-Voronoi cell that contains the query location $q.l$ has the smallest value of $D_{AW}(q, o)$ among all generators, while $D_{AW}(q, o) = \tau(q, o)/\alpha$ according to Equations 3.1 and 3.8.

[2]Note that for the MkSK queries, the generators are the relevant objects in $O_r$ that match at least one of the query keywords in their descriptions.

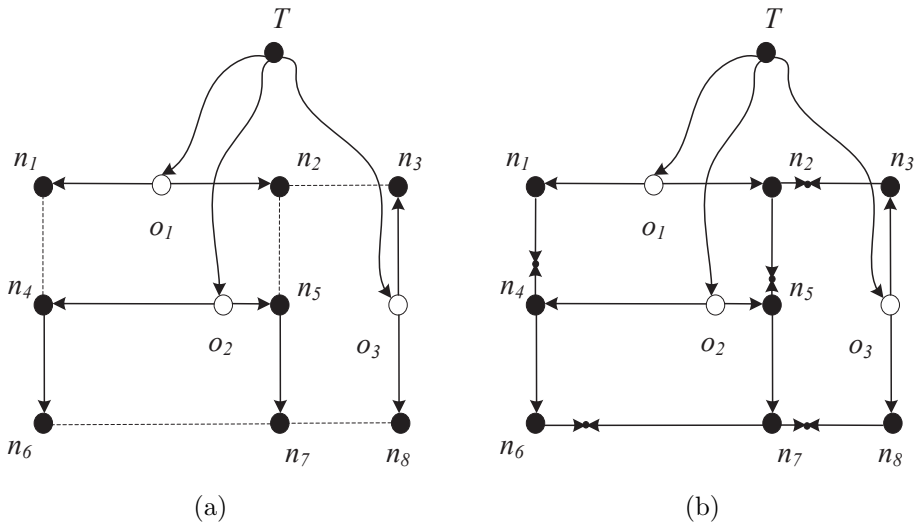



Figure 3.5: Network AW-Voronoi diagram construction using the extended shortest path tree technique.

Next, we briefly describe how a network AW-Voronoi diagram can be constructed based on a technique using the extended shortest path trees [57]. The shortest path tree (SPT), defined as the set of edges connecting all nodes such that the sum of the edge lengths from a given root to each node is minimum. A simple example is shown in Fig. 3.5(a), where the SPT rooted at $T$, $SPT(T)$, is signified by the line segments. The edges signified by the dotted line segments which are referred to as *uncovered edges* are not included in the SPT. To cover the whole network, the shortest path tree is extended as shown in Fig. 3.5(b) in the following manner. On each uncovered edge, we first find a *break point*, $b_i$, such that the shortest path distance from the point $b_i$ to the root node through one end node of the uncovered edge is equal to the shortest path distance to the root node through the other end node, as signified by the small black dots in the figure. We then cut the network at these break points and add nodes on both cut ends. For this modified network, we again construct the ordinary SPT. The result is called the *extended shortest path tree* (ESPT). It gives the information on the shortest path from any arbitrary point on the network to a given root node. Given a road network $G = (N, E)$, the network AW-Voronoi diagram can be obtained using the ESPT. First, we assume a dummy node $T$ as the root node and join $T$ to each generator $o_i$ with the length $\frac{(1-\alpha)}{\alpha}[1 - \theta(p.d, o_i.d)]$ as the weight. Then, we construct the extended shortest path tree of $T$.

There are two major limitations of using the above NAWVD construction technique for the MkSK queries. First, it can only handle a *single* object (top ranked) but

cannot support top-$k$ results. Second, it requires to construct the *whole* network AW-Voronoi diagram. In the following, we generalize the SPT technique to the order-$k$ SPT (kSPT), and show how kSPT can be constructed in an incremental manner.

### 3.4.3 Order-$k$ Shortest Path Tree

In order to support obtaining top-$k$ results, we can extend SPT by introducing overlaps between branches. The kSPT branches are overlapped in such a way that each node appears in the tree exactly $k$ times, in $k$ different branches. We first introduce the data structure used by our algorithm of constructing kSPT.

**Indexing structure.** OCA uses the spatial component and the adjacency component (introduced in Section 3.3.2) to locate the edges on which the objects lie and to get the adjacent nodes of a given node, respectively. Besides, our algorithm uses an inverted index to load some objects, which match at least one of the query keywords in their descriptions, at the beginning of the algorithm. We use a B-tree that maps the *term id* to the inverted list that contains the objects with term $t$ in their documents.

**Node structure.** The structure of a node $n_i$ contains the following attributes:

- *ID*: the node identification;
- *LabelList*: a list of (at most) $k$ labels. For each label $(o, d)$ in the label list of $n_i$, $o$ represents an object, and labeling distance $d$ represents $\frac{\tau(q,o)}{\alpha}$;
- *Type*: a node type is 'Labelable' by default and becomes 'Permanent' upon completion of $k$ labels.

Fig. 3.6 shows the order-2 shortest path tree for the network shown in Fig. 3.1, where the X coordinate represents the labeling distance $d$. $T$ is the dummy root node as illustrated in Fig. 3.5. We construct the order-$k$ shortest path tree similar with SPT, but we stop the construction until each node in the network appears in the tree exactly twice, in two different branches. Each generator (i.e., $o_9$, $o_4$ and $o_1$) has a branch. The first label indicates the labeling distance between the corresponding node and the generator whose branch the node locates on is the shortest, while the second label indicates a second shortest labeling distance. For example, $n_1$ first appears in the branch of $o_9$ and then in the branch of $o_1$. Thus, the top-2 results of $n_1$ should be $o_9$ and $o_1$.

We describe kSPT construction steps in Algorithm 3.4. The algorithm receives a query $q$ and a set $O_r$ of relevant objects that match at least one of the query keywords

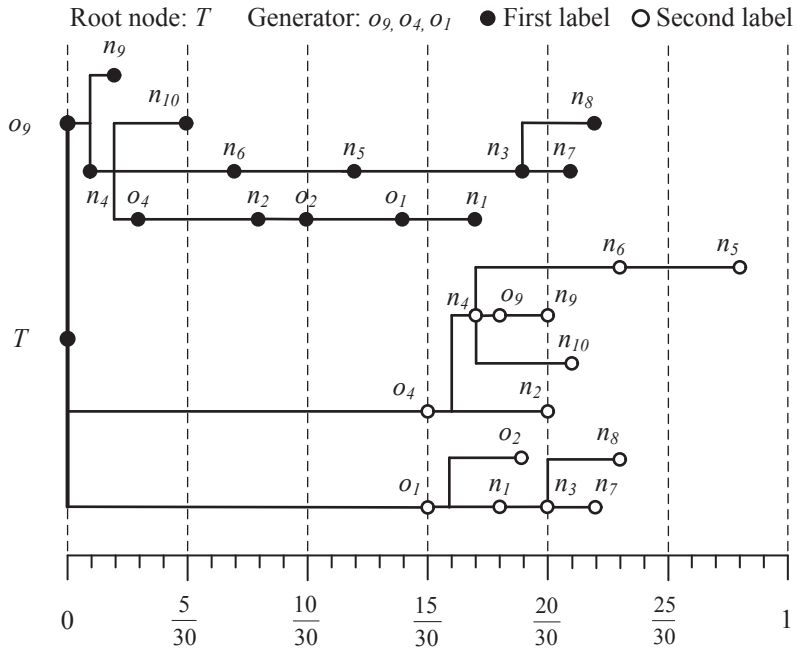



Figure 3.6: Order-2 shortest path tree for the network in Fig. 3.1.

in their descriptions as input. The output kSPT is provided as $G(N, E)$ with top-$k$ result information embedded. Specifically, we obtain $k$ labels for each $n_i$ in $N$. The initialization (lines 1-10) includes the following steps.

- First, a priority queue $PQ$ is initialized. An entry of $PQ$ is a tuple $(n, o, d)$, where $n$ is the node to which the entry corresponds, and the other two elements $o$ and $d$ form a labeling candidate for an entry in $n$.LabelList. Entries in $PQ$ are ranked according to the labeling distance $d$.

- Second, for each object $o \in O_r$, we create a node entry $n_o$ and insert it into $G(N, E)$ where affected edges in $E$ are accordingly modified (lines 3-4). We create an entry of $PQ$ for $n_o$ with the associated object $o$ and the labeling distance $d$ of $\frac{1-\alpha}{\alpha}[1 - \theta(q.d, o.d)]$ (lines 5-7).

- Third, for each network node, we create an empty label list and set the node type to 'Labelable' (lines 8-10).

Best-first search is handled by the `while` loop (lines 11-20). The first step of each iteration is to dequeue the head entry $(n, o, d)$ from $PQ$ (line 12). Since we are interested in only the first $k$ labels of each node, the entry is ignored if the label list of the node already contains $k$ labels, i.e., the node is 'Permanent'. In addition, to ensure that each node is associated with $k$ unique results, the entry is also ignored if

**Algorithm 3.4:** Construct-kSPT

**environment**: Graph $G(N, E)$, MinHeap $PQ$
**input** : Query $q$, Objects $O_r$
**output** : Labeled $G$

```
1  Initialize PQ;
2  foreach o ∈ O_r do
3  |  Node n_o ← create a network node from o;
4  |  G(N, E).Insert(n_o);
5  |  Distance d ← (1−α)/α [1 − θ(q.d, o.d)];
6  |  PQEntry e ← tuple (n_o, o, d);
7  |  insert e into PQ;
8  foreach n ∈ N do
9  |  n.LabelList ← create an empty list of labels;
10 |  n.Type ← Labelable;
11 while PQ is not empty do
12 |  PQEntry (n, o, d) ← PQ.pop();
13 |  if n.Type is Labelable and no existing label with o then
14 |  |  n.LabelList.Add((o, d));
15 |  |  if n.LabelList.Length = k then
16 |  |  |  n.Type← Permanent;
17 |  |  foreach adjacent node n_a of n and n_a is Labelable do
18 |  |  |  Distance ω ← (n_a,n).Weight / d_max;
19 |  |  |  PQEntry e_a ← tuple (n_a, o, d + ω);
20 |  |  |  PQ.Insert(e_a);
```

there exists an entry with object $o$ as the associated object in the label list. Otherwise, a label $(o, d)$ is added to the label list of the node (line 14). The node type becomes 'Permanent' if this label is the $k^{th}$ entry in the label list (lines 15-16). In lines 18 to 20, for each node $n_a$ adjacent to $n$, we create a $PQ$ entry $e_a$, $(n_a, o, d + \omega)$, where

- $n_a$ is the node to which this entry corresponds;
- $o$ is the associated object;
- $(d+\omega)$ is the labeling distance calculated by adding the current labeling distance $d$ to the normalized weight of edge (i.e., divided by $d_{max}$) between $n$ and $n_a$.

The entry $e_a$ is then inserted into $PQ$. The `while` loop continues until $PQ$ is exhausted, i.e., every node is labeled $k$ times.

Let us consider the first few steps of the algorithm, using the network in Fig. 3.1. Likewise, we also assume the maximum distance $d_{max}$ used to normalize $\delta$ is 30 units

and $\alpha$ is 0.5. After the initialization steps, the priority queue $PQ$ has the following initial entries:

$$\left[(o_9, o_9, 0), (o_4, o_4, \frac{1}{2}), (o_1, o_1, \frac{1}{2}), (o_2, o_2, \frac{2}{3})\right].$$

Then the first entry $(o_9, o_9, 0)$ is dequeued from $PQ$. As a result, node $o_9$ is labeled with $o_9$ itself as the first object of interest and the labeling distance of 0, which is obtained using $\frac{1-\alpha}{\alpha}[1 - \theta(q.d, o_9.d)]$. Next, two entries $(n_4, o_9, \frac{1}{30})$ and $(n_9, o_9, \frac{2}{30})$ are created using the two nodes adjacent to $o_9$, $n_4$ and $n_9$, respectively. These entries are then inserted into $PQ$, resulting in the following entries in $PQ$:

$$\left[(n_4, o_9, \frac{1}{30}), (n_9, o_9, \frac{2}{30}), (o_4, o_4, \frac{1}{2}), (o_1, o_1, \frac{1}{2}), (o_2, o_2, \frac{2}{3})\right].$$

The entry that is dequeued next is $(n_4, o_9, \frac{1}{30})$, so we apply the label $(o_9, \frac{1}{30})$ to $n_4$. The same process continues until $PQ$ is exhausted.

A drawback of Algorithm 3.4 is that it requires global access to all nodes, which can be disadvantageous especially in a large network. Next, we show how this drawback can be alleviated.

### 3.4.4 Incremental kSPT Construction

We now present our OCA approach which incrementally retrieves objects and computes node labels as the monitoring process progresses. The cost of order-$k$ shortest path tree construction can be greatly reduced by exploiting the fact that the offset assigned to each object $o$ corresponds to the textual relevance $\frac{1-\alpha}{\alpha}[1-\theta(q.d, o.d)]$. Thus, objects that are not very relevant to $q.d$ are likely to be involved in the computation later than objects relevant to $q.d$. Based on this property, we devise a mechanism which is incremental (it halts when a desired label list is obtained and resumes when more label lists are required). Incremental kSPT construction usually requires only local information. As a result, we can eliminate the global access requirement of the network nodes and limit the search region to a much smaller size.

As the labeling process progresses, objects are incrementally retrieved according to their textual relevance to $q.d$ (the most relevant object is first retrieved). The scope of this object retrieval is denoted as a search radius $r$, where $r$ is set to $\frac{1-\alpha}{\alpha}[1-\theta(q.d, o.d)]$. Whenever a new object $o$ is retrieved, $r$ is updated. The value of $r$ indicates whether a node is safe to label or more objects are needed.

Algorithm 3.5 provides detailed steps of how the label list of a node is obtained. The first step is to check whether the $k$ labels of the node already exist, in which case the labels are returned straightaway without further traversal of the network (lines

**Algorithm 3.5:** GetLabelList(Node $s$)

**environment**: Graph $G(N, E)$, Objects $O_r$, Query $q$, MinHeap $PQ$
**input** : Node $s$
**output** : LabelList $\langle l_1, \ldots, l_k \rangle$

```
1  if s has been initialized and s.Type is Permanent then
2  |  return s.LabelList;
3  while PQ is not empty do
4  |  PQEntry (n, o, d) ← PQ.head();
5  |  while r < d do
6  |  |  Object o ← O_r.pop() ;
7  |  |  Node n_o ← create a network node from o;
8  |  |  G(N, E).Insert(n_o);
9  |  |  PQEntry e ← tuple (n_o, o, (1−α)/α [1 − θ(q.d, o.d)]);
10 |  |  insert e into PQ;
11 |  |  r = (1−α)/α [1 − θ(q.d, o.d)];
12 |  PQEntry (n, o, d) ← PQ.pop();
13 |  if n.Type is Labelable and no existing label with o then
14 |  |  if n is not initialized then
15 |  |  |  n.LabelList ← create an empty list of labels;
16 |  |  |  n.Type ← Labelable;
17 |  |  n.LabelList.Add((o, d));
18 |  |  foreach adjacent node n_a of n and n_a is Labelable do
19 |  |  |  Distance ω ← (n_a,n).Weight / d_max;
20 |  |  |  PQEntry e_a ← tuple (n_a, o, d + ω);
21 |  |  |  insert e_a into PQ;
22 |  |  if n.LabelList.Length = k then
23 |  |  |  n.Type← Permanent;
24 |  |  |  if n.ID=s.ID then
25 |  |  |  |  return n.⟨L_1, ..., L_k⟩;
```

1-2). If the requested label list is otherwise incomplete, we proceed to the main `while` loop (lines 3-25). The `while` loop in Algorithm 3.5 is similar to that in Algorithm 3.4. The following modifications are applied to make Algorithm 3.5 incremental.

- The first modification is the search radius check (lines 5-11), which ensures that the value of $r$ is not smaller than the labeling distance $d$. Specifically, until $r$ is larger than or equal to $d$, the following steps are repeated:
  - retrieving the next relevant object with respect to $q.d$ (line 6),
  - performing graph modification and priority queue insertion (lines 7-10) similar to Algorithm 3.4,

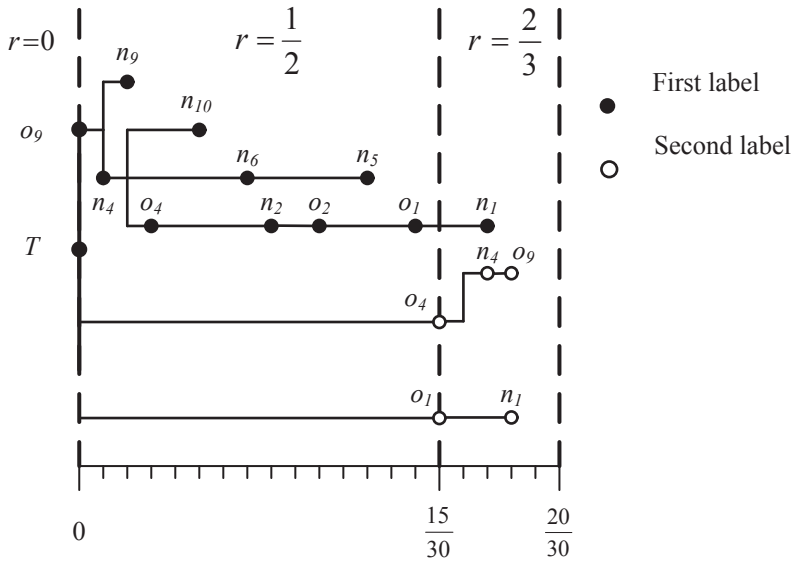



Figure 3.7: kSPT constructed when getting the top-2 results of $n_1$.

– setting the search radius $r$ to $\frac{1-\alpha}{\alpha}[1-\theta(q.d, o.d)]$ of object $o$ (line 11);

- The second modification is deferral of node initialization (lines 14-16);
- The third modification is a halt to the node labeling process after the requested node has $k$ labels (lines 24-25).

Fig. 3.7 presents a stepped explanation of how the $k$ labels of $n_1$ can be computed though incremental object retrieval and incremental node labeling.

- The first retrieved object (the object most relevant with the query keywords) is $o_9$. The search radius $r$ is set to $\frac{1-\alpha}{\alpha}[1-\theta(q.d, o_9.d)] = 0$. The only node that can be labeled with this $r$ value is $o_9$ itself. After the labeling, we proceed to consider the next object.
- The next retrieved object is $o_4$. The search radius $r$ is updated to $\frac{1}{2}$, which allows the nodes belonging to $r = \frac{1}{2}$ and $o_4$ itself to be labeled. After that, the object $o_1$ is retrieved. As the search radius $r$ remains to be $\frac{1}{2}$, only $o_1$ is labeled.
- Then $o_2$ is retrieved. The search radius $r$ is updated to $\frac{2}{3}$, which allows the nodes belonging to $r = \frac{2}{3}$ to be labeled. The labeling process halts upon completion of the second label of $n_1$. From the figure, we can see that $n_1$ appears first in the branch of $o_9$ and again in the branch of $o_1$. We can therefore infer that the top-2 result set of $n_1$ is $\{o_9, o_1\}$.

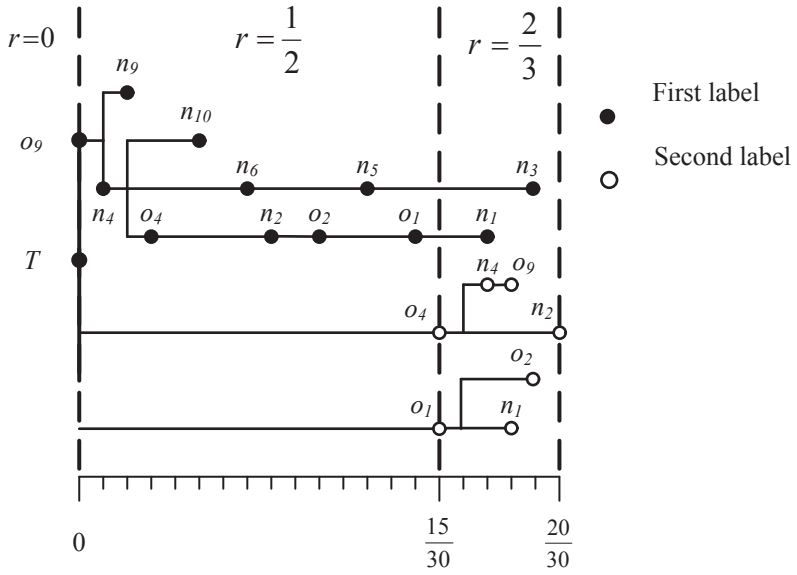



Figure 3.8: kSPT constructed when getting the top-2 results of $n_2$.

Compared with Fig. 3.6 where the whole order-2 shortest path tree is built, Fig. 3.7 only need to build a part of the order-2 shortest path tree, and thus can reduce the edge traversing cost especially in a large network.

Fig. 3.8 shows how a subsequent top-2 result set of the other end node $n_2$ of the edge on which the query location lies can be obtained. As can be seen, the second label of $n_2$ can be obtained by resuming the labeling process halted after the completion of the second label of $n_1$. The figure also shows that we only need to label $o_2$, $n_3$ and $n_2$ before obtaining the second label of $n_2$. Likewise, we know the top-2 result set of $n_2$ is $\{o_9, o_4\}$.

### 3.4.5 Complete OCA Monitoring

In this subsection, we describe the complete OCA monitoring, which is shown in Algorithm 3.6. The OCA algorithm has the following parameters as input: a network graph $G$ and a set of objects $O$. The initialization steps (lines 2-8) include:

- loading the relevant objects into a priority queue $O_r$, which is accomplished by first computing the textual relevance $\theta$ for the relevant objects using the inverted index and then inserting them into $O_r$;
- retrieving the most relevant object to $q.d$ as the initial object $o$ which provides a bound of search space, and inserting it into $G(N, E)$ and $PQ$;
- setting the search radius $r$ to $\frac{1-\alpha}{\alpha}[1 - \theta(q.d, o.d)]$.

**Algorithm 3.6:** OCA Monitoring

**environment**: Graph $G$, Objects $O$

1 **foreach** *moving client* $c$ **do**
2 $O_r \leftarrow$ load relevant objects;
3 $o \leftarrow O_r.pop()$ ;
4 $r = \frac{1-\alpha}{\alpha}[1 - \theta(q.d, o.d)]$;
5 Node $n_o \leftarrow$ create a network node from $o$;
6 $G(N, E)$.Insert($n_o$);
7 PQEntry $e \leftarrow$ tuple $(n_o, o, \frac{1-\alpha}{\alpha}[1 - \theta(q.d, o.d)])$;
8 insert $e$ into $PQ$;
9 **while** *receive message of first query* **or** *exiting the safe segment* **do**
10 $(n_i, n_j) \leftarrow$ network edge on which $q.l$ lies;
11 LabelList $L_i \leftarrow$ GetLabelList($n_i$);
12 LabelList $L_j \leftarrow$ GetLabelList($n_j$);
13 $R_q \leftarrow$ get results using LabelList $L_i$ and LabelList $L_j$;
14 $S \leftarrow$ safe segment;
15 send $R_q$ and $S$ to the client

After the initialization steps, the OCA algorithm enters the monitoring stage (lines 9-15). Once it receives the message from the client when the client first submits a query or exits the safe segment, it locates the edge on which $q.l$ lies and obtains the top-$k$ result sets for the two end nodes using Algorithm 3.5 (lines 11-12). Since OCA uses shortest path tree instead of expansion tree as used in QCA, the server does not need the direction information of the client. Note that only if the client enters a new edge, do the top-$k$ result sets for the two end nodes need to be recomputed. Moreover, with the help of shortest path tree, only a few more edges need to be traversed. Similar to QCA, OCA also makes use of Lemma 3.1 to examine network nodes. From the labels of two end nodes of the edge on which the query location lies, we can derive query results easily using the method presented in Section 3.3.4 (line 13). OCA computes the safe segment using the same method as used in QCA (line 14). To provide a more comprehensive cost comparison of the above mentioned methods, experimental results are reported in the next section.

## 3.5 Experimental Study

In this section, we evaluate the efficiency of QCA and OCA. To test the advantage of using expansion tree in QCA, we implement a baseline method called *incremental network expansion* (INE), which is essentially QCA without using expansion tree.

To test the advantage of incrementally constructing the shortest path tree used in OCA, we implement a baseline method called *order-k shortest path tree* (kSPT), which corresponds to the non-incremental Algorithm 3.4 in Section 3.4.3. We also implement a straightforward method (STM) for MkSK queries on road networks that computes query results from scratch at every timestamp using the snapshot query algorithm similar to Algorithm 3.1 (the snapshot query algorithm does not maintain the expansion tree, and can start a query from any location on the network). We can understand the effect of using Lemma 3.1 and safe segment by comparing STM and INE. These five methods return the same result set for a query and are all implemented in Java.

### 3.5.1 Experimental Setup

We use three real datasets, Singapore, London and Australia, for evaluation. Singapore is obtained from the Land Transport Authority, Singapore. The original road network has 57,138 edges. We reformatted the network to make the endpoints of its edges have degree equal to 1 or larger than 2, resulting in a network with 15,076 edges. London and Australia datasets are obtained from [62], and the formats already satisfy our requirement. Table 3.2 presents some characteristics of each dataset.

We use Brinkhoff's generator [10] to generate the trajectories of moving queries. The input of the generator is the road network of the dataset used. The output is a set of objects (e.g., cars or pedestrians) moving on the network, where each object is represented by its location at consecutive timestamps. An object appears on a network node, completes the shortest path to a random destination, and then disappears. We generate each trajectory with 100 points, where each location is generated per timestamp (second), i.e., we monitor each trajectory for 100 seconds. The keyword set of each query is generated based on the word distribution of the vocabulary dataset used. Each experiment has 100 moving queries with such trajectories and the average result is reported.

Table 3.3 shows the main parameters and values used throughout the experiments (default values are in bold). In the experiments, we measure the following metrics:

- (i) *execution time*, which is the amount of time an algorithm runs to process a query trajectory;
- (ii) *edges expanded*, which is the number of edges expanded before an algorithm finishes processing;

| Attribute | Singapore | London | Australia |
|---|---|---|---|
| Total size | 5 MB | 51 MB | 560 MB |
| Total no. of nodes | 13,023 | 203,383 | 1,181,142 |
| Total no. of edges | 15,076 | 274,947 | 1,631,421 |
| Avg. no. of lines per edge | 3.79 | 5.79 | 13.65 |
| Avg. edge length (m) | 175.00 | 105.12 | 740.47 |
| Total no. of objects | 5,387 | 34,162 | 69,884 |
| Avg. no. of objects per edge | 0.35 | 0.12 | 0.04 |
| Total no. of words | 17,049 | 121,049 | 225,865 |
| Total no. of distinct words | 1,007 | 12,551 | 18,875 |
| Avg. no. of distinct words per object | 0.20 | 3.35 | 3.04 |

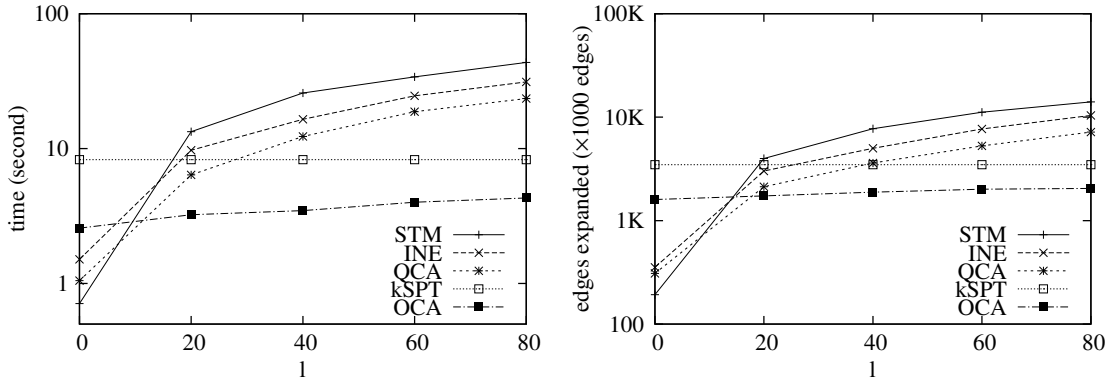
Table 3.2: Characteristics of the datasets.

| Parameter | Values |
|---|---|
| Monitoring length $l$ | 0, 20, **40**, 60, 80 |
| Number of keywords $n$ | 1, 2, **3**, 4, 5 |
| Number of results $k$ | **5**, 15, 25, 35, 45 |
| Preference parameter $\alpha$ | 0.1, 0.3, 0.5, **0.7**, 0.9 |
| Dataset | Singapore, **London**, Australia |

Table 3.3: Parameters evaluated in the experiments.

- (iii) *memory cost*, which is the memory space consumed to process a query trajectory;
- (iv) *communication cost*, which is the total number of objects transferred by the server to the client;
- (v) *communication frequency*, which is the probability of sending a request to the server, and can be computed by the ratio of the number of messages sent to the server by the client to the number of timestamps of the query trajectory.

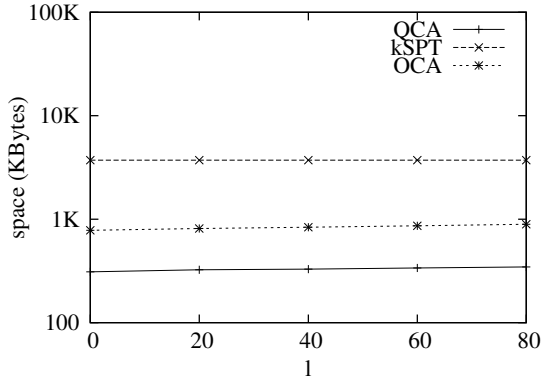
As a remark, the main memory cost for QCA is the maintaining of the expansion tree, and the main memory cost for kSPT and OCA is the maintaining of the relevant objects and the order-$k$ shortest path tree. For other algorithms, the memory cost is low. Thus, we only test the memory cost for QCA, kSPT and OCA. In addition, since all testing algorithms except STM use safe segments and have the same communication cost and communication frequency, we only test one of these algorithms and compare the results with those of STM. Note that for STM, the communication frequency is always 1, since the client will send a location update message to the



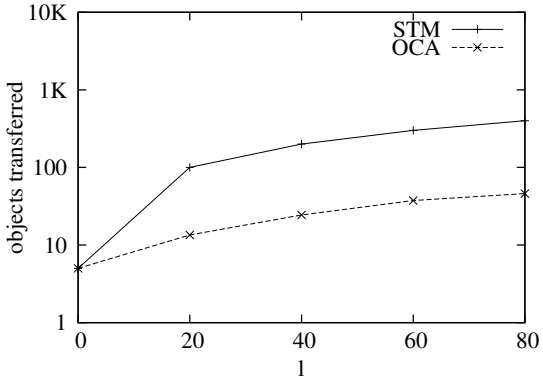

(a) Execution time



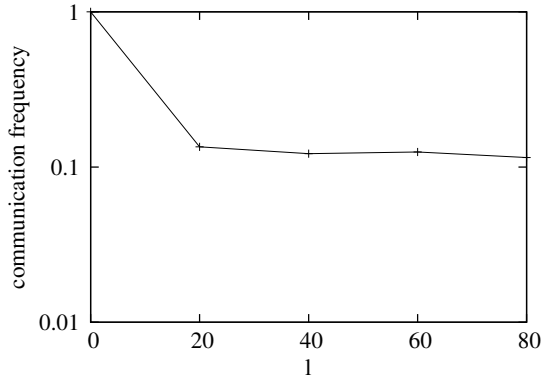

(b) Edges expanded




(c) Memory cost



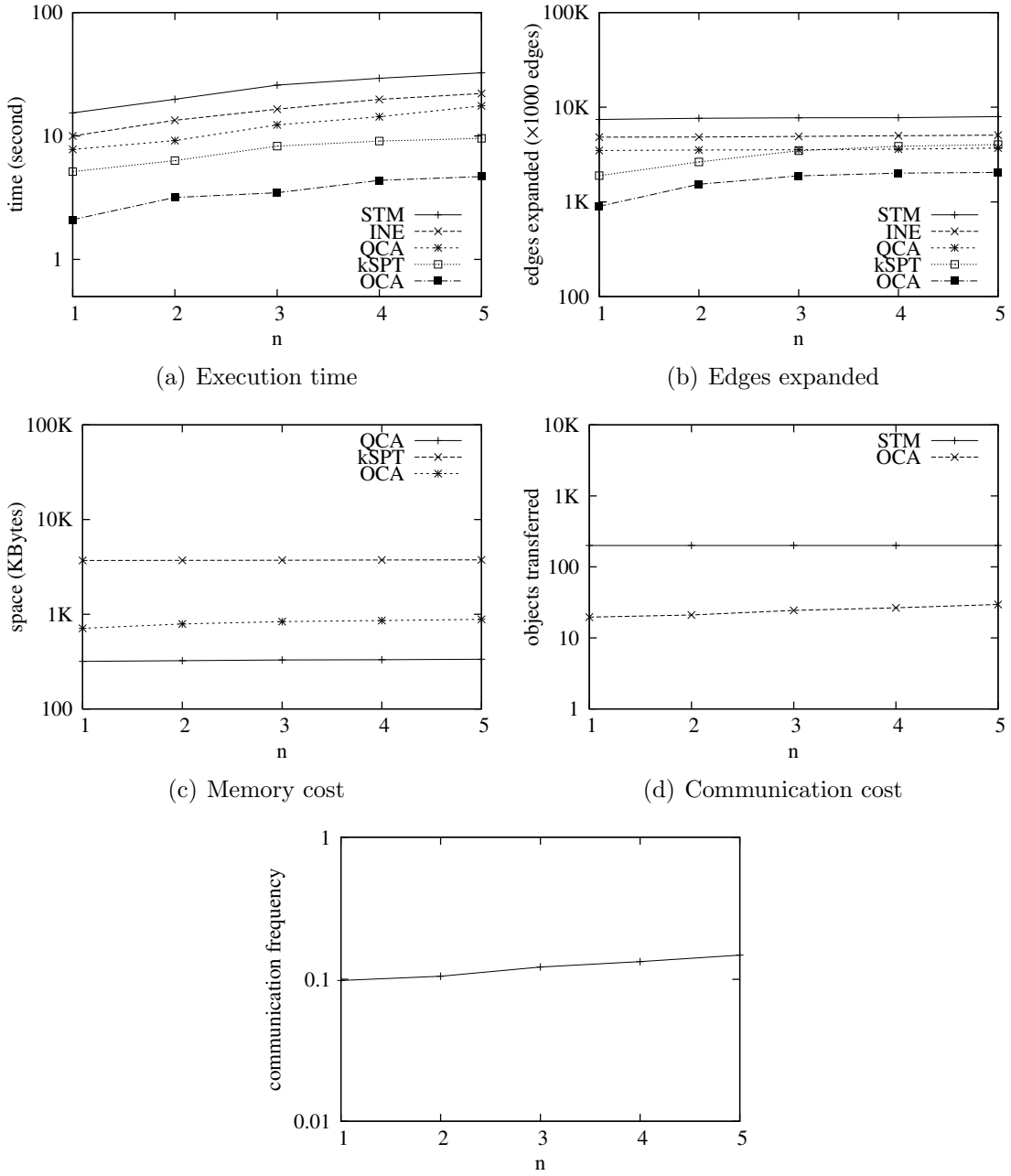

(d) Communication cost




(e) Communication frequency

Figure 3.9: Effect of the monitoring length ($l$).

server at each timestamp. Our experiments are conducted on a 3.20 GHz Intel Core i5 machine with 4 GB of RAM.




(a) Execution time

(b) Edges expanded

(c) Memory cost

(d) Communication cost

(e) Communication frequency

Figure 3.10: Effect of the number of keywords ($n$).

### 3.5.2 Experimental Results

In this subsection, we report the experiment results of STM, INE, kSPT, QCA and OCA methods.

**Effect of $l$.** In this experiment, we evaluate the effect of the monitoring length $l$, as shown in Fig. 3.9. The value of $l$ is ranged from 0 to 80 points. The value of 0

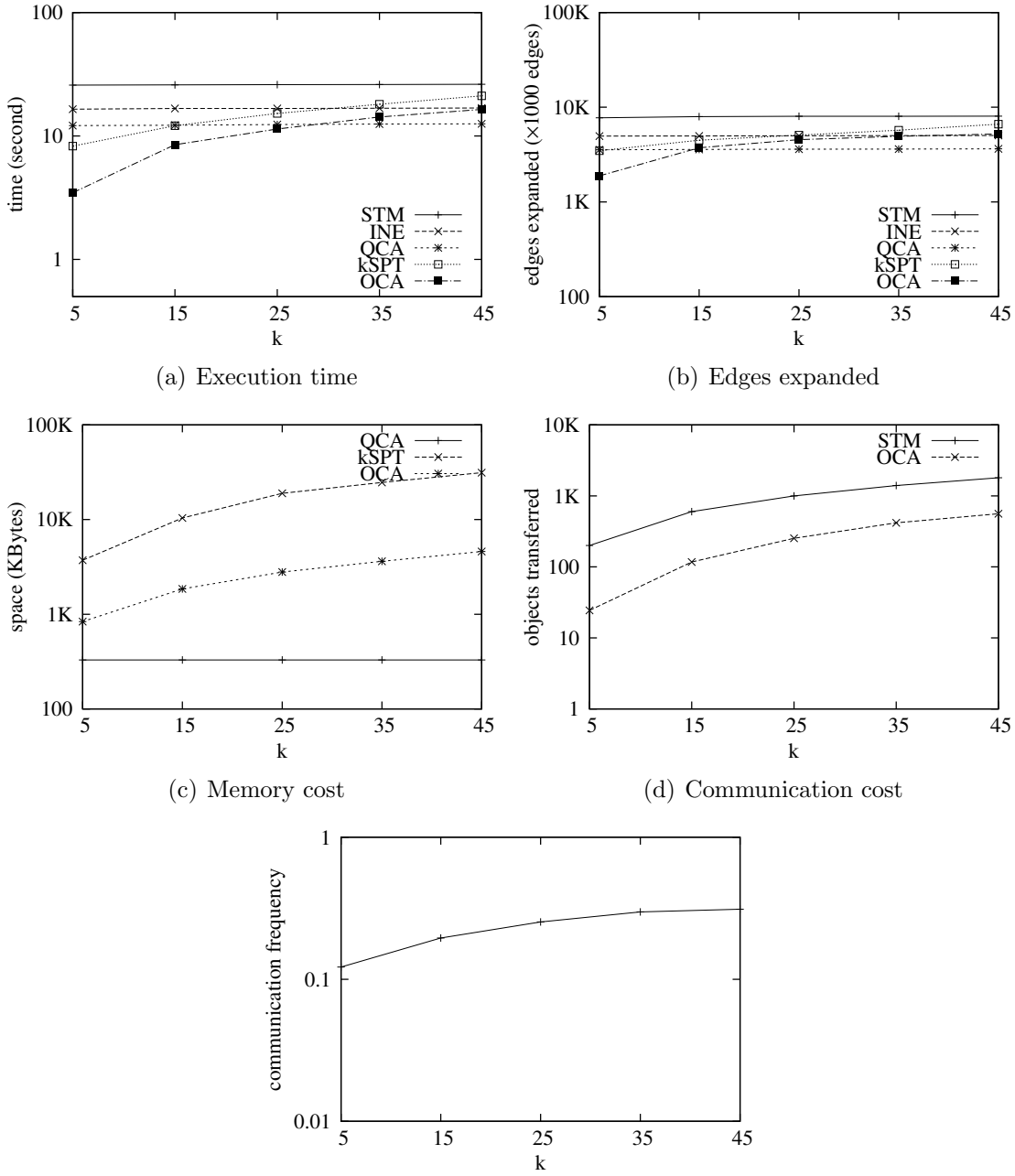
corresponds to the situation when the query is used as a snapshot query. As can be seen in Fig. 3.9(a) and Fig. 3.9(b), when used as snapshot query, STM, INE and QCA behave better than kSPT and OCA. However, as $l$ increases, execution time and edges expanded of these three methods consistently increase. INE has a better performance than STM, because it does not have to reevaluate the query and retraverse the network at each timestamp by using safe segment and Lemma 3.1, respectively. QCA behaves better than INE as QCA can avoid some repetitive traversing using expansion tree, which can be seen in Fig. 3.9(b). In spite of this, it is still outperformed by OCA. This is because QCA has to reevaluate the query results of each unvisited node encountered by the query point. Although it can avoid repetitive traversing of some network edges using expansion tree, QCA still has to traverse the edges which belong to the invalid part of the tree. For kSPT, both execution time and edges expanded remain unchanged, because kSPT computes the $k$ results for all nodes on the network. For OCA, changes in $l$ do not produce a noticeable effect on both cost measures, because when computing the top-$k$ results of the initial query node, OCA also obtains results or partial results of its surrounding nodes. Thus, the incremental cost of computing subsequent nodes can be negligible. OCA behaves much better than kSPT, because OCA adopts an incremental version of kSPT and halts when top-$k$ results of the requested nodes are found. The experimental results show that both QCA and OCA perform much better than STM, while OCA is the best method when used for longer query trajectories. For the memory cost, Fig. 3.9(c) shows that OCA consumes more space than QCA. This is because QCA only needs to preserve the node information for the expansion tree, while OCA has to preserve not only the node information but also the label information for each node. In addition, OCA also has to preserve the relevant objects. Note that compared with the order-$k$ shortest path tree preserved by OCA, the relevant objects consume rather low memory. On the other hand, OCA consumes much less space than kSPT, since OCA adopts an incremental version of kSPT. Actually, the memory consumed by kSPT is an upper bound for OCA. However, as can be seen, the incremental method is very effective and OCA consumes only a small part of the memory. For the communication cost and communication frequency, we can see that with the help of safe segment, all four methods have a very low cost compared with STM. The communication frequency is insensitive to $l$.

**Effect of $n$.** In this experiment, we evaluate the effect of $n$, the number of query keywords, as shown in Fig. 3.10. As can be seen in Fig. 3.10(a) and Fig. 3.10(b), there is a slight increase of execution time and edges expanded when $n$ increases. This is because the larger the number of keywords in the query, the larger the number of

objects that may be relevant for the query. For STM, INE and QCA, more objects need to be examined during the expansion phase of the snapshot query algorithm. While for kSPT and OCA, more relevant objects have to be taken into account and loaded into the object heap at the beginning of the algorithm and more edges have to be expanded before terminating the algorithm. QCA and OCA still perform much better than STM, while OCA remains to be the best method. Fig. 3.10(c) shows that the memory cost increases slightly with the number of query keywords. Fig. 3.10(d) and Fig. 3.10(e) show that the communication cost and communication frequency increases slightly with the number of query keywords. This is because larger query keyword sets increase the possibility of top-$k$ objects to be replaced by other candidate objects, which yields shorter safe segments.

**Effect of $k$.** In this experiment, we evaluate the effect of $k$, the number of requested results. Fig. 3.11(a) and Fig. 3.11(b) show that $k$ has no noticeable effect on the execution time or edges expanded of STM, INE and QCA. This is because when running the snapshot query algorithm, the algorithms also considers some relevant objects that are not in the final results. However, they may fall in the results when $k$ increases. Therefore, although $k$ increases, these methods consider similar number of relevant objects and do not induce much additional processing cost. On the other hand, kSPT and OCA are more sensitive to the parameter $k$. This is because $k$ determines the number of labels for each node. For kSPT, more edges need to be expanded to complete $k$ labels of each node. For OCA, the larger the number of results, the later OCA terminates when requested nodes are complete, which means more edges need to be expanded and longer execution time. These can also explain the results shown in Fig. 3.11(c). As $k$ increases, the memory cost for kSPT and OCA increase quickly, while the memory cost for QCA has no noticeable increase. In this case, QCA scales better than OCA when $k$ increases. Fig. 3.11(d) and Fig. 3.11(e) show that the communication cost and communication frequency increase when $k$ increases. This is because more objects need to be considered due to a larger $k$, which yields shorter safe segments. In addition, for the communication cost, a larger $k$ means the server needs to send more objects to the client.

**Effect of $\alpha$.** In this experiment, we evaluate the effect of the query preference parameter $\alpha$, as shown in Fig. 3.12. A small value of $\alpha$ gives more preference to the textual description of the objects, while a large value of $\alpha$ gives more preference to the network proximity. As can be seen, kSPT and OCA are not sensitive to $\alpha$. This is because $\alpha$ has little influence on the node labeling process of the requested nodes. All the other three methods perform slightly better for larger values of $\alpha$. This is




(a) Execution time

(b) Edges expanded

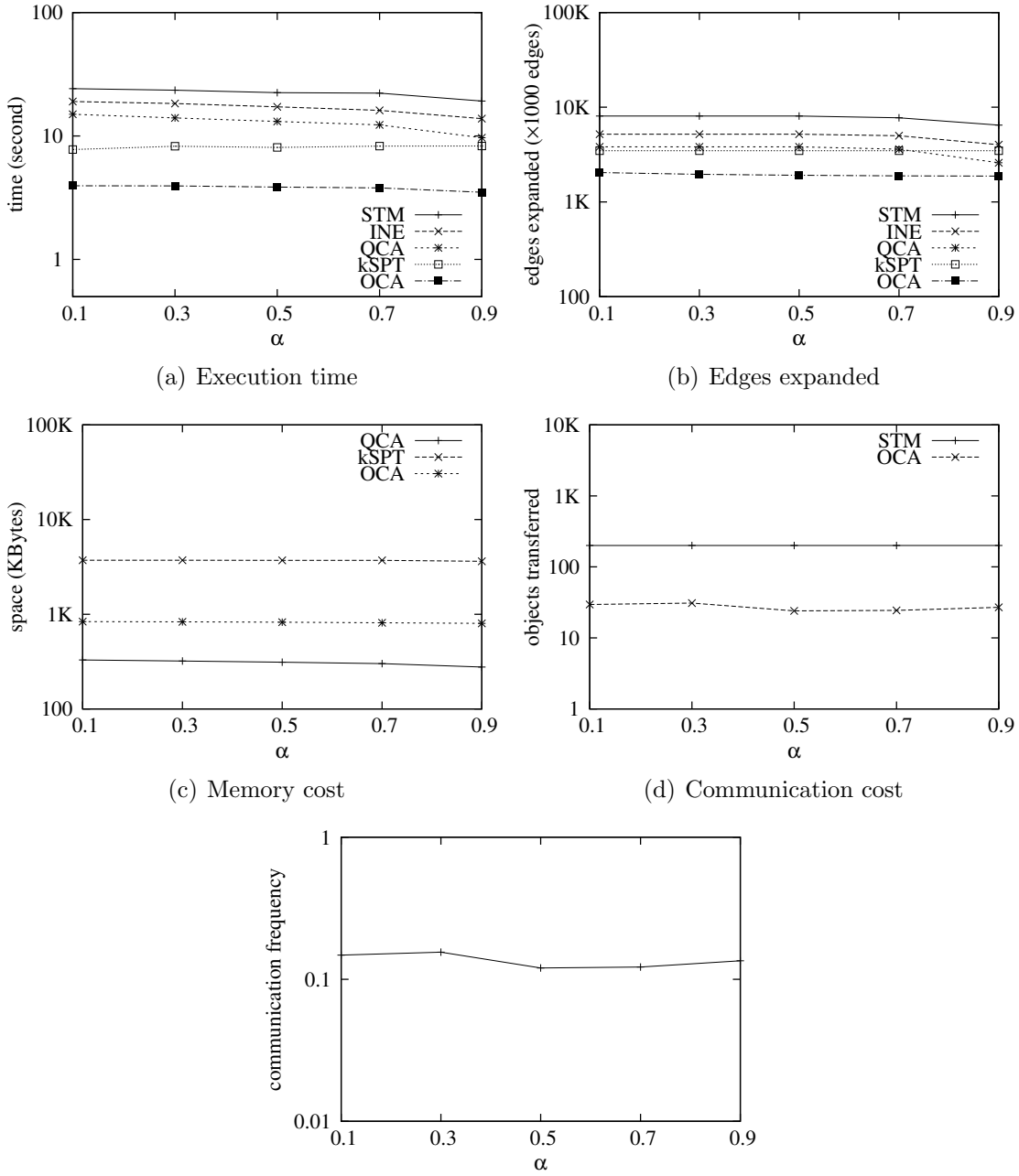
(c) Memory cost

(d) Communication cost

(e) Communication frequency

Figure 3.11: Effect of the number of results ($k$).

reasonable as the objects near the query location have a lower score when $\alpha$ increases, and thus, STM, INE and QCA can process fewer edges or terminates the algorithm earlier. Fig. 3.12(c) shows that the memory cost behaves similarly as the execution time. As can be seen in Fig. 3.12(d) and Fig. 3.12(e), the communication cost and communication frequency are insensitive to $\alpha$.




(a) Execution time

(b) Edges expanded

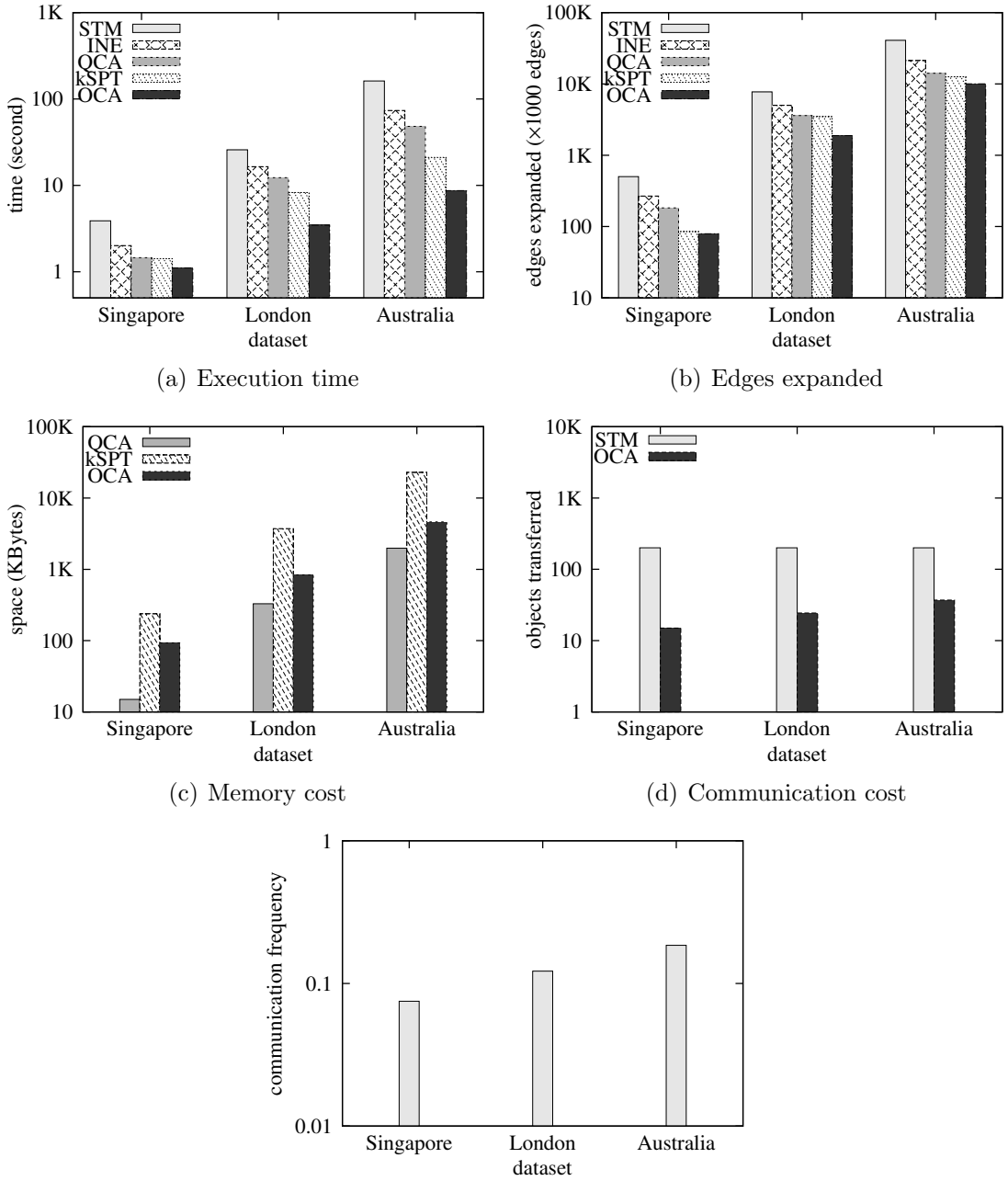
(c) Memory cost

(d) Communication cost

(e) Communication frequency

Figure 3.12: Effect of the preference parameter ($\alpha$).

**Effect of different datasets.** In this experiment, we study the total running time and the number of edges expanded for three real road network datasets of different sizes, as shown in Fig. 3.13(a) and Fig. 3.13(b). As can be seen, OCA scales well when the dataset size increases. The OCA method is more than one order of magnitude better than baseline methods STM and INE in terms of execution time for




(a) Execution time

(b) Edges expanded

(c) Memory cost

(d) Communication cost

(e) Communication frequency

Figure 3.13: Performance on different datasets.

the Australia dataset. Fig. 3.13(c) shows that QCA consumes the least space among the three methods, while OCA consumes much less space than kSPT. We also test the communication cost and communication frequency for these three datasets, as shown in Fig. 3.13(d) and Fig. 3.13(e). All four methods have a low communication cost and communication frequency compared with STM due to the use of safe segments.

### 3.5.3 Discussion

We have tested the effect of different parameters about MkSK queries. QCA behaves better than STM and INE because of using Lemma 3.1 with safe segment and expansion tree, while OCA behaves better than kSPT because of incremental order-$k$ shortest path tree construction. OCA scales better than QCA as the monitoring length $l$ increases, while QCA scales better than OCA as the number of results $k$ increases. Thus, this provides an insight on which method should be used in the realistic monitoring applications. If the number of results $k$ required by the client is large, it is better to use QCA to answer the query. Otherwise, it is better to use OCA. Overall, OCA behaves better than QCA for MkSK queries with a realistic parameter setting. One disadvantage of OCA is that it consumes more memory than QCA, which makes it be able to support less clients simultaneously than QCA. However, this disadvantage can be alleviated by the fact that the order-$k$ shortest path tree is valid for all the queries with the same query keywords, since OCA starts traversing the network from the relevant objects rather than the query location. Thus, for the queries with the same keywords which are submitted to the server during the same period, the server only needs to build one order-$k$ shortest path tree. In this way, both the execution time and the memory cost can be reduced dramatically, and the server can support more clients simultaneously. In addition, the server can cache the order-$k$ shortest path tree built for the frequent keywords, which can be used for the subsequent queries with these frequent keywords. QCA also has a potential advantage that other optimized snapshot query algorithms for top-$k$ spatial keyword queries on road networks (such as the overlay method in [62]) could be employed to further improve the performance, which is orthogonal to our work. Moreover, with the help of safe segment, QCA and OCA can have a rather low communication frequency, which is very important in the realistic monitoring applications.

## 3.6 Summary

In this chapter, we investigate the problem of continuous top-$k$ spatial keyword (MkSK) queries on road networks, and propose two efficient methods for query processing. QCA monitors the top-$k$ results of the moving point by examining the intersections the query encounters. It uses expansion tree to avoid repetitive traversing of some network edges. OCA incrementally retrieves the top-$k$ results according to textual relevance first and computes all or partial top-$k$ results of a subset of nodes on the network. We compare the proposed methods with three baseline methods.

Experimental results confirm the superiority of our two methods and reveal their relative advantages.

# Chapter 4

# Efficient Moving Spatial Queries Against Dynamic Event Streams

## 4.1 Introduction

In the previous chapter, we have discussed the efficient processing of moving spatial keyword queries on road networks. There, we have moving queries but the POIs are static. In this chapter, we extend the traditional moving spatial queries to search for not only static POIs but also dynamic events in order to provide users more rich information they are interested.

The prevalence of social networks and mobile devices has facilitated the real-time dissemination of local events such as sales, shows and exhibitions. To avoid missing interesting events in the neighborhood, various location-aware pub/sub systems have been proposed. These fall into two categories: they either focused on how to handle incoming event streams efficiently by assuming users' locations are static [6, 50, 13, 24]; or they attempted to process continuous moving subscriptions against a static event dataset [84, 35, 5, 75, 37, 32]. None of them can really support subscriptions from mobile users moving all the time against spatial events that are continuously published by local businesses.

In this chapter, we propose a new location-aware pub/sub system, Elaps, that continuously monitors moving users subscribing to dynamic event streams from social media and E-commerce applications. Users are notified instantly when there is a matching event nearby. Unlike existing pub/sub systems, Elaps uses boolean expressions, which are more expressive than keywords, to model user intent. This means users can subscribe to structured, semi-structured and unstructured data.

Fig. 4.1 illustrates a working scenario of such a system. Here, the subscribers with mobile devices are the moving objects, and a subscription is represented in the form of

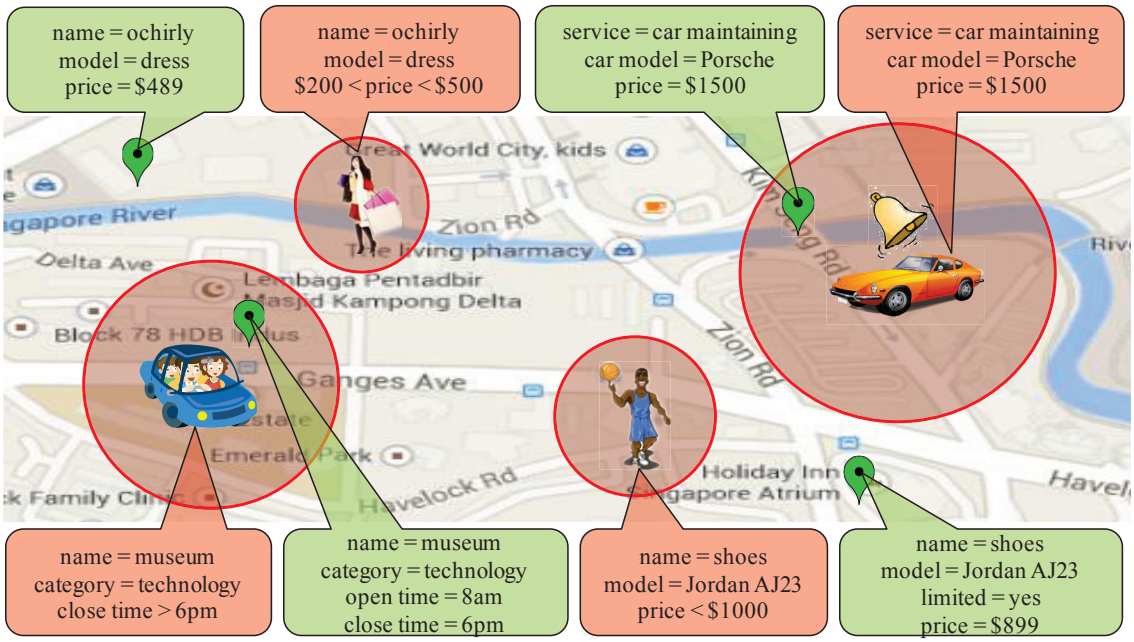



Figure 4.1: A working scenario of Elaps.

a boolean expression. For example, if a user is interested in Jordan basketball shoes, he can use a boolean expression to model the interest: (name=shoes $\wedge$ model=Jordan AJ23 $\wedge$ price $<$ $1000). Note that pub/sub systems based on keyword subscription cannot support numeric attribute matching such as price $<$ $1000. To specify the locational matching constraint, a subscriber can set a notification radius so that events lying inside the circle are considered as candidates. For example, the circles in Fig. 4.1 represent the notification regions of different users. When a user moves, the notification region moves along. On the publisher side, an event is published at a location. If a shoe shop has a sale, then the location of the shop is the event location. Our system continuously monitors the moving subscribers and notify them once there is a matching event in their circles.

To meet the above desiderata, our system is designed to tackle the following challenges:

The first challenge is *how to effectively process continuous moving subscriptions against dynamic event streams*. Safe region has been widely used to reduce communication cost for continuous moving query processing [84, 35, 5, 75, 37, 32]. The intuition behind the notion of safe region is that if there are no matching events nearby, the users are safe to disconnect from the server and do not need to periodically update their locations. However, we observed that the safe region techniques used in [84, 35, 5, 75, 37, 32] fail to work effectively when dynamic events are con-

sidered. This is because newly arrived events can trigger new communication needed to update the safe regions, besides the communication incurred by location updates. Moreover, these two types of communication have conflicting requirements on the size of safe regions. Therefore, there is a need to reconsider how best to exploit safe regions. In order to handle the dynamic event streams, we propose the impact region and exploit it as well as the safe region to monitor the newly arriving events. The impact region serves as a filtering mechanism so that only newly published events located within the impact region has a probability to change the safe region. Furthermore, we propose a novel cost model to capture the tradeoff between safe region size and the communication cost in a more precise way, and use it as a guide to find the optimal safe region for the moving subscribers against the dynamic event streams. Based on this cost model, we propose two safe region construction methods iGM and idGM that construct the safe region in an incremental manner. These two methods expand the safe region staring from the current location of the subscriber and stop when a termination condition based on the cost model is satisfied.

The second challenge is *how to design a system scalable to the fast growth of the number of subscriptions and events*, as the number of mobile users worldwide and the volume of internet information published have been growing exponentially. Two types of matching are considered here: *event matching* and *subscription matching*. Given an event, *event matching* returns the set of subscriptions matching this event. Given a subscription, *subscription matching* returns the set of events matching this subscription. *Event matching* has been well studied and we can simply adopt existing techniques [26, 71, 64]. However, there is a lack of efficient solutions to *subscription matching*.

It in turn brings the third challenge to the table, i.e. *can we support a more expressive subscription beyond the keyword subscription*. Thereby, we adopt boolean expression as our subscription semantic, and to bridge the gap in the second challenge, we propose a novel index named BEQ-Tree which uses a grid-based hierarchical structure and integrates the spatial information into the boolean expression information seamlessly to support spatial subscription matching efficiently.

More specifically, our technical contributions are summarized as follows:

- We optimize the design and processing of safe regions in several ways. First, given a safe region, we derive its *impact region*. The impact region is a novel concept used to identify if its corresponding safe region is affected by newly arrived events. Second, we propose a cost-based approach to determine the optimal safe region size to keep the communication overhead low. Our cost

model considers the communication cost incurred by location updates as well as that incurred by event arrival. Third, based on the cost model, we design two new schemes, iGM and idGM, to incrementally construct safe regions.

- We propose a new index named BEQ-Tree which integrates Quadtree [29] with boolean expressions seamlessly to support spatial boolean expression matching and safe region construction efficiently.

Moreover, we conduct comprehensive experiments using real datasets to evaluate the system performance. We use geo-tweets from Twitter and venues from Foursquare to simulate publishers and boolean expressions generated from AOL search log to represent users intentions. We test user movement in both synthetic trajectories and real taxi trajectories. The results show that our proposed iGM and idGM can reduce the communication overhead by 10 times. Also, our proposed index handles spatial boolean expression matching significantly faster than the competing methods.

In this chapter, we study the efficient processing of continuous moving range queries against dynamic event streams. Given a set of continuous arriving events, for a moving subscriber with a boolean expression subscription associated with a notification region, the subscriber is notified once there is a matching event located within his notification region. We aim for a solution that (i) optimizes the client/server communication cost and (ii) guarantees that the matching events are disseminated to subscribers in real-time.

The rest of this chapter is organized as follows. We first propose how to handle continuous moving queries against dynamic event streams in Section 4.2. We further introduce how to handle spatial boolean expression matching in Section 4.3. Based on the techniques proposed in Section 4.2 and Section 4.3, we present the system framework of Elaps in Section 4.4. Finally, we evaluate our system performance in Section 4.5 and conclude this chapter in Section 4.6.

## 4.2 Moving Queries over Dynamic Event Streams

Compared to existing location-based pub/sub systems, Elaps is the first to consider continuous moving queries against dynamic event streams from publishers. The main challenge is to reduce communication overhead because users have to periodically report their current locations to the server to guarantee that no matching events in their neighborhood are missed. Existing pub/sub systems that can handle moving users mainly use safe region techniques to reduce communication cost. The intuition

| Symbol | Description |
|---|---|
| $s$ | a moving subscriber |
| $e$ | a spatial event |
| $\mathcal{O}$ | notification region specified by a subscriber |
| $r$ | notification radius |
| $\mathcal{R}$ | safe region of a subscriber |
| $\mathcal{I}$ | impact region of a subscriber |
| $f$ | event arrival rate |
| $v_s$ | user moving speed |
| $n$ | total number of events in the system |
| $n_e$ | total number of matching events in $\mathcal{I}$ |
| $\mathtt{d}(s, \mathcal{R})$ | the minimum distance from $s$ to the boundary of $\mathcal{R}$ |
| $c$ | a grid cell in iGM and idGM |
| $G$ | a cell partition in BEQ-Tree |
| $\sigma$ | a reference point in $G$ |
| $y$ | the spatial attribute for distance indexing |
| $W$ | counter array in each cell partition in BEQ-Tree |

Table 4.1: Summary of Notations.

behind the notion of safe region is that if there are no matching events nearby, the users are safe to disconnect from the server and do not need to periodically update their locations. In this way, the communication cost can be significantly reduced and there would be no matching events missed. In this section, we first propose how to apply these techniques into our pub/sub application. Then, we explain why the safe regions constructed in these systems fail to work well when dynamic event streams are considered. Finally, we propose our solutions based on a concept named *impact region* as well as a new cost model for safe region construction.

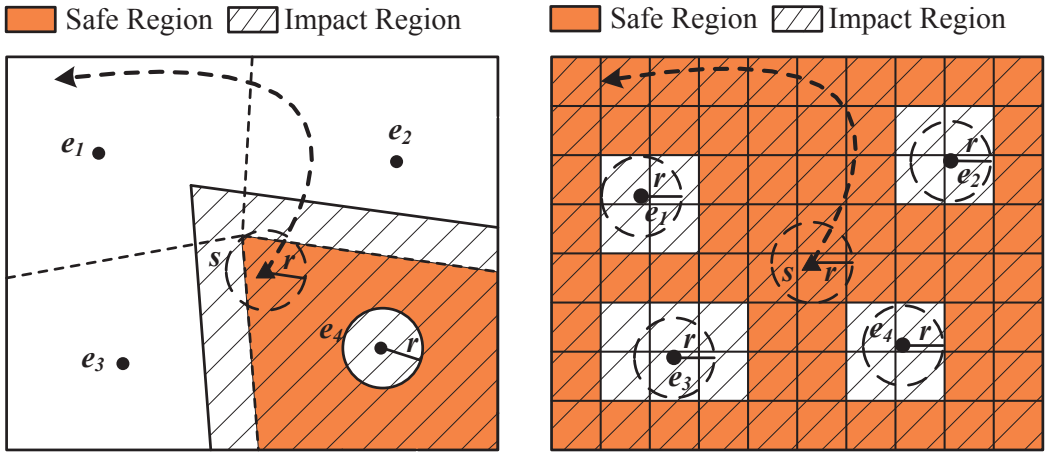
### 4.2.1 Safe Region against Static Event Datasets

Safe region has been widely used to reduce communication cost for continuous spatial query processing [84, 35, 5]. In these work, a common assumption is that the continuous query is issued against a static dataset and the goal is to determine an area in which there is no matching events or the matching events remain the same. Since the publisher dataset is static, there would be no new event matching in the safe region. For our application, we define a region is safe if its minimum distance to any matching event is larger than the user's notification radius, denoted by $r$[1].

[1]A notation table consisting of symbols and their meanings is provided in Table 4.1

**Definition 4.1 (Safe Region)** *The safe region $\mathcal{R}$ for a subscriber $s$ is a region such that $s \in \mathcal{R}$ and for any matching event $e$, we have $d(p, e) > r$ for any $p \in \mathcal{R}$.*

In the following, we examine existing safe region techniques and show how to apply them for continuous proximity detection between subscribers and their matching events.




(a) VM (b) GM

Figure 4.2: Applying existing methods for safe region construction.

**Voronoi-based Method (VM).** Voronoi diagram is often used in processing continuous $k$NN queries in continuous spatial query applications [84, 35]. Since the publisher dataset is static, the space is partitioned into Voronoi cells based on the locations of publishers. Each Voronoi cell indicates a region dominated by a publisher such that as long as a subscriber moves in that cell, the publisher is assured to be the nearest neighbor. To apply Voronoi diagram in our application, we first find all the matching events outside of the notification region [2] and construct Voronoi cells based on the matching events. Each cell contains one matching event. The safe region is the Voronoi cell containing the user, excluding the circle centered at the matching event with notification radius.

Figure 4.2(a) shows an example of safe region based on Voronoi diagram. There are four matching events that partition the space into four Voronoi cells. Since the subscriber is located in $e_4$, the safe region is the Voronoi cell for $e_4$ excluding the circle and marked in the shaded area. As long as the user moves in the safe region,

[2] If a matching event appears in the notification region, we notify the subscriber immediately about the matching. Then, the event will not be considered again for this user in the future.

we can guarantee that there is no matching event nearby and the user only need to check his distance with $e_4$ and can temporarily disconnect from the server.

**Grid-based Method (GM).** In [5], safe region based on grid cells was proposed to handle spatial alarm applications. By partitioning the space into cells, a safe region is represented by a set of cells whose distance to any matching events is larger than the notification radius. An example of grid-based safe region is shown in Figure 4.2(b). The safe region for the subscriber contains the whole space except the cells close to the matching events. Compared to the Voronoi diagram method, the grid based safe region is easier to construct and contains a larger region than VM.

When the event dataset is static, GM generates a larger safe region and achieves better performance in terms of communication I/O. However, when we consider dynamic event streams, the results can be totally different. In this case, a larger safe region does not necessarily mean a better solution. When a new matching event arrives close or inside the safe region, the region may not be "safe" any more. Then the whole safe region has to be re-constructed, leading to additional communication I/O. This observation motivates us to first develop a new concept named impact region to identify whether a safe region is "safe" or not when a new event arrives, and then propose a novel cost model based on the safe region and impact region to guide the construction of the safe region .

### 4.2.2 Impact Region

We first define the *impact region* for a certain safe region $\mathcal{R}$, and use it to determine whether $\mathcal{R}$ will be affected by the arrival of a new event. Intuitively, if a new matching event arrives in the impact region, the safe region $\mathcal{R}$ is not "safe" any more and needs to be updated. Otherwise, the safe region remains the same. With the help of the impact region, we do not need to update the safe region each time a new matching event arrives. Thus, the impact region can help reduce the communication cost incurred by event arrival. The concept of impact region will also be needed in our proposed cost model to estimate the expected time before the next matching event affects its corresponding safe region.

**Definition 4.2 (impact region)** *Given a safe region $\mathcal{R}$, the impact region $\mathcal{I}$ for $\mathcal{R}$ is defined as*

$$\{p \mid p \in U \ \text{ and } \exists\, p' \in \mathcal{R}\,(d(p,p') < r)\}$$

*where $U$ is the whole space and $r$ is the notification radius.*

Based on the definition, we know that if a point is located outside the impact region, its minimum distance to safe region $\mathcal{R}$ must be larger than $r$. Hence, we can consider impact region as an expansion of the safe region by the length of notification radius. Note that the impact region is uniquely determined by the safe region and should always be used together with the safe region. In the following, we briefly introduce how to construct impact regions from the safe regions in Figure 4.2.

**Example 4.1** *In VM, the impact region is constructed by expanding the Voroni cell of the nearest event with distance $r$. As shown in Figure 4.2, the impact region covers the notification region of the subscriber after the expansion. We can guarantee that when a new event arrives outside the impact region, its minimum distance to the impact region is larger than the notification radius. This new event will not fall inside the notification region as long as the user moves within the safe region. Hence, we only need to update the safe regions whose impact region contains this new event.*

*In GM, the impact region is constructed by expanding the safe region with $r$ to see which cells are contained in or intersected with the expanded region, which is the whole space in this example. When a new matching event arrives in the impact region, we need users to report their location to guarantee that the current safe region is "safe" and no matching notification is missed. Since the impact region contains the whole space, new communication overhead is incurred each time when a new matching event arrives.*

Up till now, we have introduced the three types of regions maintained for each moving user: notification region $\mathcal{O}$, safe region $\mathcal{R}$ and impact region $\mathcal{I}$. The notification region is set by the subscribers and move with them. It is a circle centered at the subscriber's current location with radius $r$. For each subscriber, the system constructs a safe region and an impact region for him. The safe region is sent to the subscriber to monitor the location update and the impact region is stored at the server side to monitor the event arrival. Their relationships are summarized as follows:

**Lemma 4.1** *$\mathcal{O}$ is contained in $\mathcal{I}$, denoted by $\mathcal{O} \subseteq \mathcal{I}$.*

**Proof 4.1** *Suppose we can find a point $p \in \mathcal{O}$ but $p \notin \mathcal{I}$. Since $p \notin \mathcal{I}$, based on the definition of the impact region, for any point $p'$ in the safe region $\mathcal{R}$, we have $d(p, p') > r$. Since the safe region is required to cover the user's current location $s$, i.e, $s \in \mathcal{R}$, we have $d(p, s) > r$. However, since $p$ is located in the notification region, for any points $p \in \mathcal{O}$, we have $d(p, s) < r$. This leads to a contradiction.*

**Lemma 4.2** *$\mathcal{R}$ is contained in $\mathcal{I}$, denoted by $\mathcal{R} \subseteq \mathcal{I}$.*

**Proof 4.2** *Based on the definition of impact region, for any point $p \in \mathcal{R}$, if we can find a point $p' \in \mathcal{R}$ such that $d(p, p') < r$, we know $p$ is also a point in the impact region. Let $p = p'$ and we finish the proof.*

Note that the safe region $\mathcal{R}$ does not necessarily contain the notification region $\mathcal{O}$. In fact, when a user increases his notification radius $r$, the safe region shrinks such that the minimum distance of the new safe region to any matching event is guaranteed to be larger than $r$. In addition, we can prove that the area of impact region grows when the safe region expands.

**Lemma 4.3** *Given $\mathcal{I}_1$ derived from $\mathcal{R}_1$ and $\mathcal{I}_2$ derived from $\mathcal{R}_2$, if $\mathcal{R}_1 \subseteq \mathcal{R}_2$, we have $\mathcal{I}_1 \subseteq \mathcal{I}_2$.*

**Proof 4.3** *Suppose we can find a point $p \in \mathcal{I}_1$ but $p \notin \mathcal{I}_2$. Since $p \notin \mathcal{I}_2$, based on the definition of the impact region, for any point $p' \in \mathcal{R}_2$, we have $d(p, p') > r$. Since $\mathcal{R}_1 \subseteq \mathcal{R}_2$, this conclusion also applied in the sub-region, i.e., for any point $p' \in \mathcal{R}_1$, we have $d(p, p') > r$. This leads to a contradiction with $p \in \mathcal{I}_1$. Hence, if $p \in \mathcal{I}_1$, we have $p \in \mathcal{I}_2$. a new matching event $e$ with $p$ as its location will influence $\mathcal{R}_1$ because $e$ is located within $\mathcal{I}_1$. Since $\mathcal{R}_1 \subseteq \mathcal{R}_2$, $e$ will also influence $\mathcal{R}_2$. However, since $e$ is located outside $\mathcal{I}_2$, this contradicts with the definition of impact region that any matching event located outside the impact region will not influence the safe region. Hence, if $p \in \mathcal{I}_1$, we have $p \in \mathcal{I}_2$.*

In the following, we show that there is no matching event in the impact region.

**Lemma 4.4** *Suppose $\mathcal{I}$ is an impact region constructed for subscriber $s$, for any event $e$ matching $s$, $e$ is located outside $\mathcal{I}$.*

**Proof 4.4** *If there is a matching event $e \in \mathcal{I}$, based on the definition of the impact region, we can find a point $p$ in the safe region such that $d(p, e) < r$. Then, by following the definition of safe region, since $p \in \mathcal{R}$, we have $d(p, e) > r$. This leads to a contradiction. Hence, $e$ must be located outside $\mathcal{I}$.*

From Lemma 4.4, we can ensure that if a matching event expires, it is located outside the impact region and has no effect on the current safe region.

### 4.2.3 Cost Model for Safe Region Construction

Since we consider continuous query processing against dynamic event streams, we first identify all the possible cases in which new communication can be triggered for an existing subscriber.

1. The subscriber moves out of his current safe region. He needs to report his new precise location to the server. At the server side, a new safe region is calculated and sent back to the user.
2. A new matching event arrives in the system. Whether a communication is triggered depends on the location of the new event. If the event is located outside the impact region, the safe region is not affected based on the definition. Otherwise, the safe region has to be updated. This triggers a new communication. First, the server notifies the subscriber to update the location. Then, the precise location is reported by the client. When the server receives the accurate location, it calculates the distance from the event to the subscriber. If the distance is smaller than the user's notification radius, a matching notification is sent to the user. Otherwise, the server needs to calculate and sends a new safe region to the client. In the meanwhile, the impact region is updated, but stored at the server side.
3. An existing matching event is expired and removed from the system. Based on Lemma 4.4, the matching event is located outside the impact region. We can guarantee that the current safe region is still "safe". As long as the subscriber is in the current safe region, he can disconnect from the server and no matching event will be missed.

Therefore, there are two types of communication incurred when handling a continuous query over dynamic event streams: I) The subscriber moves out of the safe region. II) A new matching event arrives in the impact region. These two types of communication have conflicting requirements on the size of safe region. The first type prefers larger safe region so that it takes longer time for the subscriber to move out of the safe region. However, the second type prefers smaller safe region. This is because a smaller safe region results in a smaller impact region according to Lemma 4.3 and it becomes less likely for a matching event to occur in the impact region. Existing safe region techniques do not work well for dynamic event streams because they neglect the second type of communication. Therefore, we propose a new cost model for safe region construction taking into account all these two types of communication.

The goal of our cost model is to minimize the communication overhead which is measured by the number of the two types of communication I/O for a subscriber $s$.

Hence, we construct a safe region $\mathcal{R}$ for $s$ such that the expected elapsed time before the next communication I/O is maximized. The expected elapsed time is denoted by $f_{obj}(\mathcal{R}, \mathcal{I})$ and used as the objective function to maximize. Let $t_s(\mathcal{R})$ denote the expected time to move out of the current safe region $\mathcal{R}$ and $t_i(\mathcal{I})$ denote the expected time before the next matching event occurs in the impact region $\mathcal{I}$ constructed from $\mathcal{R}$. Since there are only two circumstances in which communication I/O is triggered, we have

$$f_{obj}(\mathcal{R}, \mathcal{I}) = \min(t_s(\mathcal{R}), t_i(\mathcal{I})) \tag{4.1}$$

Next, we define $b_m$ to measure the tradeoff between the two types of communication.

$$b_m(\mathcal{R}, \mathcal{I}) = \frac{t_s(\mathcal{R})}{t_i(\mathcal{I})} \tag{4.2}$$

We can prove that $b_m(\mathcal{R}, \mathcal{I})$ has a positive correlation with the area of $\mathcal{R}$:

**Lemma 4.5** *Given two safe regions $\mathcal{R}$ and $\mathcal{R}'$ with $\mathcal{R} \subseteq \mathcal{R}'$, we have $b_m(\mathcal{R}, \mathcal{I}) \leq b_m(\mathcal{R}', \mathcal{I}')$.*

**Proof 4.5** *Since $\mathcal{R} \subseteq \mathcal{R}'$, we know that $t_s(\mathcal{R}) \leq t_s(\mathcal{R}')$ and $t_i(\mathcal{I}) \geq t_i(\mathcal{I}')$ based on Lemma 4.3 and the definition of $t_s$ and $t_i$. Hence, $b_m(\mathcal{R}, \mathcal{I}) = \frac{t_s(\mathcal{R})}{t_i(\mathcal{I})} \leq \frac{t_s(\mathcal{R}')}{t_i(\mathcal{I}')} = b_m(\mathcal{R}', \mathcal{I}')$.*

If $b_m(\mathcal{R}, \mathcal{I}) \leq 1$, we have $t_s(\mathcal{R}) \leq t_i(\mathcal{I})$ and $f_{obj} = t_s(\mathcal{R})$. In this case, we need to maximize $t_s(\mathcal{R})$ and we prefer a larger safe region for $\mathcal{R}$. If $b_m(\mathcal{R}, \mathcal{I}) > 1$, we have $f_{obj} = t_i(\mathcal{I})$ and we prefer a smaller safe region for $\mathcal{R}$. The relationship between $f_{obj}(\mathcal{R}, \mathcal{I})$ and $b_m(\mathcal{R}, \mathcal{I})$ are stated in the following two lemmas:

**Lemma 4.6** *Given two safe regions $\mathcal{R}$ and $\mathcal{R}'$ with $\mathcal{R} \subseteq \mathcal{R}'$, suppose $b_m(\mathcal{R}', \mathcal{I}') \leq 1$, we have $f_{obj}(\mathcal{R}, \mathcal{I}) \leq f_{obj}(\mathcal{R}', \mathcal{I}')$.*

**Proof 4.6** *Based on Lemma 4.5, $b_m(\mathcal{R}, \mathcal{I}) \leq b_m(\mathcal{R}', \mathcal{I}') \leq 1$ because $\mathcal{R} \subseteq \mathcal{R}'$. For $\mathcal{R}$, since $b_m(\mathcal{R}, \mathcal{I}) \leq 1$, we have $t_s(\mathcal{R}) \leq t_i(\mathcal{I})$ and thus $f_{obj}(\mathcal{R}, \mathcal{I}) = t_s(\mathcal{R})$. Similarly, $f_{obj}(\mathcal{R}', \mathcal{I}') = t_s(\mathcal{R}')$. Since $\mathcal{R}$ is contained in $\mathcal{R}'$, we have $t_s(\mathcal{R}) \leq t_s(\mathcal{R}')$. Therefore, $f_{obj}(\mathcal{R}, \mathcal{I}) = t_s(\mathcal{R}) \leq t_s(\mathcal{R}') = f_{obj}(\mathcal{R}', \mathcal{I}')$.*

**Lemma 4.7** *Given two safe regions $\mathcal{R}$ and $\mathcal{R}'$ with $\mathcal{R} \subseteq \mathcal{R}'$, suppose $b_m(\mathcal{R}, \mathcal{I}) \geq 1$, we have $f_{obj}(\mathcal{R}, \mathcal{I}) \geq f_{obj}(\mathcal{R}', \mathcal{I}')$.*

**Proof 4.7** *Based on Lemma 4.5,* $b_m(\mathcal{R}', \mathcal{I}') \geq b_m(\mathcal{R}, \mathcal{I}) \geq 1$ *because* $\mathcal{R} \subseteq \mathcal{R}'$*. For* $\mathcal{R}$*, since* $b_m(\mathcal{R}, \mathcal{I}) \geq 1$*, we have* $t_s(\mathcal{R}) \geq t_i(\mathcal{I})$ *and thus* $f_{obj}(\mathcal{R}, \mathcal{I}) = t_i(\mathcal{I})$*. Similarly,* $f_{obj}(\mathcal{R}', \mathcal{I}') = t_i(\mathcal{I}')$*. Since* $\mathcal{R}'$ *is contained in* $\mathcal{R}$*, we have* $t_i(\mathcal{I}) \geq t_i(\mathcal{I}')$*. Therefore,* $f_{obj}(\mathcal{R}, \mathcal{I}) = t_i(\mathcal{I}) \geq t_i(\mathcal{I}') = f_{obj}(\mathcal{R}', \mathcal{I}')$*.*

Our safe region construction method relies on the above lemmas. The idea is to start from the user's current location and incrementally expand the area towards an "optimal" safe region (i.e., $f_{obj}$ is maximized). Initially, $\mathcal{R}$ contains only a point. Thus, $t_s$ can be seen as 0 and $b_m(\mathcal{R}, \mathcal{I}) \leq 1$. When we expand $\mathcal{R}$, $b_m(\mathcal{R}, \mathcal{I})$ and $f_{obj}(\mathcal{R}, \mathcal{I})$ also increase (Lemma 4.5 and 4.6). In this case, it encourages us to expand the safe region until any further expansion would cause $b_m(\mathcal{R}, \mathcal{I}) \geq 1$ or the region not "safe". This is because $f_{obj}(\mathcal{R}, \mathcal{I})$ decreases when $b_m(\mathcal{R}, \mathcal{I}) \geq 1$ if we continue to expand $\mathcal{R}$ based on Lemma 4.7.

Note that there are many possible ways to expand a safe region, resulting in lots of candidate safe regions. For each of these candidate safe regions, we have $b_m \leq 1$ and thus $f_{obj} = t_s$. Hence, the "optimal" safe region is the candidate safe region with the largest $t_s$. Based on this observation, we should expand the safe region in such a way that the corresponding $t_s$ can be maximized.

### 4.2.4 Incremental Grid-based Method

We are now ready to present our incremental method, named iGM (incremental Grid-based Method), towards optimal safe region construction. Since we need a flexible way to represent safe region in arbitrary shape, we partition the space into $N \times N$ cells and a safe region is represented by the set of cells that it covers. Our algorithm starts from the cell containing the user's current location and iteratively expands it to cover nearby cells. In each expansion, a "good" cell based on certain criteria is added to the current safe region. When a cell is added into the current safe region, we also need to expand the corresponding impact region to calculate $b_m(\mathcal{R}, \mathcal{I})$. The algorithm terminates when any further expansion would cause $b_m(\mathcal{R}, \mathcal{I}) > 1$ or there is no more valid adjacent cells to expand. In the following, we introduce the estimation of $b_m(\mathcal{R}, \mathcal{I})$ for a candidate safe region $\mathcal{R}$ as well as the selection criteria for the next cell to expand.

Assume that subscribers are moving with linear motion functions. The expected time $t_s(\mathcal{R})$ to move out of $\mathcal{R}$ can be calculated by

$$t_s(\mathcal{R}) = \frac{\mathtt{d}(s, \mathcal{R})}{v_s} \tag{4.3}$$

where $\mathtt{d}(s, \mathcal{R})$ is the minimum distance from a subscriber's location to the boundary of a candidate safe region $\mathcal{R}$ and $v_s$ is the current moving speed.

The expected time $t_i(\mathcal{I})$ before the next matching event occurs in the impact region $\mathcal{I}$ for $\mathcal{R}$, is estimated by

$$t_i(\mathcal{I}) = \frac{t_e}{p(e, \mathcal{I})} \tag{4.4}$$

where $t_e$ is the average time interval between two new events arriving at the system and $p(e, \mathcal{I})$ is the probability for a new event to match subscriber $s$ and occur in his impact region $\mathcal{I}$.

We can estimate $t_e$ from the average arrival speed in the event streams. We denote the average arriving rate of the new events by $f$ and we have $t_e = \frac{1}{f}$. The probability $p(e, \mathcal{I})$ can also be estimated from the distributions of the existing events in the system. Let $n_e$ be the number of matching events located in the impact region $\mathcal{I}$. $n_e$ can be estimated by $n_e = \sum_{c \in \mathcal{I}} n_c$ where $n_c$ is the number of existing matching events located within cell $c$. Let $n$ be the total number of events in the system. We can estimate $p(e, \mathcal{I}) = \frac{n_e}{n}$. Then $t_i$ can be calculated as follows.

$$t_i(\mathcal{I}) = \frac{n}{f \cdot n_e} \tag{4.5}$$

Based on Equation 4.3 and Equation 4.5, we have

$$b_m(\mathcal{R}, \mathcal{I}) = \frac{t_s(\mathcal{R})}{t_i(\mathcal{I})} = \frac{f \cdot n_e \cdot \mathtt{d}(s, \mathcal{R})}{n \cdot v_s} \tag{4.6}$$

Note that among the parameters, $v_s$, $f$ and $n$ are system statistics and are independent of the safe region and impact region. $\mathtt{d}(s, \mathcal{R})$ and $n_e$ are dependent on the areas of the candidate safe region and impact region, respectively.

Next, we introduce our expansion criteria. Suppose the current safe region is $\mathcal{R}$. We mark all the cells covered by or intersected with $\mathcal{R}$ as visited. The candidate cells to expand include all the adjacent cells of $\mathcal{R}$ that are unvisited. Our expansion criteria is to pick the cell $c$ with the minimum distance to the subscriber. In other words, we expand the area circularly. This is because we expand the safe region until any expansion would cause $b_m(\mathcal{R}, \mathcal{I}) \geq 1$. In this process, we have $b_m(\mathcal{R}, \mathcal{I})$ always smaller than 1 and our goal is to maximize $t_s$. Since the moving pattern of the subscriber is not available, we expand the cells in every direction uniformly so as to improve the performance in the worst case.

Our iGM algorithm for safe region construction is shown in Algorithm 4.1. The input is a subscription $s$ with notification radius $r$ and speed $v_s$. We first initialize

**Algorithm 4.1:** ConstructSafeRegion

**input**: Subscription $s$
**output**: Safe region $\mathcal{R}$ and impact region $\mathcal{I}$

```
1  b_m ← 0
2  n_e ← 0
3  d(s, R) ← 0
4  H ← ∅
5  c ← the cell that contains s
6  HEntry h ← tuple(c, d(s, c))
7  insert h into H, mark c as visited
8  while H ≠ ∅ do
9      HEntry (c', d(s, c')) ← H.pop()
10     if β[c'] ≠ false then
11         d(s, R) ← min{H.top().dist, min{d(s, c'')}}
12         I_c ← getImpactCells(c')
13         foreach c_n ∈ I_c do
14             n_e = n_e + φ[c_n]
15         b_m = (f · n_e · d(s,R)) / (n · v_s)
16         if b_m ≤ 1 then
17             R ← R ∪ c'
18             I ← I ∪ I_c
19             foreach non-visited adjacent unit cell c'' do
20                 HEntry h ← tuple (c'', d(s, c''))
21                 insert h into H, mark c'' as visited
22 return R and I
```

the parameters $b_m$, $n_e$ and $\mathtt{d}(s, \mathcal{R})$ (lines 1-3) and build a min-heap $\mathcal{H}$ whose root stores the cell candidate with the minimum distance to $s$. The heap is initialized to contain the cell $c$ where the subscriber is located (lines 4-7). After the initialization steps, we expand the safe region in a breadth-first fashion (lines 8-21). The candidate cell $c'$ with the minimum distance to $s$ is popped in each iteration (line 9). A boolean array $\mathcal{B}$ is used to indicate whether a grid cell is *safe* or not. A cell in the array is set to *safe* if its distance to the nearest matching event is larger than the notification radius of $s$. If $c'$ is not *safe*, we simply ignore it and continue to examine the next cell in the heap. Otherwise, we calculate $\mathtt{d}(s, \mathcal{R})$ and $n_e$ for the enlarged safe region $R \cup c'$ and impact region $\mathcal{I} \cup \mathcal{I}_c$ to evaluate $b_m$ (lines 10-18), where $\mathcal{I}_c$ is the set of candidate cells that need to be added to $\mathcal{I}$ if $c'$ is added to $\mathcal{R}$. The calculation of $\mathtt{d}(s, \mathcal{R})$ follows the equation:

$$\mathtt{d}(s, \mathcal{R} \cup c') = min\{\mathcal{H}.top().dist, min\{d(s, c'')\}\} \tag{4.7}$$

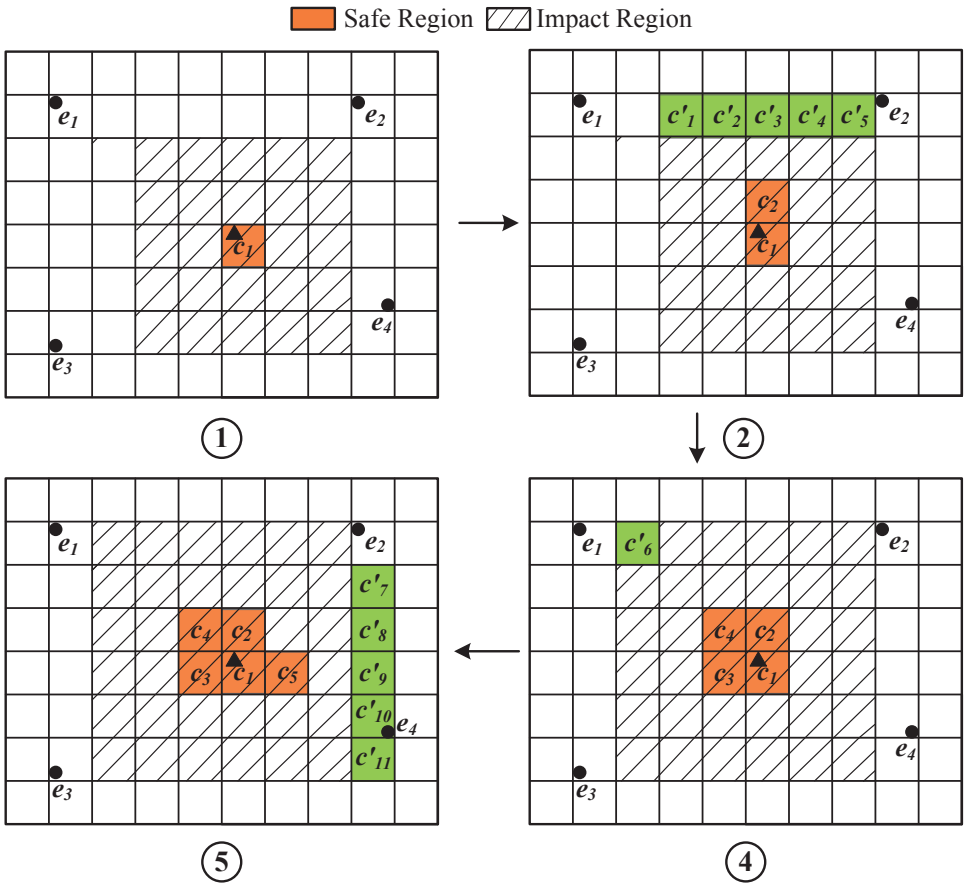



Figure 4.3: Safe region and impact region expansion.

where $c''$ is an unvisited adjacent cell of $c'$(line 11). The procedure *getImpactCells* is used to get $\mathcal{I}_c$(line 12). Then we update $n_e(\mathcal{I} \cup \mathcal{I}_c)$ (lines 13-14) and calculate $b_m(\mathcal{R} \cup c', \mathcal{I} \cup \mathcal{I}_c)$ (line 15). If $b_m(\mathcal{R} \cup c', \mathcal{I} \cup \mathcal{I}_c) \leq 1$, we enlarge $\mathcal{R}$ to contain cell $c'$ and $\mathcal{I}$ to contain the cells in $\mathcal{I}_c$ (lines 16-18). We then insert the adjacent cells of $c'$ which are not visited to $\mathcal{H}$ (lines 19-21). The whole algorithm terminates when $\mathcal{H}$ becomes empty and we return the safe region and impact region. Note that the two types of regions are constructed together in the expansion.

When a cell is added into the current safe region, we expand the impact region in the meanwhile. A naive solution is to scan all the cells within the notification radius and add them into current impact region. However, this incurs many redundant operations because a candidate cell will be added into impact region multiple times. To avoid scanning so many cells in each expansion, we propose an incremental solution that is able to add only the unvisited cells into the impact region. We use an example to illustrate our idea.

**Example 4.2** *Illustrating examples of safe region expansion in multiple steps are*

*shown in Figure 4.3. In the first step, the safe region $\mathcal{R}$ is initialized to the cell containing $s$, i.e., $\mathcal{R} = c_1$ and the impact region is an enlarged area of $\mathcal{R}$ by notification radius $r$. In the second step, a neighboring cell $c_2$ with the minimum distance is selected. Since $c_2$ has a neighboring cell $c_1$ included in the safe region, we know that the cells within distance to $c_1$ has been added in the impact region. Thus, we can directly add cells $c'_1$, $c'_2$, $c'_3$, $c'_4$ and $c'_5$ into the impact region without scanning all the cells. In the third step, $c_3$ is selected into the safe region. The expansion of impact region is similar to the second step and we do not show this step in the figure. In the fourth step, the nearest unvisited cell $c_4$ is selected. This time, the cell has two adjacent cells included in the safe region. We only need to add one cell $c'_6$ to the impact region. As the expansion process continues, the safe region and impact region become larger, resulting in a larger $b_m$. Suppose $c_5$ is the last cell to cause $b_m \leq 1$ during the expansion, we can terminate the algorithm after we expand the safe region to include $c_5$ and update the impact region accordingly.*

Next, we propose a direction-aware version of iGM when the direction information of the moving objects is available.

### 4.2.5 Incremental Direction-aware GM

Since most smartphones are equipped with sensors for direction detection, we propose a direction-aware iGM, named idGM, that takes into account user moving direction to better maximize $t_s(\mathcal{R})$, the expected time to leave a safe region.

To make iGM direction-aware, we should take into account the direction information when expanding the safe region. In other words, the original expansion mechanism relies on the distance between $d(s, \mathcal{R} \cup c)$ and each time the cell with the minimum distance is selected. We extend the idea to propose a more general scoring function in determining the next cell to expand. The ranking function takes into account of the user moving direction as well as the cell distance and is defined as follows:

$$\tau(s, c) = \alpha \cdot \mathcal{A}(s, c) + (1 - \alpha) \cdot \mathcal{D}(s, c) \tag{4.8}$$

The direction preference $\mathcal{A}(s, c)$ is the cosine value of the angle $\theta$ between the moving direction $\vec{v_s}$ and the vector $\vec{sc}$ from $s$ to $c$.

$$\mathcal{A}(s, c) = cos\theta = \frac{\vec{v_s} \cdot \vec{sc}}{\|\vec{v_s}\|\|\vec{sc}\|} \tag{4.9}$$

The distance preference $\mathcal{D}(s, c)$ is the normalized distance from $c$ to $s$, which is defined as

$$\mathcal{D}(s, c) = \frac{d(s, c)}{d_{max}} \tag{4.10}$$

where $d_{max}$ is a normalization parameter which can be set to the maximum distance between any two points in the space.

Note that our goal is to maximize $t_s$ in the safe region construction until the condition $b_m \leq 1$ is not satisfied. If the user moving pattern is predictable and we have high confidence that the user will continue to move along the direction, we can set $\alpha$ to a value close to 1. Then, the cells within the user's moving direction have higher priority to be added to the safe region. As long as the direction does not change, the user can stay in the safe region for a long time. If the user moving pattern is not clear and there are many uncertainties, we can set $\alpha$ to be a small value and uniformly expand the safe region in all directions.

With the new scoring function $\tau$, we can modify Algorithm 4.1 to be direction-aware. The entry of $\mathcal{H}$ is modified as a tuple $(c, d(s, c), \tau)$ (line 7). And the entries in $\mathcal{H}$ are sorted in increasing order of $\tau$. Now the algorithm expands the safe region by taking both the distance preference and direction preference into consideration. The grid cell with the minimum $\tau$ will be accessed firstly. Compared to iGM, idGM constructs a direction-aware safe region which takes a longer period before the next communication occurs.

To reduce the bytes transferred in the communication between servers and subscribers, we use Bitmap as a more compact representation of safe region. We allocate a Bitmap whose length is the number of cells. If a cell is in the safe region, we set the corresponding element of the Bitmap to be 1. During the communication, we adopt the run-length encoding compression method (e.g., BBC [3] and WAH [76]) to further reduce the size of the Bitmap. In this way, we achieve a very compact representation of the safe region. When subscribers receive the message, they first decode it into the original Bitmap and can easily detect whether the cell they are located in is in the safe region or not.

## 4.3 Spatial BE-Matching

In Section 4.2, we have introduced how to optimize the communication cost between the server and the client. In this section, we present how to disseminate the matching events to the subscribers in real-time. We observe that existing pub/sub systems using boolean expression matching [26, 71, 64, 81] rarely pay attention to index construction

for the event stream. However, in the location-based service scenario, a subscriber wants to be notified when there is a matching event near him, even though this event has already been published before his subscription. This motivates us to build an index to cater for continuously arriving subscriptions' matching, namely BEQ-Tree (Boolean Expression Quad-Tree). In addition, we can utilize BEQ-Tree to improve the efficiency of construcing the safe region in iGM and idGM.

First of all, we describe the problem setting. In Elaps, a subscriber expresses his interest in the form of spatial subscription and is modeled as a moving object, while a publisher is associated with a geo-location and publishes spatial events.

**Spatial Subscription.** A spatial subscription extends a boolean expression with a notification region $\mathcal{O}$. In this work, we model a boolean expression as a conjunction of predicates. Each predicate is determined by three elements: an attribute $A$, an operator $f_{op}$ and an operand $\overline{o}$. It accepts an input value $x$ and the output is a boolean value indicating whether the operator constraint is satisfied or not: $P^{(A,f_{op},\bar{o})}(x) \rightarrow \{0,1\}$. Elaps can support relational operators $<, \leq, =, >, \geq, [\ ]$ [3] and set operators $\in, \notin$. As mentioned, the notification region $\mathcal{O}$ is a circle centered at user's current location with radius $r$. Formally, a spatial subscription $s$ is defined over $|s|$ predicates and an notification region $\mathcal{O}$:

$$s : P_1^{A,f_{op},\overline{o}}(x) \wedge P_2^{A,f_{op},\overline{o}}(x) \wedge ... \wedge P_{|s|}^{A,f_{op},\overline{o}}(x) \wedge \mathcal{O}$$

For example, a user interested in Samsung TQ can submit a subscription like *(brand=samsung* $\wedge$ *size>50)*$\wedge$ *r=1km* $\wedge$ *lat=1.28* $\wedge$ *lng=103.8)*. Note that the location is detected automatically. Hereafter, we use spatial subscription and subscription interchangeably.

**Spatial Event.** A spatial event $e$ contains $|e|$ tuples and a location $loc$: $e : (A_1 = \overline{o}_1) \wedge (A_2 = \overline{o}_2) \wedge ... \wedge (A_{|e|} = \overline{o_{|e|}}) \wedge loc$, where $A_i$ is the attribute and $\bar{o}_i$ is the associated value or operand. For example, a Samsung TQ promotion event can be represented by *(brand=samsung* $\wedge$ *size=55* $\wedge$ *3D=yes* $\wedge$ *lat=1.28* $\wedge$ *lng=103.8)*.

**Spatial Subscription Match.** The match between a spatial boolean expression $s$ and an event $e$ consists of two aspects: boolean expression match and spatial match, as defined below.

**Definition 4.3 (Boolean Expression Match)** *A boolean expression match is satisfied if for each predicate $P$ in $s$, $P$ is satisfied by a tuple $A_i = \overline{o}_i$ in $e$. We use $s \sim_b e$ to denote a boolean expression match and say $S$ be-matches $E$.*

**Definition 4.4 (Spatial Match)** *We say there is a spatial match between $s$ and $e$, denoted by $s \sim_s e$, if the location $loc$ of $e$ is inside the notification region $\mathcal{O}$ of $s$.*

[3] In this case, the operand $\overline{o}$ contains two values: $\overline{o}.l$ and $\overline{o}.r$

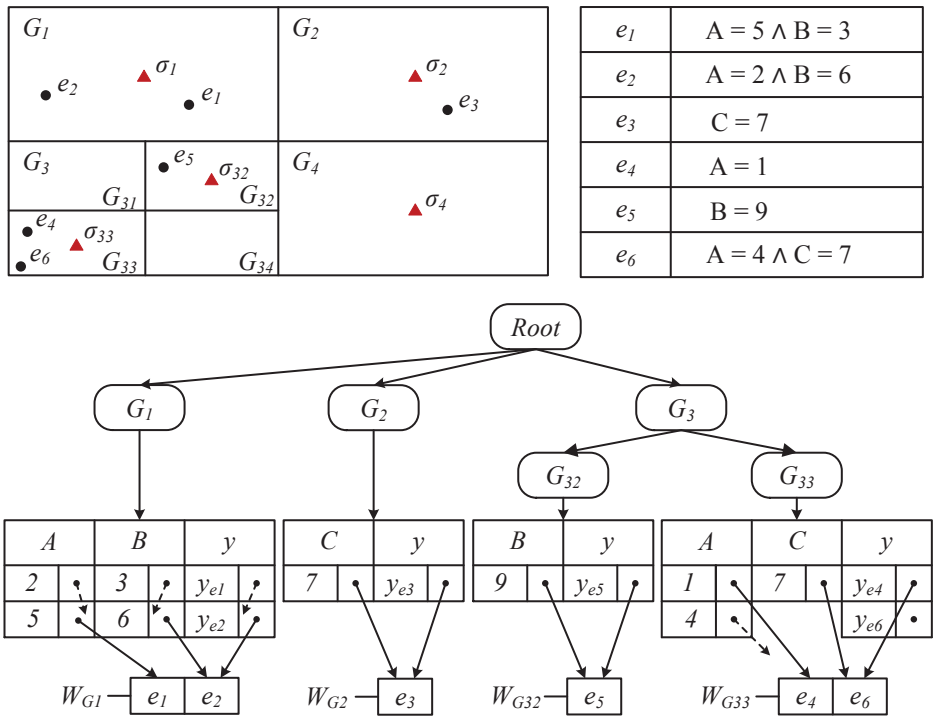



Figure 4.4: An example for BEQ-Tree.

**Definition 4.5 (Match)** *We say a subscription $s$ matches an event $e$, denoted by $s{\sim}e$, if $s{\sim}_b e$ and $s{\sim}_s e$.*

### 4.3.1 Spatial Event Index

BEQ-Tree is designed to be efficient in both query processing and event update. It adopts a two-layer partitioning mechanism, one for the spatial attribute and the other for the predicates in the boolean expression.

In the first layer, we partition the events based on the spatial attribute. Since Quadtree [29] can answer spatial range query quickly and support efficient update operations, we adopt it to partition the space such that each cell in the leaf level contains at most $E_{max}$ events, where $E_{max}$ is a moderately large number. In each cell, we select a reference point and adopt the idea of *iDistance* [38] to calculate and index the distance from each event to the reference point. The reference point is denoted as $\sigma$ and selected as the center of a tree cell in this work.

In the second layer, we further partition the events in each cell based on the attributes in the predicates. The predicates with the same attribute and appear in the same tree cell are organized in the same inverted list, denoted by $L_{\langle G_i,A\rangle}$ where $G_i$ is the tree cell and $A$ is the attribute. The location of an event $e$ in $G_i$ is converted to

a single dimensional value $y = dist(e, \sigma_i)$, where $dist(e, \sigma_i)$ represents the Euclidean distance between $e$ and $\sigma_i$. We then build an inverted list for attribute $y$ for each cell. Each inverted list is sorted by the operand value. For each cell, we also maintain a counter array for all the events in this cell.

**Example 4.3** *Fig. 4.4 shows an example for the BEQ-Tree, where we set $E_{max} = 2$ for a Quadtree cell. The space is first hierarchically partitioned into cells. Each cell $G_i$ is associated with a set of inverted lists to store predicates with the same attribute. All tuple lists are sorted in ascending order of their tuple values. There is an extended attribute $y$ to store the distance from the event to the reference point in a cell. Each predicate has a pointer to a counting array for the corresponding Quadtree cell, which will be used in subscription matching.*

**Memory Cost.** Let $E$ denote the set of spatial events published to Elaps, $\mathbb{E}$ denote the size of $E$, $|T|$ denote the total number of tuples in $E$ and $|t|$ denote the memory space cost by an entry in the list. The event index contains two components: the tuple lists and the counter arrays. Each tuple corresponds to a unique entry in the tuple lists. Thus, the tuple lists occupy $O(|T||t|)$ memory space. In addition, each event corresponds to a unique entry in the counter arrays. Since the size of an entry in the counter arrays is much smaller than the size of an entry in the tuple list, the total memory cost for the index is $O(|T||t|)$. As can be seen, our index takes linear space cost.

### 4.3.2 Index Maintenance

In our system, new events will be continually published by the publishers. Each event has a valid period and will expire after this period. Thus, BEQ-Tree should be efficient in terms of the maintenance cost.

We first consider how a new event is inserted into the BEQ-Tree index. Given an event $E$, we first find the cell with the lowest level that contains $E$. If the number of events within that cell is smaller than $max_{event}$, we append a new entry $e$ in the associated counter array $V_{G_i}$ and set its value to the location of $E$. For each non-spatial attribute, we insert its value and the key pointing to $e$ into the corresponding tuple list. For the spatial attribute, we convert it to the one-dimensional distance $y$ and insert $y$ to the spatial list. If the cell is full, we need to partition it into four child cells and insert the event into the corresponding child cell.

The delete operation is processed as follows. We first find the cell with the lowest level that contains $E$. Then, we traverse the inverted lists whose attribute is contained

**Algorithm 4.2:** BESpatialMatch(Subscription $s$, Cell partition $G$)

```
1  R_e ← ∅
2  for each predicate (A f_op ō) ∈ s do
3  |  if G does not contain A then
4  |  |  return R_e
5  W_G ← counter array associated with G
6  for each predicate (A f_op ō) ∈ s do
7  |  for each operator f_op ∈ {=, ≠, ≤, ≥, [ ]} do
8  |  |  determine the range R_a in L_⟨G,A⟩
9  |  |  for each matching entry t ∈ R_a do
10 |  |  |  ++W_G[t.e]
11 y ← dist(s, σ)
12 if s is located within G then
13 |  d_min ← y − r, d_max ← y + r
14 else
15 |  d_min ← y − r
16 |  determine d_max accordingly
17 for each t ∈ L_⟨G,y⟩ and t.ō ∈ [d_min, d_max] do
18 |  if W_G[t.e] == |s| then
19 |  |  if dist(s.l, t.l) ≤ r then
20 |  |  |  add the corresponding event into R_e
21 return R_e
```

in $E$ and delete the tuple. The deletion is fast because the list is sorted and we can use binary search to quickly identify the tuple to delete. If the cell becomes empty after deletion, we check whether its sibling nodes are also empty. If yes, we merge them to the parent node.

**Update Complexity.** Let $\mathbb{N}$ denote the maximum level of the BEQ-Tree, $|L|$ denote the maximum length of the tuple lists and $\mathbb{P}$ denote the maximum number of conjunctions in a subscription. The cost of Quadtree cell identification is $O(\mathbb{N})$. The insertion or deletion cost of a conjunction into a sorted list is $log(|L|)$. Thus, the total insertion or deletion complexity is $O(N) + O(\mathbb{P}log|L|) = O(\mathbb{P}log|L|)$, because usually $N \ll L$ in our BEQ-Tree. Note that with the help of the hierarchical structure in our index, $|L|$ will not be too large, making the index maintenance very efficient.

### 4.3.3 Subscription Matching

In the following, we show how to find the matching events given a subscription $s$ with an notification region $\mathcal{O}$. We first find all the leaf cells that intersect with $\mathcal{O}$ using

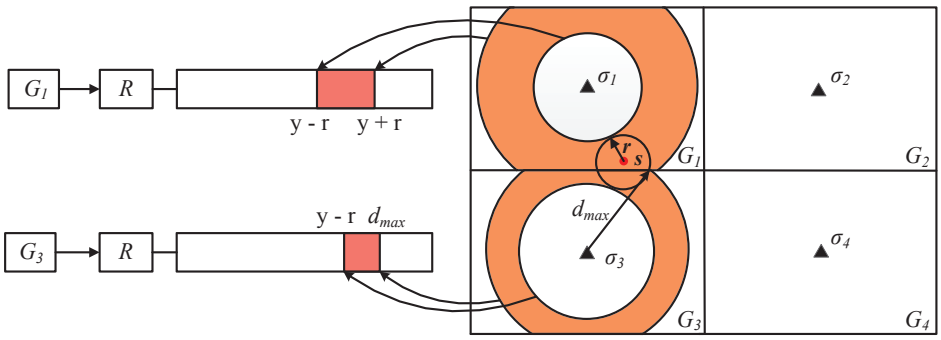



Figure 4.5: Spatial range match.

the Quadtree, and then examine the events in each candidate leaf cell $G$ by calling Algorithm 4.2.

In Algorithm 4.2, we first check whether a cell partition $G$ in the BEQ-Tree contains all the attributes that appear in $s$. If we find an attribute of $s$ not appearing in $G$, we can prune the search space and examine other cells (lines 2-4). Otherwise, we copy the counter array associated with $G$ and set the initial values of the entries to be 0 (line 5). Next, we introduce how to find the matching events by performing boolean expression match (lines 6-10) and spatial range match (lines 11-20).

The boolean expression match algorithm adopts the idea of the classic counting algorithm in [79, 26, 71]. Given a predicate $A\ f_{op}\ \bar{o}$, we use different accessing strategy for different operators in $s$ (lines 6-8). If $f_{op}$ is '=', we can check whether $\bar{o}$ appears in the sorted tuple list $L_{\langle G,A\rangle}$ using binary search. If $f_{op}$ is '$\neq$', all the values in the tuple list except $\bar{o}$ are visited. If $f_{op}$ is '$\leq$' ('$<$'), all the tuple entries whose value is no larger (smaller) than $\bar{o}$ match $A\ f_{op}\ \bar{o}$. The case is similar for '$\geq$' (or '$>$') and '[ ]'. For each list entry visited, we increase the corresponding counter value by 1 (lines 9-10). If the value increases to the size of $s$, the corresponding event be-matches $s$ (see Definition 4.3).

We use Fig. 4.5 to explain how to perform spatial range match. Similar to boolean expression match, we need to locate the interval in the spatial list within which the corresponding event may fit in the notification region $\mathcal{O}$. There are two cases to consider. First, $s$ is located within $G$ (lines 11-13). In this case, the interval is $[y-r, y+r]$, where $y$ is the one-dimensional distance value of $s$. Second, $s$ lies out of $G$ (lines 14-16). In this case, the lower bound of the interval is $y-r$. If the notification range contains a vertex of $G$, we have $d_{max} = \infty$, which means that we start from $d_{min}$ and traverse to the end of the spatial list. Otherwise, $d_{max} = max\{dist(\sigma, p_i)\}$, where $p_i$ is one of the intersection points between $G$ and $\mathcal{O}$. As shown in Fig. 4.5, the notification region crosses two grid cells and we need to conduct spatial range

search in these two partitions. The shaded areas indicate the search interval using the indexed distance. The interval for $G_1$ is $[y - r, y + r]$, because $s$ is located within $G_1$. The interval for $G_2$ is $[y - r, d_{max}]$, because $s$ is located outside $G_2$ and $\mathcal{O}$ does not contain the vertex. For each tuple in the interval, if its corresponding event be-matches $s$, we check whether the event is located within $\mathcal{O}$. If yes, we find a match (lines 17-20). Since subscribers tend to specify a small notification range, the spatial range match would be very efficient, because only a few entries in the spatial list are traversed.

By performing the boolean expression match and spatial match, we only need to traverse a small part of entries in a list, which can further improve the matching performance besides the spatial pruning of the first layer.

**BEQ-Tree used in iGM and idGM.** To construct the safe region in iGM and idGM, the set of be-matching events should be found first. A naive method is to find all the be-matching events in the whole space. However, one property of iGM and idGM is that these two algorithms usually do not expand to the whole space with the constraint of $b_m(\mathcal{R}, \mathcal{I}) \leq 1$. Based on this property, we use BEQ-Tree in an incremental manner to get the be-matching events on demand. To construct the safe region for a subscriber $s$, we start from the cell $c$ containing $s$ and search the BEQ-Tree to get the set of be-matching events in $c$. In the meantime, if the leaf Qudatree cells intersected with $c$ contains other cells around $c$, we also get the be-matching events in these cells. When iGM or idGM expands to a cell whose be-matching events have not been found, we check the surrounding Quadtree cells to get the be-matching events. In this way, we only need to find the set of be-matching events on demand and traverse only a part of the space.

## 4.4 System Framework

So far, we have introduced the techniques of safe region and impact region to reduce communication I/O and the BEQ-Tree to reduce the response time. In this section, we present the system framework in Fig. 4.6 as a whole picture to see how different function components are connected. In particular, we introduce how to process subscription arrival/expiration, event arrival/expiration and user location update.

**Subscription arrival/expiration.** In Elaps, users can submit new subscriptions and an existing subscription expires if the user is no longer interested in receiving matching events. Such kind of messages are handled by the *Subscription Processor*. When a new subscription arrives, we need to find if there exist any matching events

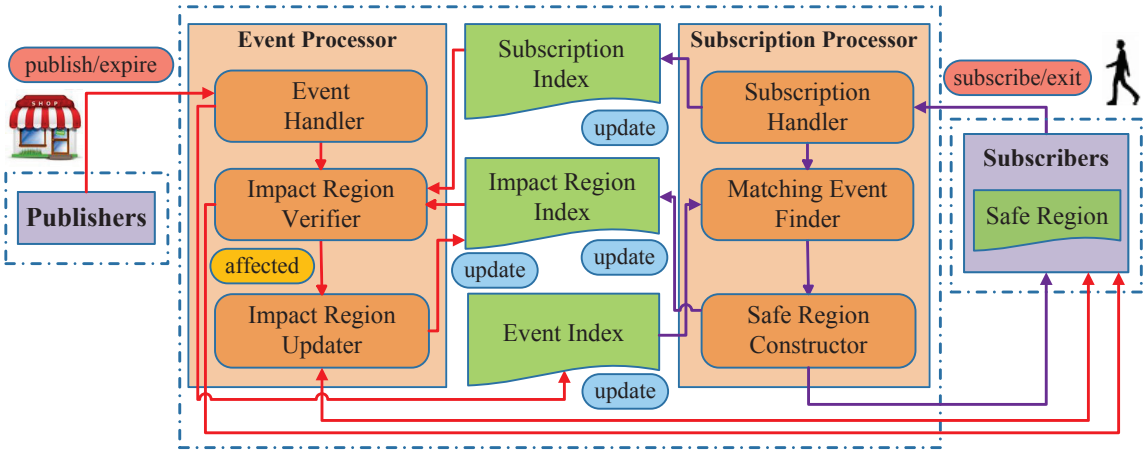



Figure 4.6: The workflow in Elaps framework.

in the event database. Since this is an extension of boolean expression matching with spatial constraints, we propose a new index named BEQ-Tree to solve the problem in Section 4.3. For each new subscriber, we also call Algorithm 4.1 in Section 4.2 to construct a new safe region and impact region for the user. The safe region is sent to the user and the impact region is maintained at the server side. The impact region is inserted into an inverted index with cell id as the key and the elements are impact regions covering the cell. When a subscription expires, we remove it from the subscription index and impact region index.

**Event arrival and expiration.** When a new event $e$ arrives, we need to identify from the subscriber database the users whose interests match the new event ($e$ is contained in the notification region) and those users whose safe regions are affected ($e$ is contained in the impact region). Since we use safe region to reduce the communication overhead, the subscriber's precise location is not maintained at the server side. We only know the user is currently in the safe region but the precise location is not available. Thus, we build separate indexes to handle boolean expression matching and spatial constraint verification. For the boolean expression matching, we can simply adopt existing subscription index such as OpIndex [81] and BE-Tree [64]. For the spatial attribute, we store the impact region for subscribers in the impact region index maintained as a hash table. When $e$ arrives, we find the matching users $U$ from the subscription index. For each subscriber $u \in U$, we check whether $e$ locates within his impact region. If yes, the server send $e$ to $u$. If the distance from $u$ to $e$ is smaller than the notification radius, the user is notified. Otherwise, we calculate a new safe region and impact region. The new safe region is sent to the subscriber and the impact region index is also updated.

**User location update.** Each time a subscriber moves out of the safe region, he reports the new location to the server. Once the server receives the new location, it calls Algorithm 4.1 to construct a new safe region and impact region for the user. Again, the safe region is sent to the subscriber and the inverted index for impact regions is updated.

## 4.5 Experimental Study

In this section, we present a comprehensive evaluation of the performance for Elaps. In particular, we are interested in evaluating (1) the communication overhead caused by continuous moving query processing against dynamic event streams and (2) the matching efficiency of our proposed BEQ-Tree. For safe region techniques, we compare our proposed iGM and idGM based on the new cost model against existing methods VM and GM. In all these methods, we calculate and store the impact regions at the server side for performance evaluation. For spatial boolean expression matching, we compare BEQ-Tree with three baseline algorithms: (i) Quad-tree, which first filters events outside the notification region using the spatial index Quad-tree [34] and then verifies the boolean expression matching; (ii) extension of $k$-index [71], which finds matching events for a given subscription. The be-matching events are further verified for spatial matching; (iii) OpIndex, which first filters the events not matching the boolean expressions using a variant of OpIndex [81] and then verifies the spatial matching. Note that all the indexes are memory resident and all the approaches produce the same and complete results. We implemented all the methods in C++ and conducted the experiments on a server with 48GB memory running Centos 5.6.

### 4.5.1 Experimental Setup

**Event and Subscription Datasets.** We use geo-tweets from Twitter and venues from Foursquare to simulate the event streams. In Twitter, each geo-tweet is considered as a spatial event. We treat each keyword as an attribute and the value is the frequency of the keyword in the tweet. This converts a geo-tweet into a list of attribute-value pairs. In the following experiments, we use 50 million geo-tweets as the event database and another 10 million to simulate a dynamic event stream. To generate subscriptions on the Twitter dataset, we adopt the same technique in [81] to convert a keyword query in AOL search log [4] into a boolean expression. For example, a query "SIGMOD Melbourne" can be transformed to $(\mathsf{SIGMOD}{=}1 \wedge \mathsf{Melbourne}{=}1)$

[4] `http://www.gregsadetsky.com/aol-data/`

to support equal operator or (SIGMOD∈[5,20] ∧ MelBourne∈[2,8]) to support interval operator. In Foursquare, each venue is considered as one spatial event. We extract structured information, i.e., attribute-value pairs, from the venues. Each venue has around 50 attributes and we harvest $1,832,418$ venues. Among them, we use 1.5 million venues as the event database and the remaining to simulate the event stream. To generate the subscriptions over the Foursquare venue stream, we let the subscriptions follow the same distribution as the venues. In other words, if an attribute is frequent in the venues, it is also frequent in the subscriptions. The operators and operands in the predicates are synthetically attached.

**User Trajectory Datasets.** We use both synthetic trajectories and real trajectories to simulate user moving patterns. For the synthetic trajectories, we generate $10,000$ trajectories using Brinkoff's generator [10]. The average travel period of each trajectory is 1000 timestamps. In our experiments, timestamp is used to capture the periodicity of location update. We set each timestamp to be 5 seconds in the following experiments, which means the GPS location of a user is sampled every 5 seconds. For the real trajectories, we use the GPS probing records of taxies in Singapore as the real trajectories [85]. We extract 10,000 trajectories from different taxies and each trajectory contains 1000 sequential points.

| Communication Overhead | |
|---|---|
| Event arrival rate $f$ $(/tm)$ | 10, 50, **100**, 500 |
| Moving speed $v_s$ $(m/tm)$ | 20, 40, **60**, 80, 100 |
| Notification radius $r$ $(km)$ | 1, **2**, 3, 4, 5 |
| Number of events $\mathbb{E}$ | 10M, **20M**, 30M, 40M, 50M |
| Matching Performance | |
| Number of events $\mathbb{E}$ | 10M, **20M**, 30M, 40M, 50M |
| Avg. sub size $\delta$ | 1, 2, 3, **4**, 5 |
| Notification radius $r$ $(km)$ | 1, **2**, 3, 4, 5 |

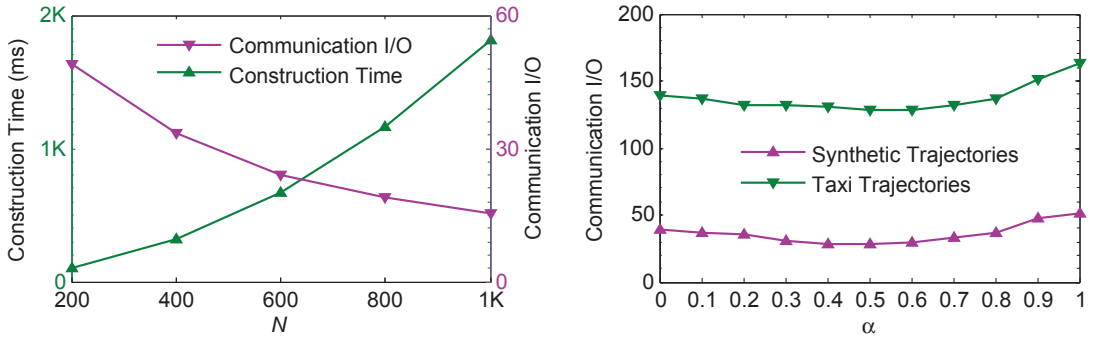
Table 4.2: Parameters evaluated in the experiments

**Compression Method.** Our proposed algorithm iGM and idGM use a set of cells to approximate the safe region. Thus, when the safe region of a user is updated, a list of cell ids are sent to the user. In our implementation, we assign each cell a value derived from the z-ordering of the cells [59]. Based on the z-order, the cells close to each other will be assigned similar ids. Then, we use the classic WHC [76] to compress the list of ids. Our experimental results show that the size of compressed ids is around $5\% - 10\%$ of the original size.

**Performance Metrics.** In the experiments, we report the following metrics: (i) *Average Communication I/O*, which is the average number of messages incurred for a subscriber moving along the synthetic or real trajectory; (ii) *Matching Time*, which is the average time cost to match a spatial subscription against a large event corpus. Note that the communication cost and matching time are our primary optimization goal, as elaborated in the problem statement. We also report the server side computation cost in safe region construction during the experiment period (i.e., 1000 seconds), which is a dominating operation in the server side.

**Evaluation Parameters.** Table 4.2 shows the main parameters and values used throughout the experiments (default values are in bold). To evaluate continuous moving query processing, we examine the scalability with respect to increasing event arrival rate $f$, user moving speed $v_s$, notification radius $r$ and event corpus size $\mathbb{E}$. To evaluate the matching performance of spatial boolean expressions, we examine increasing event corpus size $\mathbb{E}$, average subscription size $\delta$ and notification radius $r$. The system performance can be measured by the average communication I/O and event matching time for a subscriber. Note that we do not need to examine the performance w.r.t. increasing number of subscribers. This is because the subscribers will not affect each other in terms of communication overhead and event matching efficiency. In our system, the number of subscribers directly influences the scalability of the subscription index (see Fig. 4.6). However, in this work, we adopt an existing index (OpIndex [81]) which has been shown to be scalable even for a large number of subscribers. Moreover, the focus of this work is not on the subscription index. Thus, we did not look into this any further. Pease refer to [15] for details.

### 4.5.2 Parameter Tuning

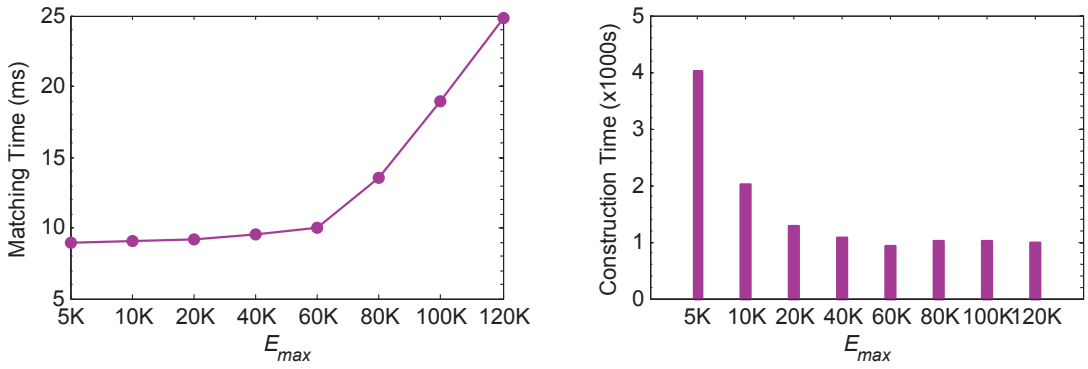



(a) Varying $N$ (b) Varying $\alpha$

Figure 4.7: Parameter tuning for iGM and idGM.

In continuous moving query processing, our proposed iGM and idGM use small cells to approximate the safe region and impact region. Suppose the whole space is split into $N \times N$ cells and we increase $N$ from 200 to 1000 to test how the performance varies when the cell size becomes smaller. As shown in Fig. 4.7(a), when $N$ increases, we can represent the cell in a more precise fashion and the optimal safe region is better approximated. Thus, it can reduce communication I/O. However, it requires more construction time since we need more iterations to expand the safe region until the termination condition is satisfied. For the following experiments, we set $N$ to be 600 as a tradeoff between communication I/O and CPU cost in safe region construction[5]

Another parameter in *idGM* is $\alpha$ which reflects the confidence level of user moving pattern. In the synthetic trajectories, we increase $\alpha$ from 0 to 1.0. If $\alpha = 0$, idGM degrades to iGM. We can see that direction is a factor that can improve system performance. The optimal performance occurs when $\alpha = 0.5$. If $\alpha$ is set very high, the performance is bad because the system assumes that the subscriber always moves along the original direction and is likely to construct a safe region along the direction. Then, it is easy for the subscriber to move out of the safe region when he changes the moving direction.




(a) Matching Time

(b) Index Construction Time

Figure 4.8: Parameter tuning for BEQ-Tree.

For BEQ-Tree, we tune the parameter $E_{max}$ which is the maximum number of events in a cell. Fig. 4.8(a) and Fig. 4.8(b) illustrate the matching time and construction time of BEQ-Tree when varying $E_{max}$. When $E_{max}$ increases, the cell becomes larger and the effect of spatial pruning degrades. Thus, it takes more matching time (Fig. 4.8(a)). However, a smaller $E_{max}$ results in more levels in the BEQ-Tree, which

[5]The parameter tuning in this section hold for all the data set and experiments.

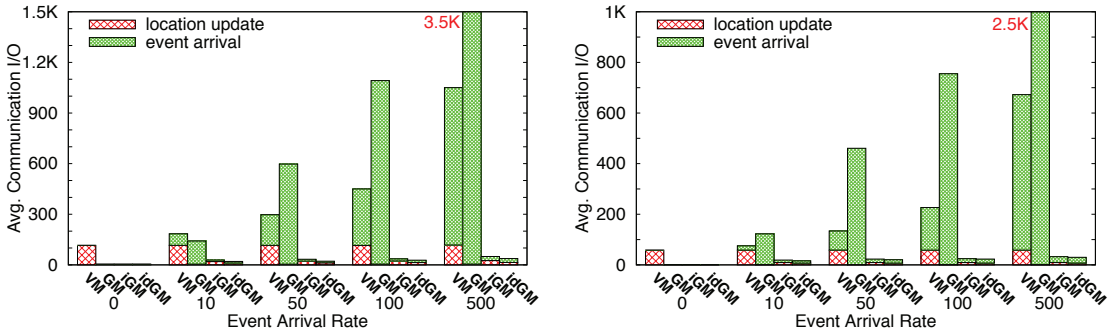
increases the update cost (Fig. 4.8(b)). Therefor, we set $E_{max}$ to be 60K as a good tradeoff between subscription matching time and event update cost.

### 4.5.3 Evaluation on Continuous Moving Query Processing

In this part, we evaluate the communication overhead caused by continuous moving query processing against dynamic event streams.

#### 4.5.3.1 Synthetic Trajectories

The first set of experiments on continuous moving query processing are conducted on the synthetic trajectories using the Twitter and Foursquare events.

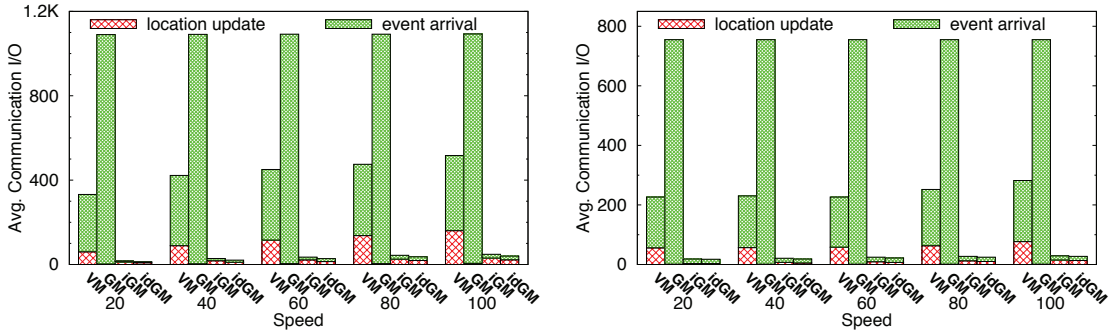



(a) Twitter (b) Foursquare

Figure 4.9: Effect of the event arrival rate.

**Effect of the event arrival rate $f$.** Fig. 4.9 presents the impact of the event arrival rate $f$ on the performance. We report the communication overhead incurred when users move out of the safe region (location update) and when a new matching event occurs in the impact region (event arrival), respectively. We have the following observations. (1) As $f$ increases, all the methods scale in varying degrees. Because there would be more matching events that fall within the impact region, which increases the cost incurred by event arrival. (2) in terms of the total communication cost, iGM and idGM can outperform the other baseline methods by one order of magnitude. Such performance gain increases when $f$ increases, especially on the cost incurred by event arrival. This is because iGM and idGM can adjust the size of the safe region dynamically according to several parameters of the system to balance the cost incurred by location update and the cost incurred by event arrival. (3) Regarding the cost incurred by location update alone, GM has the smallest one, because it constructs the largest safe region. VM performs much worse than the other methods, because it constructs a safe region around the nearest matching event regardless of

the user's location. idGM performs better than iGM, because idGM can construct a safe region with a larger $t_s$ which is the expected time before the subscriber leaves the safe region by considering the user's moving direction. (4) Regarding the cost incurred by event arrival alone, GM is the highest, because it constructs a rather large impact region even though $f$ is very high. Although VM can construct a smaller impact region compared to GM, the impact region of VM always contains some highly skewed area (around the nearest matching event). Therefore, the cost incurred by event arrival of VM is also high. When $f$ is larger, iGM and idGM would construct a smaller safe region and impact region based on the cost model. Thus, the cost incurred by event arrival can be controlled very well.

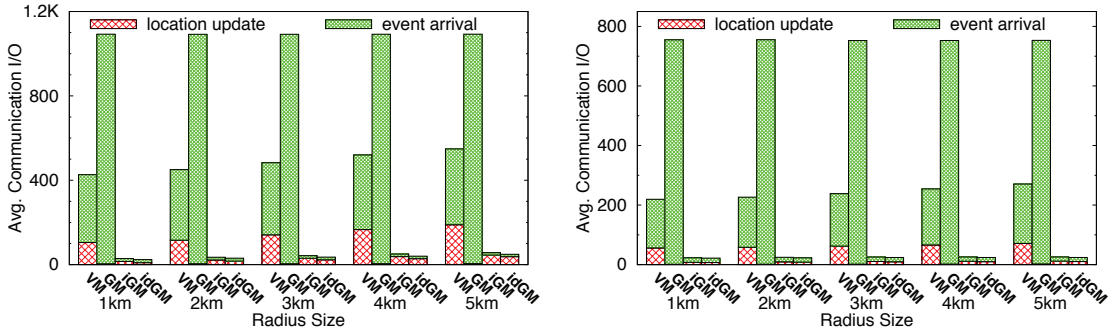



(a) Twitter (b) Foursquare

Figure 4.10: Effect of the moving speed.

**Effect of the speed $v_s$.** Fig. 4.10 depicts the effect of another key factor, the moving speed $v$, on the communication overhead. When $v$ increases, the subscriber would leave the safe region more frequently, resulting in an increased cost incurred by location update. Therefore, the total communication cost increases for all methods except GM. GM is not sensitive to $v$ for Foursquare and Twitter, because it constructs a rather large safe region. As shown, iGM and idGM still outperform the rest, since they can dynamically increase the size of the safe region as $v$ increases. We can also observe that the superiority of idGM compared with iGM is more significant with a larger $v$.

**Effect of the notification radius $r$.** Next, we evaluate the effect of the notification radius $r$. The results are shown in Fig. 4.11. A larger $r$ results in a smaller safe region, which increases the cost incurred by location update for all the methods. For VM, the cost incurred by event arrival increases. This is because we expand the safe region for VM by the length of $r$ and thus a larger $r$ results in a larger impact region. In contrast, iGM and idGM can adjust the size of the impact region dynamically




(a) Twitter (b) Foursquare

Figure 4.11: Effect of the notification radius.

even though $r$ increases. Thus, iGM and idGM show 10X better performance than the other baseline methods.

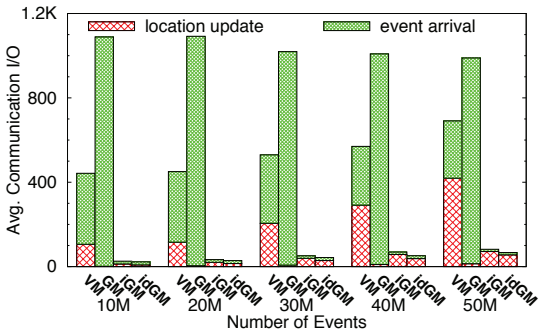
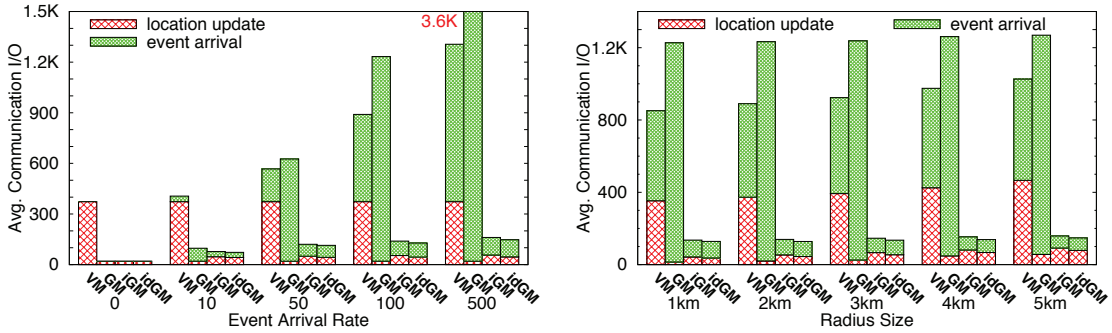



Figure 4.12: Effect of the number of events.

**Effect of the number of events $\mathbb{E}$.** Lastly, we study the communication overhead when increasing the size of the Twitter events from 10M to 50M. The results are shown in Fig. 4.12. More events (i.e., more matching events) in the system result in a smaller safe region and impact region. A smaller safe region leads to more cost incurred by location update, while a smaller impact region leads to less cost incurred by event arrival. For VM, the overall communication overhead increases because the increase in the cost incurred by location update surpasses that in the cost incurred by event arrival. On the other hand, for GM, the overall communication overhead decreases, because the performance of GM relies more on the impact region. Compared with VM and GM, our two methods can adjust the area of the safe region and impact region better when the system environment changes and thus scale very well when the event size increases.




(a) Event arrival rate

(b) Notification radius

Figure 4.13: Communication IO on the taxi trajectories.

#### 4.5.3.2 Taxi Trajectories

The second set of experiments are conducted on the real taxi trajectories in Singapore. Since the trajectories are only located within Singapore, we extract those geo-tweets located within Singapore from the Twitter dataset. In total, we extract 906,977 geo-tweets. We use 0.5 million of them as the event database and the remaining to simulate the event stream. We evaluate the communication overhead with respect to the event arrival rate $f$ and the notification radius $r$. The results are shown in Fig. 4.13(a) and Fig. 4.13(b). Compared with the synthetic trajectories where the speed is constant, the taxi trajectories contain all kinds of taxies with different moving status. Besides, the moving speed of a taxi is influenced by the road traffic greatly. Therefore, it is more difficult to predict the moving behavior for the taxi trajectories. However, as can be seen, our two methods can still achieve a much better performance compared with the other baseline methods. As compared to the counterpart GM, our iGM and idGM have a comparable performance in terms of the cost incurred by location update and reduce the cost incurred by event arrival significantly by more than 1 order of magnitude.

#### 4.5.3.3 Cost Model Evaluation

In this part, we study the optimality and robustness of our cost model.

**Optimality Evaluation.** When constructing a safe region for a subscriber, we start from his location and incrementally expand. In this process, the value of $b_m(\mathcal{R}, \mathcal{I})$ increases and our cost model indicates that the best safe region occurs when $b_m(\mathcal{R}, \mathcal{I}) = 1$. To test the optimality, we terminate the expansion at different values of $b_m(\mathcal{R}, \mathcal{I})$ and this value is denoted by $\beta$. As shown in Fig. 4.14, we can see that

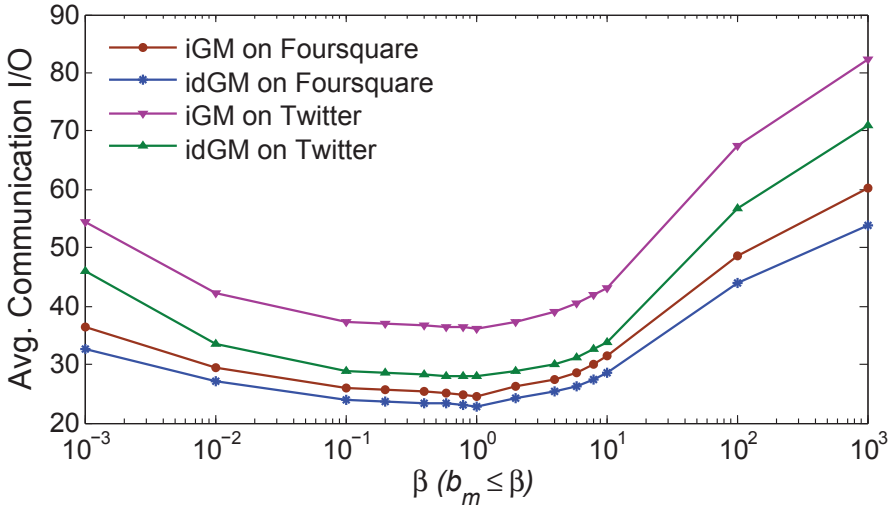
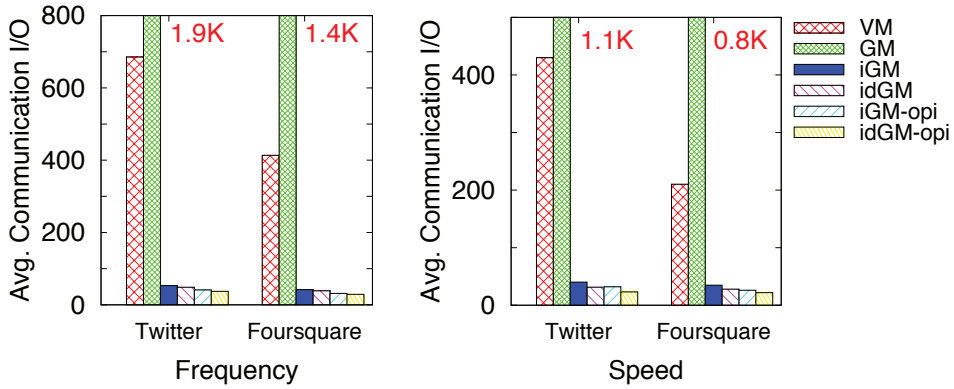



Figure 4.14: Optimality evaluation.

when terminating too early ($\beta < 1$) or too late ($\beta > 1$), the performance is inferior to the case when $\beta = 1$.




(a) Dynamic arrival rate

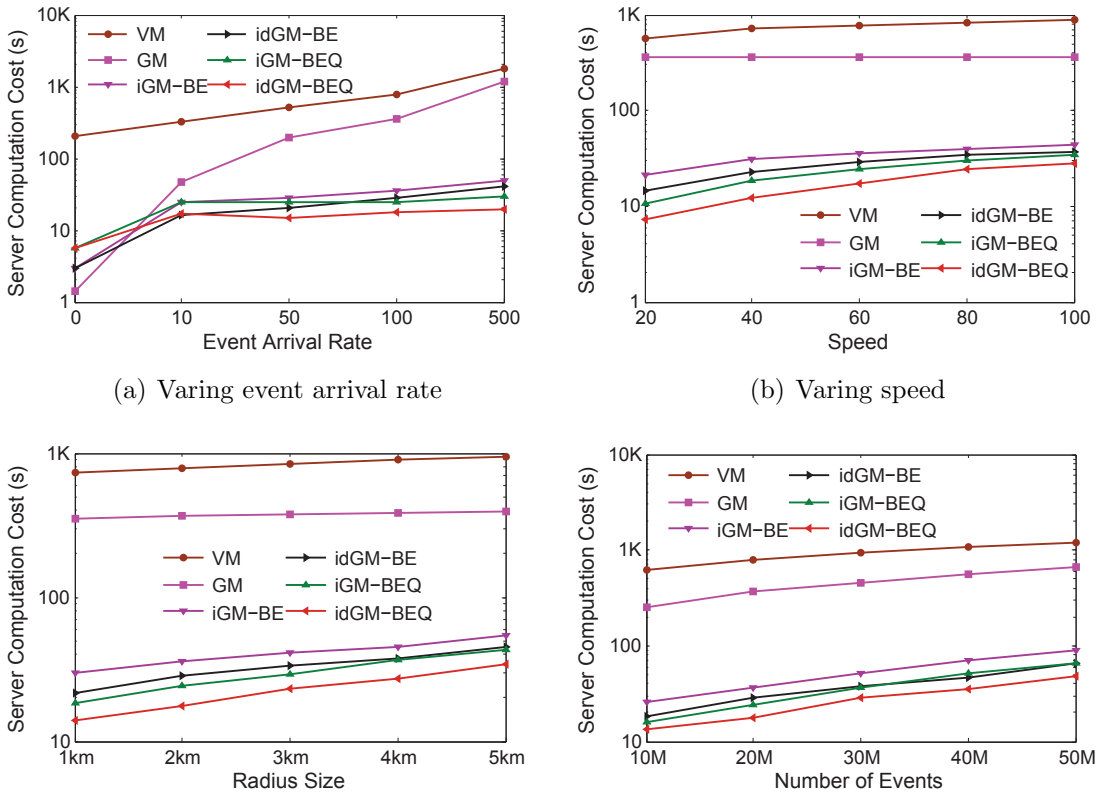
(b) Dynamic speed

Figure 4.15: Adaptability evaluation.

**Adaptability Evaluation.** To evaluate the robustness of our cost model, we look into two parameters that may change frequently in reality: the event arrival rate $f$ and the speed $v_s$ (The other parameters are dependent on the area of safe region and impact region). We set up a dynamic environment in which $f$ or $v_s$ varies all the time. For comparison, we designed two optimal methods, iGM-opi and idGM-opi with the knowledge of future pattern update in advance. When $f$ or $v$ varies, the optimal methods would construct a new safe region accordingly using the new parameters and such safe region update is not counted as a new communication I/O in the experiments. Fig. 4.15(a) and Fig. 4.15(b) show the experimental results with

dynamic $f$ and $v_s$, respectively. We gradually increase the event arrival rate $f$ from $0/tm$ to $500/tm$ and then reduce it back to $0/tm$. This process is repeated 10 times. The dynamic moving speed $v_s$ is set in a similar way ($0 \rightarrow 100m/tm \rightarrow 0m/tm$). As shown in Figure 4.15, iGM and idGM can achieve a comparable performance as iGM-opt and idGM-opt due to the good adaptability, because they can adapt to the update of $f$ and $v_s$ when the update causes a communication. This shows that our methods are robust to different user motion and event arrival patterns.

#### 4.5.3.4 Server Computation Cost




(a) Varing event arrival rate

(b) Varing speed

(c) Varing radius

(d) Varing number of events

Figure 4.16: Server computation cost for safe region construction.

In the following, we report the average server computation cost for safe region and impact region construction during the experiment period (i.e., 1000 timestamps). For VM, GM, iGM-BE, idGM-BE, we use $k$-index to find all the matching events first and then construct the safe region and impact region. For iGM-BEQ and idGM-BEQ, we use BEQ-Tree to find the set of matching events on demand, as described in Section 4.3.3. The results are shown in Fig. 4.16. We have the following observations.

(1) VM performs worse than GM in terms of computation cost although VM incurs less communication cost than GM, because VM needs more time to construct the safe region and impact region. (2) iGM and idGM can outperform VM and GM by one order of magnitude, because each communication would trigger an update for the safe region and impact region while the communication cost of iGM and idGM can be reduced significantly with the guide of the cost model. (3) iGM-BEQ and idGM-BEQ show superior performance than iGM-BE and idGM-BE, because our safe region construction methods usually do not expand to the whole space and we can use BEQ-Tree to traverse only a part of the whole space. In addition, the advantage is more obvious when the event arrival rate is higher or the number of events is larger, making our system more scalable.

### 4.5.4 Evaluation on Spatial Boolean Expression Matching

In this section, we study the efficiency in spatial boolean expression matching, which can be further categorized into the elapsed time on the boolean expression match and that on the spatial match, namely BE and Spatial, respectively. In particular, we use the Twitter dataset to compare our proposed BEQ-Tree with several baseline indexes as described at the beginning of Section 4.5.

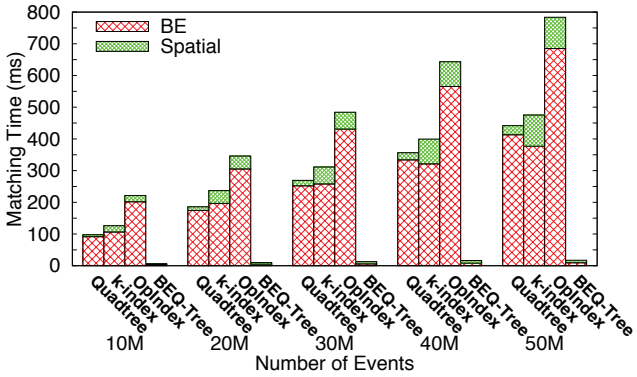



Figure 4.17: Effect of the number of events.

**Effect of the number of events $\mathbb{E}$.** We evaluate the average matching time when increasing the number of events. From the results presented in Fig. 4.17, we have the following findings: (1) Quadtree can do the spatial match quickly, but it needs much time to filter candidate events. (2) $k$-index and OpIndex need more time for spatial match compared to Quadtree. (3) BEQ-Tree outperforms the other methods, and exhibits a 97% better matching time as compared to the next best algorithm. This is because it utilizes a Quadtree-like structure in the first layer that exhibits

good pruning power, and maintains a sorted inverted list in the second layer, with which fewer events are examined. As shown, BEQ-Tree can answer the subscription matching within a few microseconds for a 20 million dataset, which is highly favored in real applications.

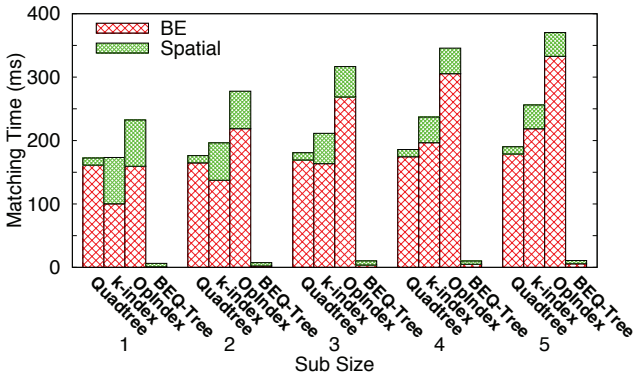



Figure 4.18: Effect of the subscription size.

**Effect of the subscription size $\delta$.** Fig. 4.18 presents the elapsed time w.r.t. the varying subscription size $\delta$. We make three observations. (1) Overall, BEQ-Tree achieves a speedup of 20 times w.r.t. Quadtree, k-index and OpIndex, respectively. (2) Regarding the boolean expression match (BE) time, more attributes are involved in the match as $\delta$ increases, resulting in a higher computation time. (3) Regarding the spatial match time, $k$-index and OpIndex are sensitive to the subscription size, because they generate fewer candidate events as $\delta$ increases.

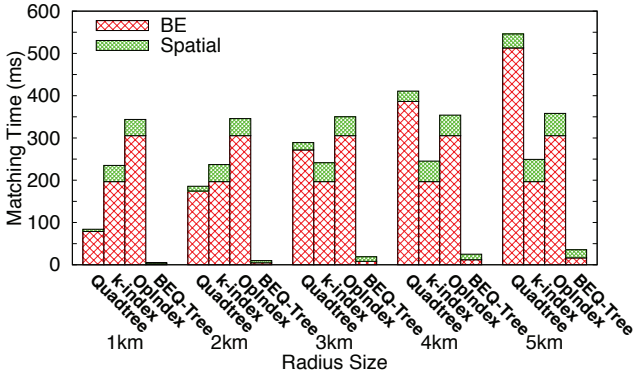



Figure 4.19: Effect of the notification radius.

**Effect of the notification radius $r$.** The matching performance w.r.t. varying notification radius $r$ is presented in Fig. 4.19. Only Quadtree is sensitive to $r$, because more candidate events are generated in the first step. In contrast, the performance of BEQ-Tree is stable and scales very well with $r$.

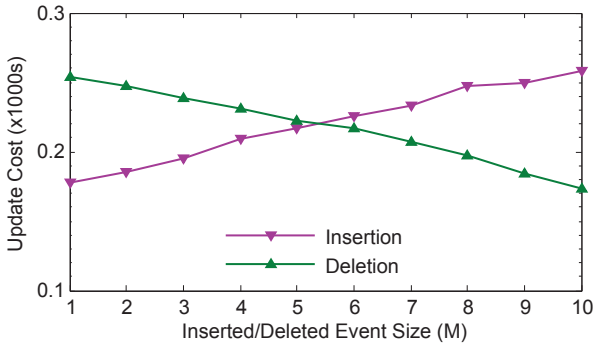



Figure 4.20: Update cost for BEQ-Tree.

**BEQ-Tree Update Cost.** At last, Fig. 4.20 shows the update cost for BEQ-Tree. To test the insertion cost, we start from a BEQ-Tree with 20 million events and incrementally insert 10 million events. The total time of inserting every 1 million tuples is plotted in Figure 4.20. We can see that as the size BEQ-Tree increases, it takes more time to insert the same number of tuples because the tree height also increases. The deletion process starts from a BEQ-Tree with 30 million events and the cost for deleting every 1 million records is reported in Figure 4.20 as well. We can see that as more records are deleted, the tree becomes smaller and the deletion operation becomes faster. In both cases, it takes less than 300 seconds to delete or insert 1 million events, which means our BEQ-Tree is efficient in update.

## 4.6 Summary

In this chapter, we build a novel location-aware pub/sub system, Elaps, which takes into account continuous moving queries over dynamic event streams. To reduce communication overhead, we propose a concept named impact region and a novel cost model. Based on the cost model, we propose two incremental methods to construct the safe region and impact region. To reduce the response time of Elaps, we propose a novel index BEQ-Tree which can support efficient spatial subscription matching over a collection of events in the dynamic event environment. Experimental results on real datasets show that Elaps can greatly reduce the communication overhead and disseminate events to users in real-time.

# Chapter 5

# Efficient Optimal Trajectories Queries for Influence Maximization

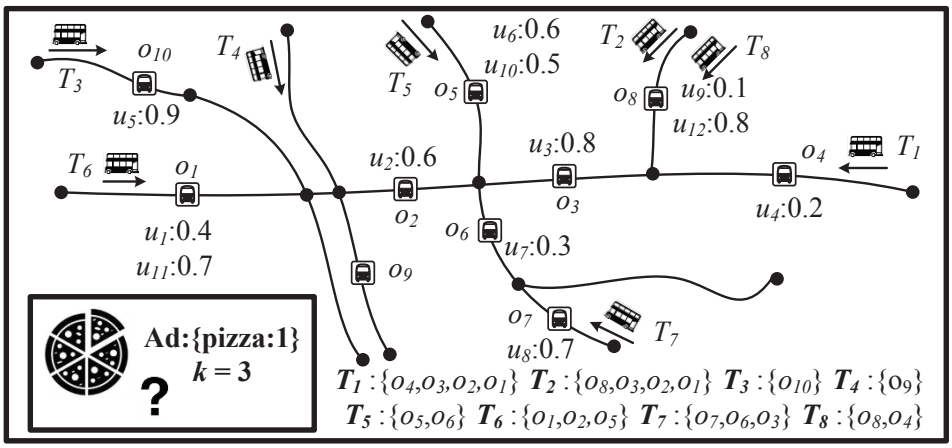
## 5.1 Introduction

In Chapter 3 and Chapter 4, we process the moving spatial queries that only focuses on real-time location data of moving objects to improve location-based services. With the increasing availability of trajectory data, it is interesting and important to be able to use trajectories to support many real-life applications. In this chapter, we look at how trajectories can be exploited for mobile advertisement. In particular, we extend the traditional influence maximization problem to the trajectory databases to maximize the influence of an advertisement by mining important motion patterns of users from their historical movement data.

Influence Maximization in a social network is a key algorithmic problem behind online viral marketing. By word-of-mouth propagation effect among friends, it finds a set of $k$ seeds to maximize the expected influence among all the users. It has attracted significant attention from both academic and industry communities due to its potential commercial value, such as viral marketing [9, 30, 39], rumor control and information monitoring [25, 40, 60].

There have been some efforts to extend the influence maximization problem to topic-aware influence maximization [7, 4, 16, 15] so as to support online advertisements. The propagation model is required to take into account influence probability based on different topics so that relevant users play a more important role in the advertisement propagation. The topic-aware influence maximization can be used in many fields, such as topic-aware advertisement and topic-aware rumor control.

To our knowledge, we are the first to study influence maximization in a trajectory database. In our data model, a trajectory derived from a vehicle can be attached




| | | |
|---|---|---|
| $u_1$: {NBA 0.8, pizza 0.3} | $u_2$: {pizza 0.9, NBA 0.4} | $u_3$: {pizza 0.6, NBA 0.8} |
| $u_4$: {pizza 0.5, swim 0.5} | $u_5$: {pizza 0.9, gym 0.2} | $u_6$: {pizza 0.3, swim 0.9} |
| $u_7$: {pizza 0.9, gym 0.1} | $u_8$: {pizza 0.5, gym 0.5} | $u_9$: {pizza 0.1, NBA 0.9} |
| $u_{10}$: {NBA 0.9, gym 0.1} | $u_{11}$: {gym 0.9, swim 0.4} | $u_{12}$:{gym 0.9, swim 0.4} |

Figure 5.1: A working scenario.

with an advertisement. An audience is associated with a personal profile as well as motion patterns. If a user meets a moving vehicle and the profile matches the carried advertisement, we consider the user influenced by the trajectory. Given an advertisement, our goal is to find $k$ trajectories such that when they are attached with the advertisement, the expected influence is maximized. The problem finds many useful applications in offline local marketing.

**Mobile advertisement management**. Nowadays, mobile advertisement (e.g., bus or subway) offers advertisers the opportunity to reach consumers seamlessly as they spend more time commuting out of home. With the help of the trajectory influence maximization, mobile advertisement can be well placed to reach consumers on the go, delivering advertising messages to the right audience at the right time. Fig. 5.1 shows a working scenario with 8 trajectories from different buses. Each user is associated with a profile as well as motion patterns. For example, $u_2$ likes pizza and visits the bus stop $o_2$ with probability 0.6 in a certain time period [1]. Given an advertisement for "pizza", our goal is to find top-$k$ buses (trajectories) to carry the advertisement to maximize the influence, where $k$ is determined by the campaign budget

**Route recommendation systems**. Given $k$ vehicles decorated for a presidential election or a boxing match waiting for route navigation, the trajectory influence

[1]For the ease of presentation, the temporal attribute is not displayed in the figure.

maximization problem can help select $k$ best trajectories from a large collection of candidates to maximize the influence of the promotion activity.

In this work, we formulate the trajectory influence maximization problem and show that it is NP-hard. To find the exact top-$k$ trajectories, we propose an expansion-based framework that enumerates the trajectory combinations in a best-first manner. The algorithm starts by calculating the influence score of each trajectory w.r.t. to the query advertisement. The trajectories are then sorted by the influence and accessed accordingly. In each iteration, combinations with the new trajectory are enumerated. If a combination contains fewer than $k$ trajectories, it is considered *incomplete* and we estimate its upper bound influence from the unvisited trajectories. If a combination is *complete*, we calculate its exact influence score. The algorithm terminates when the upper bound influence score of all the incomplete combinations are smaller than the best result ever found. To further improve the performance, we propose two effective and tighter upper bound estimation methods based on a new concept named *incremental influence* for early termination.

However, the expansion-based method is not scalable when $k$ is large. This is because the number of candidates grows exponentially with $k$, resulting in high computation cost and memory consumption. To address the issue, we propose three approximate methods with performance guarantees to solve the problem. The first is a baseline greedy algorithm which finds the trajectory with the maximum incremental influence at each iteration until $k$ trajectories are found and achieves a $(1 - 1/e)$ approximation ratio. The second is a cluster-based algorithm that further improves the efficiency of the greedy algorithm and guarantees the same approximation ratio. It partitions the trajectory database into clusters and allows us to access the clusters in an order such that promising trajectories will be found earlier. Our third approximate solution, named threshold-based method, provides a flexible means to adjust the tradeoff between efficiency and accuracy using a threshold $\epsilon$. It guarantees a $\epsilon$ approximation ratio for any $\epsilon \in (0, 1]$. In addition, we propose a group greedy method to support the influence maximization for a group of advertisements, which selects the trajectories by considering all the advertisements simultaneously and can guarantee a $(1 - 1/e)$ approximation ratio.

To sum up, our contributions of this chapter include:

- We are the first to study and formulate the influence maximization problem in trajectory databases.

- We devise an expansion-based framework that enumerates the trajectory combination in a best-first manner. In addition, we design two effective upper bound estimation techniques based on a concept named incremental influence.
- We propose three approximate methods with performance guarantees to solve the problem when $k$ is large.
- We extend the influence maximization problem to find $k$ best trajectories for a group of advertisements .
- We use real datasets to construct user profiles, motion patterns and trajectory databases. Experimental results show that our proposed methods can solve the trajectory influence maximization problem efficiently.

The rest of this chapter is organized as follows. We first formulate the problem in Section 5.2. The exact algorithm is proposed in Section 5.3. We present two effective methods for the upper bound estimation in Section 5.4. We further propose three approximate methods with performance guarantees in Section 5.5. The solution for influence maximization for a group of advertisements is provided in Section 5.6. In Section 5.8, we conduct extensive performance evaluations for our proposed methods. Finally, we conclude this chapter in Section 5.9.

## 5.2 Problem Definition

In this work, we study a novel problem named trajectory influence maximization. Given an advertisement, the query finds top-$k$ trajectories with the maximum expected influence to a large group of audience. Table 5.1 summarizes the notations frequently used in the chapter. In our data model, three types of roles are involved:

**Audience.** An audience $u$ is modeled as a text profile associated with spatial-temporal patterns. It captures a user's preference in terms of tags and the likelihood to visit a region in certain time period. In this work, we use a POI (place of interest) to represent a region. We partition a day into a fixed number of time periods $\{t_1, t_2, \ldots\}$ and formally represent each audience as $u = \{\mathtt{t}, (o_1, t_1, p_1), \ldots, (o_m, t_m, p_m)\}$, where $\mathtt{t}$ are tags of the audience, $o_i$ is a POI, $t_i$ is a period and $p_i \in (0, 1]$ refers to the probability that $u$ visits $o_i$ during time period $t_i$.

**Trajectory.** A trajectory is generated by a vehicle which serves as the carrier for an advertisement. Traditionally, a trajectory is represented by a sequence of timestamped geo-coordinates. In this work, we preprocess a trajectory and represent

| Symbol | Description |
|---|---|
| $T$ | a trajectory |
| $q$ | an advertisement |
| $Q$ | a group of advertisement |
| $u$ | an audience |
| $U$ | an audience set |
| $\mathcal{T}$ | a trajectory database |
| $o$ | a POI |
| $\mathtt{o}$ | a POI attached with a time period |
| $\sigma(u, q)$ | the textual relevance score between $u$ and $q$ |
| $S$ | a trajectory set with $k$ trajectories |
| $\mathcal{I}(q, u, T)$ | the influence between user $u$ and a trajectory $T$ with advertisement $q$ |
| $\mathcal{I}(q, u, S)$ | the influence between $u$ and a set of trajectories $S$ with advertisement $q$ |
| $\mathcal{I}(q, U, S)$ | the expected influence between a group of audience $U$ and a set of trajectories $S$ with advertisement $q$ |
| $\mathcal{I}(Q, U, S)$ | the expected influence between a group of audience $U$ and a set of trajectories $S$ with a group of advertisements $Q$ |
| $\mathcal{I}(q, U, T\|S)$ | the incremental influence by adding $T$ into the trajectory set $S$ |
| $C$ | an incomplete candidate |
| $C_n$ | a complete candidate |
| $\mathbb{C}$ | a candidate set |
| $OD(T_i, T_j)$ | the overlap distance between $T_i$ and $T_j$ |

Table 5.1: Summary of Notations

a trajectory by a sequence of POIs [2] along the roads, each associated with a time period when the POI is visited by the vehicle. Let $T$ denote a trajectory and $T = \{\mathtt{o}_1 : (o_1, t_1), \ldots, \mathtt{o}_n : (o_n, t_n)\}$, where $o_i$ is a POI and $t_i$ is the time period when the trajectory passes the POI.

**Advertisement.** An advertisement $q$ is represented by a set of weighted tags. Our goal is to find the top-$k$ best trajectories to carry the advertisement and generate the maximum influence.

In the following, we propose how to define the expected influence and present a formal problem definition. We then prove that the new problem is NP-hard.

[2]The notion of POI refers to any two-dimensional location that can be semantically annotated.

### 5.2.1 Trajectory Influence

Given an advertisement $q$, we can define the influence score between an audience $u$ and a trajectory $T$ carrying $q$ as follows:

$$\mathcal{I}(q, u, T) = \sigma(q, u) \cdot \rho(u, T) \tag{5.1}$$

where $\sigma(q, u)$ measures the textual relevance between an advertisement and an audience, and $\rho(u, T)$ measures the influence probability that an audience will "meet" a vehicle that carries the advertisement. Such an influence score captures the textual relevance, spatial relevance and temporal relevance between $u$ and a vehicle attached with $q$ moving along $T$.

Given an advertisement $q$, an audience $u$ is influenced by an advertisement $q$ attached on a trajectory $T$ if the following two conditions are satisfied:

1. $\sigma(q, u) > 0$.
2. $\rho(u, T) > 0$, i.e., $\exists(o_i, t_i) \in T \wedge \exists(o_j, t_j, p_j) \in u$ such that $o_i = o_j$ and $t_i = t_j$.

In other words, if an audience is influenced by a trajectory carrying advertisement $q$, they have to be matching in the textual, spatial and temporal attributes.

*Textual relevance* ($\sigma$). Textual relevance can be captured by any information retrieval model. In this work, cosine similarity is used to evaluate the similarity between $q$ and $u$, which is defined as

$$\sigma(q, u) = \frac{q.\mathtt{t} \cdot u.\mathtt{t}}{\|q.\mathtt{t}\| \, \|u.\mathtt{t}\|} = \frac{\sum\limits_{w \in q.\mathtt{t}} q.\mathtt{t}[w] \cdot u.\mathtt{t}[w]}{\sqrt{\sum\limits_{w \in q.\mathtt{t}} (q.\mathtt{t}[w])^2 \cdot \sum\limits_{w \in u.\mathtt{t}} (u.\mathtt{t}[w])^2}} \tag{5.2}$$

where $q.\mathtt{t}[w]$ is the weight of $w$ in an advertisement and $u.\mathtt{t}[w]$ is the weight of $w$ in the audience profile.

*Influence probability* ($\rho$). Let $M(u, T)$ denote the spatial-temporal components of an audience $u$ that match a trajectory $T$. We have

$$M(u, T) = \{j | \exists(o_j, t_j, p_j) \in u \; \exists(o_i, t_i) \in T, o_i = o_j \wedge t_i = t_j\} \tag{5.3}$$

Then, influence probability can be defined as

$$\rho(u, T) = 1 - \prod_{j \in M(u,T)} (1 - p_j) \tag{5.4}$$

where $\prod_{j \in M(u,T)} (1 - p_j)$ measures the probability that an audience will not "meet" a vehicle that carries the advertisement.

Similar to Eqn. 5.1, we measure the influence between an audience $u$ and a set of trajectories $S = \{T_1, T_2, \ldots, T_k\}$ carrying the same advertisement $q$ as follows:

$$\begin{aligned} \mathcal{I}(q, u, S) = &\ \sigma(q, u) \cdot \rho(u, S) \\ = &\ \sigma(q, u) \cdot (1 - \prod_{j \in M(u,T_1) \bigcup \ldots \bigcup M(u,T_k)} (1 - p_j)) \end{aligned} \tag{5.5}$$

Again, $\prod_{j \in M(u,T_1) \bigcup \ldots \bigcup M(u,T_k)} (1 - p_j)$ measures the probability that $u$ will not be influenced by any of the $k$ trajectories in $S$.

### 5.2.2 Problem Definition

Let $U$ denote a group of audience. We define the expected influence between a group of audience $U$ and a set of trajectories $S$ with advertisement $q$:

$$\mathcal{I}(q, U, S) = \sum_{u \in U} \mathcal{I}(q, u, S). \tag{5.6}$$

**Property 5.1** *$\mathcal{I}(q, U, S)$ is a submodular function, i.e., for any two trajectory sets $S_1$ and $S_2$, we have $\mathcal{I}(q, U, S_1) + \mathcal{I}(q, U, S_2) \geq \mathcal{I}(q, U, S_1 \cup S_2) + \mathcal{I}(q, U, S_1 \cap S_2)$*

Now, we are ready to present the influence maximization problem in a trajectory database.

**Definition 5.1 (Trajectory Influence Maximization)** *Given a trajectory database $\mathcal{T}$ and a group of audience $U$ attached with profiles and spatial-temporal patterns, for an advertisement $q$, our goal is to find a trajectory set $S$ where $S \subset \mathcal{T}$ and $|S| = k$ such that the expected influence $\mathcal{I}(q, U, S)$ is maximized.*

We show the hardness of the trajectory influence maximization problem.

**Theorem 5.1** *The trajectory influence maximization problem is NP-hard.*

**Proof 5.1** *We prove by reducing the Set Cover problem to our trajectory influence maximization problem. In the Set Cover problem, given a collection of subsets $S_1, S_2, ..., S_m$ of a ground set $G = \{g_1, g_2, ..., g_n\}$, we wish to know whether there exist $k$ of the subsets whose union is equal to $G$. We map each subset $S_i$ in the Set Cover problem to the set of audience influenced by a trajectory $T_i \in \mathcal{T}$. We also map each $g_i \in G$ to each user $u_i \in U$. If an audience $u$ is influenced, we set the score $\mathcal{I}(q, u, T) = 1$. Consequently, if there are $n$ users influenced by a trajectory set $S$, the expected influence $\mathcal{I}(q, U, S) = n$. Based on the settings, the Set Cover problem finds $k$ subsets whose union is equal to $G$. It is equivalent to deciding if there is a $k$-advertiser set $S$ with the maximum influence $|G|$ in our problem. Since the Set Cover is NP-complete, we finish the proof.*

## 5.3 Expansion-based Algorithm

A naive solution to finding the best $k$ trajectories for advertisement is to exhaustively examine all the possible size-$k$ combinations, calculate the generated influence score for each candidate set and return the one with the maximum influence. However, the method is intractable because the number of candidate sets grows exponentially with the number of trajectories and the cost of influence calculation for each candidate set is expensive. In this section, we propose an expansion-based algorithm with early termination to improve the naive method.

### 5.3.1 Algorithm Sketch

Our expansion-based algorithm starts by sorting the trajectories in descending order according to their influence $I(q, U, T)$ upon all the audience, which is defined as

$$\mathcal{I}(q, U, T) = \sum_{u \in U} \mathcal{I}(q, u, T). \tag{5.7}$$

Then, we iteratively examine the trajectories according to the order and enumerate all the possible combinations whose size is at most $k$. If a candidate set contains $k$ trajectories, we call it a *complete candidate*, denoted by $C^*$, and can calculate its accurate influence using Eqn. 5.7. Otherwise, it contains fewer than $k$ trajectories and we call it an *incomplete candidate*, denoted by $C$. In each iteration, we check the next trajectory in the sorted list and enumerate new combinations by extending existing incomplete candidates with this trajectory. The algorithm can be safely terminated if we can guarantee that the influence score of the current best $k$-trajectory set is larger than the upper bound influence score of all the possible complete candidates that have not been enumerated.

The pseudo-code sketch of our expansion-based algorithm is illustrated in Algorithm 5.1. For each trajectory, we calculate its influence score and insert it into a list $\mathcal{L}$ sorted by the value (line 1). Then, we initialize $S$ to contain the first $k$ trajectories in the sorted list (line 2) and initialize the global maximum influence $\mathcal{I}_{opt}$ to be $\mathcal{I}(q, U, S)$ (line 3) to facilitate pruning in the following enumerations of new candidates. We also maintain a set $\mathbb{C}$ to contain the incomplete candidates that have been enumerated (line 4). The candidates in $\mathbb{C}$ are sorted by the candidate size, which refers to the number of trajectories in the candidate, in descending order. In this way, we can generate complete candidates earlier and use the updated $\mathcal{I}_{opt}$ to help pruning.

**Algorithm 5.1:** Expansion-based Algorithm

**input**: An advertisement query $q$, a trajectory database $\mathcal{T}$ and an audience set $U$

**output**: $k$-trajectory set $S$

```
1  Sort trajectories in L according to their influence I(q, U, T)
2  Initialize S to contain the first k trajectories in L
3  I_opt ← I(q, U, S)
4  C ← {∅}
5  for i = 1; i ≤ |L|; i++ do
6      T ← L[i]
7      foreach incomplete combination C ∈ C do
8          C_n ← C ⋃ {T}
9          if |C_n| < k then
10             UB_0 ← UpperBound(L, C_n, i + 1)
11             if UB_0 > I_opt then
12                 Insert C_n into C
13             overline(UB_0) = max(overline(UB_0), UB_0)
14         else
15             if I(q, U, C_n) > I_opt then
16                 S ← C_n
17                 I_opt ← I(q, U, C_n)
18         if UpperBound(L, C, i + 1) < I_opt then
19             Remove C from C
20     overline(UB'_0) ← UpperBound(L, ∅, i + 1)
21     if overline(UB_0) ≤ I_opt ∧ overline(UB'_0) ≤ I_opt then
22         break
23 return S
```

We iteratively examine the trajectories in $\mathcal{L}$ in descending order of the influence score (lines 5-22). When visiting trajectory $T$ in the $i$-th iteration, we scan all the incomplete candidates stored in $\mathbb{C}$ [3]. For each $C \in \mathbb{C}$, we first examine the new candidate $C_n$ generated by combining $C$ and $T$ (line 8). If $C_n$ is an incomplete candidate, only when its upper bound score is larger than $\mathcal{I}_{opt}$, will the new candidate be inserted into $\mathbb{C}$ (lines 9-13). If $C_n$ is a complete candidate, we calculate the exact influence using $\mathcal{I}(q, U, C_n)$ in Eqn. 5.6. If a better result is found, we update $\mathcal{I}_{opt}$ and $S$ accordingly (lines 14-17). Then, we update the upper bound score of $C$ by combining the trajectories that have not been visited. If the new upper bound is smaller than $\mathcal{I}_{opt}$, we remove $C$ from $\mathbb{C}$ (lines 18-19). Note that the upper bound for the same incomplete candidate $C$ could be different as iterations continue because we

[3] $C = \emptyset$ is also viewed as an incomplete candidate in $\mathbb{C}$

only examine its combinations with the unvisited trajectories. The reason for being aggressive in reducing the size of $\mathbb{C}$ is that each incomplete candidate may spawn exponential number of new candidates if it is not pruned early. We use $\overline{UB_0}$ to denote the upper bound influence for all the incomplete candidates in $\mathbb{C}$ and $\overline{UB'_0}$ to denote the upper bound influence for candidates that only contain unvisited trajectories. If we can guarantee that $\overline{UB_0}$ and $\overline{UB'_0}$ are both no greater than $\mathcal{I}_{opt}$, the algorithm can be terminated safely. Note that it is not safe to terminate the algorithm when only $\overline{UB_0}$ is no greater than $\mathcal{I}_{opt}$. We should also guarantee that $\overline{UB'_0}$ is no greater than $\mathcal{I}_{opt}$. This is because a better combination may be composed solely of the unvisited trajectories, due to the fact that the influence of the trajectories is not independent to each other. In the following, we present how to estimate $\overline{UB_0}$ and $\overline{UB'_0}$ for early termination.

### 5.3.2 Estimation of $\overline{UB_0}$ and $\overline{UB'_0}$

To calculate $\overline{UB_0}$, we calculate an upper bound influence $UB_0$ for each incomplete candidate $C$ in $\mathbb{C}$ and $\overline{UB_0}$ is the maximum $UB_0$. A straightforward method is to examine all the possible combinations with $k - |C|$ unvisited trajectories. The derived bound is accurate but incurs too much computational cost. A more efficient alternative is to aggregate the influence score of the first $k-|C|$ unvisited trajectories to estimate $UB_0$ for $C$. Let $\mathcal{T}_u$ denote the set of unvisited trajectories following the same order in $\mathcal{L}$ [4] and $\mathcal{T}_0$ denote a set containing the first $k - |C|$ trajectories in $\mathcal{T}_u$. We define our upper bound for an incomplete candidate $C$ as

$$UB_0 = \mathcal{I}(q, U, C) + \sum_{T \in \mathcal{T}_0} \mathcal{I}(q, U, T), \tag{5.8}$$

We prove that $UB_0$ is an upper bound score.

**Lemma 5.1** *For any complete candidate $C^*$ that contains $C$ and trajectories from $\mathcal{T}_u$, we have $\mathcal{I}(q, U, C) \leq UB_0$.*

**Proof 5.2** *Let $C_{opt}$ be the optimal candidate expanded from $C$. Due to the submodular property of the influence scoring function, we have*

$$\mathcal{I}(q, U, C_{opt}) \leq \mathcal{I}(q, U, C) + \sum_{T \in C_{opt} \setminus C} \mathcal{I}(q, U, T)$$

[4] In other words, $\mathcal{T}_u = \{\mathcal{L}[i+1], \mathcal{L}[i+2], \ldots\}$

*Since $\mathcal{T}_0$ contains trajectories with the maximum influence in $\mathcal{T}_u$, we have*

$$\sum_{T \in C_{opt} \setminus C} \mathcal{I}(q, U, T) \leq \sum_{T \in \mathcal{T}_0} \mathcal{I}(q, U, T)$$

*Therefore, we have $UB_0 \geq \mathcal{I}(q, U, C_{opt})$, which indicates $UB_0$ is an upper bound influence for any candidate expanded from $C$.*

Next we present how to estimate $\overline{UB'_0}$. It is equivalent to estimating the upper bound for an empty set using trajectories in $\mathcal{T}_u$. Thus, we can still apply Eqn. 5.8 to get the bound.

$$\overline{UB'_0} = \sum_{T \in \mathcal{T}_0} \mathcal{I}(q, U, T). \tag{5.9}$$

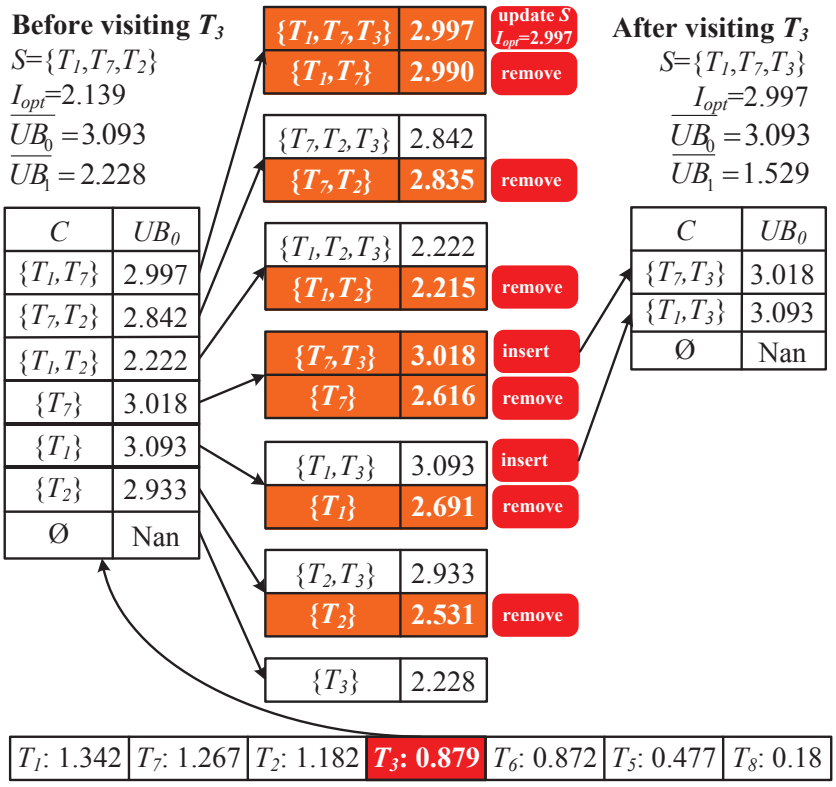



Figure 5.2: expansion-based method.

**Example 5.1** *Fig. 5.2 shows a snapshot of our expansion-based method in the 4-th iteration. Before this, we have sorted the trajectories based on the influence score in descending order, examined the top 3 trajectories $\{T_1, T_7, T_2\}$ and enumerated the incomplete candidates in the candidate table. The current best solution is to select*

$\{T_1, T_7, T_2\}$ *with an expected influence* $I_{opt} = 2.139$. *We can also estimate two upper bounds* $\overline{UB_0}$ *an* $\overline{UB'_0}$ *to be* 3.093 *and* 2.228 *respectively. For example, based on Eqn. 5.9,* $\overline{UB'_0} = \mathcal{I}(q, U, T_3) + \mathcal{I}(q, U, T_6) + \mathcal{I}(q, U, T_5) = 2.228$.

*When accessing* $T_3$ *in the* 4*-th iteration, we enumerate all the possible combinations by combining* $T_3$ *and the candidates in the candidate table. The candidates in the table are sorted by the cardinality so that complete candidates can be evaluated earlier. For each candidate* $C$ *visited, we first generate a new candidate* $C_n$ *by combining* $C$ *with* $T_3$. *If* $C_n$ *is a complete candidate, we calculate its influence to check whether it is better than* $S$. *For example, when we visit* $\{T_1, T_7\}$, *we calculate the influence for the new complete candidate* $\{T_1, T_7, T_3\}$. *Since* $\{T_1, T_7, T_3\}$ *is a better candidate with higher influence score than* $S$, *we update* $S$ *and* $I_{opt}$. *If* $C_n$ *is an incomplete candidate, we calculate its upper bound* $UB_0$ *to check whether to insert it into the candidate table. For example, when we visit* $\{T_7\}$, *we calculate the upper bound for the new incomplete candidate* $\{T_7, T_3\}$. *Since its upper bound is larger than* $\mathcal{I}_{opt}$, *it may generate a better candidate than* $S$ *and thus is inserted into the candidate table. After checking the newly generated candidate, we also need to estimate a new upper bound for the old candidate* $C$. *This is because before accessing* $T_3$, *its upper bound is estimated by taking* $T_3$ *into consideration when forming new expansions. Since we have examined* $T_3$ *in the* 4*-th iteration, their upper bound should consider trajectories excluding* $T_3$. *If the new upper bound is smaller than* $I_{opt}$, *the candidate can be pruned. For example, the old candidate* $\{T_1, T_7\}$ *has a smaller upper bound than* $\mathcal{I}_{opt}$ *and thus is removed from the candidate table. After visiting all the non-empty candidates in the candidate table, we generate a new candidate* $\{T_3\}$ *to combine* $\emptyset$ *with the unvisited trajectories. Since the candidate* $\{T_3\}$ *has a smaller upper bound than* $\mathcal{I}_{opt}$, *we do not insert it into the candidate table.*

*After visiting* $T_3$, *we have only two incomplete candidates left in* $\mathbb{C}$. *The current best* $k$*-advertiser set* $S$ *is* $\{T_1, T_7, T_3\}$ *and* $\mathcal{I}_{opt}$ *increases from* 2.139 *to* 2.997. *The algorithm will terminate when the termination condition* $max(\overline{UB_0}, \overline{UB'_0}) < \mathcal{I}_{opt}$ *is satisfied.*

### 5.3.3 Index-based Optimization

In the initialization steps of Algorithm 5.1, we need to calculate the influence for each trajectory and sort them in $\mathcal{L}$. To calculate $\mathcal{I}(q, U, T)$ for a trajectory $T$, we need to first identify audience that intersect with $T$ in spatial, temporal and textual attributes. Then, among these influenced audience, we aggregate the influence score.

When there are a large number of trajectories and audience, the computation overhead could be very high.

To reduce the cost of influence calculation, a straightforward method is to maintain an inverted index between trajectories and users based on two-level partitioning. In the first level, we partition by trajectories. For each trajectory, we maintain a partition containing users who could be influenced by the trajectory in the spatial and temporal attributes. In the second level, we maintain inverted lists for the user profiles. To calculate the influence score of $T$ w.r.t. an advertisement $q$, we retrieve the inverted lists relevant to $q$ and located in the partition of $T$, scan the users and aggregate their scores.

Although efficient, we still need to traverse the users to calculate the influence. To further reduce the computation time, we propose a method that converts the scoring function $\mathcal{I}(q, U, T)$ into the form of $f(U, T) \cdot g(q, U, T)$, where $f(U, T)$ is a term independent of query $q$ and can be pre-computed offline while $g(q, U, T)$ is a term that needs to be computed online. To achieve the goal, we define

$$\lambda_{q,\mathtt{w}} = \frac{q.\mathtt{t}[\mathtt{w}]}{\sqrt{\sum_{w \in q.\mathtt{t}} (q.\mathtt{t}[w])^2}}, \lambda_{u,\mathtt{w}} = \frac{u.\mathtt{t}[\mathtt{w}]}{\sqrt{\sum_{w \in u.\mathtt{t}} (u.\mathtt{t}[w])^2}}$$

and get a new representation of the relevance $\sigma(q, u)$ as follows.

$$\sigma(q, u) = \sum_{w \in q.\mathtt{t}} \lambda_{q,\mathtt{w}} \cdot \lambda_{u,\mathtt{w}}. \tag{5.10}$$

To avoid iterating users when calculating the influence score, we pre-compute the influence of a trajectory based on a keyword $w$ and define a partial influence $\mathcal{I}_p(w, U, T)$ for each trajectory and each keyword:

$$\mathcal{I}_p(w, U, T) = \sum_{u \in U} \mathcal{I}_p(w, u, T) = \sum_{u \in U} \lambda_{u,w} \cdot (1 - \prod_{j \in M(u,T)} (1 - p_j)), \tag{5.11}$$

We can see that the partial influence is a term not relevant to a query advertisement $q$ and can be pre-computed offline. Then, we can get a new representation of $\mathcal{I}(q, U, T)$ to be

$$\mathcal{I}(q, U, T) = \sum_{w \in q.\mathtt{t}} \lambda_{q,w} \cdot \mathcal{I}_p(w, U, T), \tag{5.12}$$

because

$$\begin{aligned}
& \sum_{w \in q.\mathtt{t}} \lambda_{q,w} \cdot \mathcal{I}_p(w, U, T) \\
= & \sum_{w \in q.\mathtt{t}} \lambda_{q,w} \cdot \sum_{u \in U} \lambda_{u,w} \cdot (1 - \prod_{j \in M(u,T)} (1 - p_j)) \\
= & \sum_{w \in q.\mathtt{t}} \sum_{u \in U} (\lambda_{q,w} \cdot \lambda_{u,w}) \cdot (1 - \prod_{j \in M(u,T)} (1 - p_j)) \\
= & \sum_{u \in U} (\sum_{w \in q.\mathtt{t}} \lambda_{q,w} \cdot \lambda_{u,w}) \cdot (1 - \prod_{j \in M(u,T)} (1 - p_j)) \\
= & \sum_{u \in U} \sigma(q, u) \cdot (1 - \prod_{j \in M(u,T)} (1 - p_j)) \\
= & \sum_{u \in U} \mathcal{I}(q, u, T).
\end{aligned}$$

Based on Eqn. 5.12, we propose a trajectory index $\mathcal{N}_t$ to compute the influence for the trajectories efficiently, as follows. For each keyword $w$, we maintain an inverted list containing trajectories whose partial influence score w.r.t. $w$ is not zero. i.e., $\mathcal{I}_p(w, U, T) \neq 0$. The inverted list is sorted by the trajectory id. Then given an advertisement $q$ and trajectory $T$, we can easily retrieve the partial influence score in each relevant list. With the partial influence scores, we can efficiently compute the final influence based on Eqn. 5.12. With the help of $\mathcal{N}_t$, we can reduce the computation time and index size significantly compared to the naive inverted index, because $\mathcal{I}_p(w, U, T)$ is pre-computed and $\mathcal{N}_t$ only includes the trajectories whose number is much smaller than the audience.

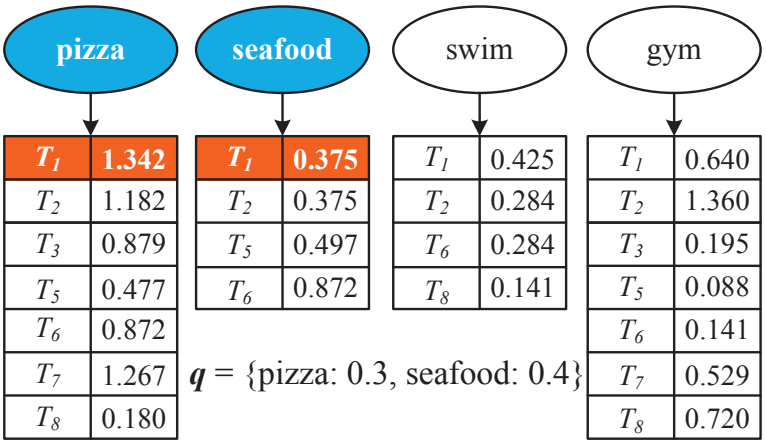



Figure 5.3: Trajectory Index.

**Example 5.2** *Fig. 5.3 shows the trajectory index $\mathcal{N}_t$ built for the trajectories in Fig. 4.1. Each keyword is associated with a trajectory list, in which we calculate*

*the partial influence for each trajectory and sort them by the trajectory id. Given an advertisement* $\{pizza : 0.3, seafood : 0.4\}$*, we first access the two trajectory lists matching the keywords in the advertisement, and then scan them by the trajectory id for partial score aggregation in a DAAT (Document-At-A-Time) manner. For example,* $T_1$ *exists in both the lists under pizza and seafood. Thus, its influence is calculated by aggregating the two partial scores in lists of "pizza" and "seafood", i.e.,* $\mathcal{I}(q, U, T_1) = (0.3/\sqrt{0.3^2 + 0.7^2}) \cdot 1.342 + (0.7/\sqrt{0.3^2 + 0.7^2}) \cdot 0.375 = 0.873$.

## 5.4 Improved Bound Estimation

In the expansion-based method, we propose an upper bound $UB_0$ for an incomplete combination $C$ which picks the top $k - |C|$ trajectories with the largest influence. Although $UB_0$ is computationally efficient, it is too loose to facilitate pruning. In this section, we propose two improved estimations of the upper bound.

### 5.4.1 Estimate $UB_1$ by Incremental Influence

We propose a new concept named *incremental influence* to estimate a tighter upper bound $UB_1$. Given an incomplete candidate $C$, let $\mathcal{I}(q, U, T|C)$ denote the incremental influence by adding $T$ into the incomplete trajectory set $C$. Formally, we have

$$\mathcal{I}(q, U, T|C) = \mathcal{I}(q, U, \{T \cup C\}) - \mathcal{I}(q, U, C) \tag{5.13}$$

Since $\mathcal{I}(q, U, C)$ is a submodular function, the incremental influence function $\mathcal{I}(q, U, T|C)$ is also a submodular function based on its definition. Then we have the following property.

**Property 5.2** *For any two sets* $S_1$ *and* $S_2$*, we have* $\mathcal{I}(q, U, S_1|C) + \mathcal{I}(q, U, S_2|C) \geq \mathcal{I}(q, U, S_1 \cup S_2|C) + \mathcal{I}(q, U, S_1 \cap S_2|C)$.

Based on the submodular property, we have the following lemma.

**Lemma 5.2** *Given an incomplete candidate* $C$ *and a complete candidate* $C^*$ *expanded from* $C$*, we have*

$$\mathcal{I}(q, U, C^*) \leq \mathcal{I}(q, U, C) + \sum_{T \in C^* \setminus C} \mathcal{I}(q, U, T|C).$$

**Proof 5.3**

$$\begin{aligned} & \mathcal{I}(q,U,C) + \sum_{T \in C^* \setminus C} \mathcal{I}(q,U,T|C) \\ \geq \; & \mathcal{I}(q,U,C) + \mathcal{I}(q,U,(C^*-C)|C) \\ = \; & \mathcal{I}(q,U,C^*) \end{aligned}$$

For each unvisited trajectory in the sorted list $\mathcal{L}$, we can calculate its incremental influence with respect to $C$. Suppose set $\mathcal{T}_1$ contains $k-|C|$ unvisited trajectories with the maximum incremental influence, we can define a new upper bound for candidate $C$ as follows:

$$UB_1 = \mathcal{I}(q,U,C) + \sum_{T \in \mathcal{T}_1} \mathcal{I}(q,U,T|C) \tag{5.14}$$

**Lemma 5.3** *For any complete candidate $C^*$ that contains $C$ and trajectories from $\mathcal{L}$, we have $\mathcal{I}(q,U,C^*) \leq UB_1$.*

**Proof 5.4** *Based on Lemma 5.2, for any complete candidate $C^*$ expanded from $C$, we have*

$$\begin{aligned} \mathcal{I}(q,U,C^*) &\leq \mathcal{I}(q,U,C) + \sum_{T \in C^* \setminus C} \mathcal{I}(q,U,T|C) \\ &\leq \mathcal{I}(q,U,C) + \sum_{T \in \mathcal{T}_1} \mathcal{I}(q,U,T|C) \end{aligned}$$

Furthermore, we prove that $UB_1$ is a tighter bound than $UB_0$.

**Lemma 5.4** $UB_1 \leq UB_0$

**Proof 5.5**

$$\begin{aligned} UB_1 &= \mathcal{I}(q,U,C) + \sum_{T \in \mathcal{T}_1} \mathcal{I}(q,U,T|C) \\ &\leq \mathcal{I}(q,U,C) + \sum_{T \in \mathcal{T}_1} \mathcal{I}(q,U,T) \\ &\leq \mathcal{I}(q,U,C) + \sum_{T \in \mathcal{T}_0} \mathcal{I}(q,U,T) \\ &= UB_0 \end{aligned}$$

To estimate the new upper bound defined in Eqn. 5.14, we need to find $k-|C|$ trajectories in $\mathcal{L}$ with the largest incremental influence. We propose an efficient method in Algorithm 5.2 in which we iteratively access the trajectories in $\mathcal{L}$ according to its original order. Our pruning is based on an important observation stated in the following lemma.

**Algorithm 5.2:** UpperBound($\mathcal{L}, C, i+1$)

```
1  Build a max-heap H ← ∅
2  UB_1 ← I(q, U, C)
3  j ← 0
4  while j < k − |C| do
5  |  T ← L[i + 1]
6  |  if I(q, U, T) > H.top() then
7  |  |  Insert I(q, U, T|C) into heap H
8  |  |  i ← i + 1
9  |  else
10 |  |  UB_1 ← UB_1 + H.top()
11 |  |  j ← j + 1
12 |  |  H.pop()
13 return UB_1
```

**Lemma 5.5** *$\mathcal{I}(q, U, \mathcal{L}[j])$ is an upper bound of incremental influence for all the trajectories $\mathcal{L}[m]$, where $m > j$.*

**Proof 5.6** *Since $\mathcal{L}$ is sorted by the influence score, we have*

$$\mathcal{I}(q, U, \mathcal{L}[m]|C) \leq \mathcal{I}(q, U, \mathcal{L}[m]) \leq \mathcal{I}(q, U, \mathcal{L}[j])$$

The max-heap $\mathcal{H}$ is used to contain the incremental influence of the trajectory and the incremental influence are sorted in a descending order (line 1). If the influence score of $T$ is larger than the top incremental influence in $\mathcal{H}$, we simply calculate the incremental influence of $T$ and insert the incremental influence into $\mathcal{H}$ (lines 7-8). Otherwise, we can guarantee that the incremental influence of all the remaining trajectories in $\mathcal{L}$ will not be better than the one in the heap based on Lemma 5.5. Thus, we consider it as a top-$k$ answer in terms of incremental influence. We update $UB_1$ and pop it from $\mathcal{H}$ (lines 10-12). The algorithm terminates when the top-$k$ trajectories with the maximum incremental influence are identified.

### 5.4.2 Upper bound $UB_2$ with Better Tradeoff

Although $UB_1$ is a tighter bound, its estimation in Algorithm 5.2 requires frequent calculation of incremental influence $\mathcal{I}(q, U, T|C)$, whose computational cost is expensive. To achieve a better tradeoff between efficiency and effectiveness in pruning, we propose a new upper bound $UB_2$ which is slightly looser than $UB_1$ but can be computed much more efficiently. More specifically, we define our new bound

$$UB_2 = \mathcal{I}(q, U, C) + \sum_{u \in \mathcal{T}_2} \hat{\mathcal{I}}(q, U, T|C) \tag{5.15}$$

by replacing the exact incremental influence $\mathcal{I}(q, U, T|C)$ in $UB_1$ with its upper bound $\hat{\mathcal{I}}(q, U, T|C)$ and $\mathcal{T}_2$ is the set of $k - |C|$ trajectories with the largest $\hat{\mathcal{I}}(q, U, T|C)$. In the following, we introduce how to find such a bound that is computationally efficient and only slightly larger than $\mathcal{I}(q, U, T|C)$.

Let $\mathcal{I}(q, U, \mathtt{o})$ denote the influence of a POI in the time period $\mathtt{o}.t_i$. It is equivalent to the influence $\mathcal{I}(q, U, T)$ of a trajectory $T$ with $T = \{\mathtt{o}\}$. Similarly, we can define incremental influence $\mathcal{I}(q, U, \mathtt{o}|C)$ w.r.t. to an incomplete candidate $C$ to be

$$\mathcal{I}(q, U, \mathtt{o}|C) = \mathcal{I}(q, U, C \cup \{\mathtt{o}\}) - \mathcal{I}(q, U, C). \tag{5.16}$$

Then, we can use $\mathcal{I}(q, U, \mathtt{o}|C)$ to estimate the upper bound of $\mathcal{I}(q, U, T|C)$ by defining

$$\hat{\mathcal{I}}(q, U, T|C) = \sum_{\mathtt{o} \in \{T \setminus \mathbb{O}\}} \mathcal{I}(q, U, \mathtt{o}). \tag{5.17}$$

where $\mathbb{O}$ is a set containing all the POIs of the trajectories in $C$. We prove that $\hat{\mathcal{I}}(q, U, T|C)$ is an upper bound of $\mathcal{I}(q, U, T|C)$.

**Lemma 5.6** $\mathcal{I}(q, U, T|C) \leq \hat{\mathcal{I}}(q, U, T|C)$.

**Proof 5.7** *Based on the submodular property of* $\mathcal{I}(q, U, T|C)$ *(see Property 5.2), we have*

$$\begin{aligned} \mathcal{I}(q, U, T|C) &= \mathcal{I}(q, U, \{\mathtt{o}_1, \mathtt{o}_2, ...\mathtt{o}_n\}|C) \\ &\leq \sum_{\mathtt{o} \in T} \mathcal{I}(q, U, \mathtt{o}|C) \end{aligned}$$

*If a POI* $\mathtt{o}$ *belongs to* $\mathbb{O} \cap T$*, the incremental influence of* $\mathtt{o}$ *would be 0. Thus, we have*

$$\begin{aligned} \mathcal{I}(q, U, T|C) &\leq \sum_{\mathtt{o} \in \{T \setminus \mathbb{O}\}} \mathcal{I}(q, U, \mathtt{o}|C) \\ &\leq \sum_{\mathtt{o} \in \{T \setminus \mathbb{O}\}} \mathcal{I}(q, U, \mathtt{o}). \end{aligned}$$

The estimation of $\hat{\mathcal{I}}(q, U, T|C)$ can be done efficiently by maintaining an inverted index $\mathcal{N}_o$ for the POIs. For each keyword, we first retrieve the relevant audience in the textual attribute. Then, we find the set of POIs that can influence these users and put them in the inverted list of the keyword. For each POI in the list, we pre-compute its partial influence score $\mathcal{I}_p(w, U, \mathtt{o})$. When calculating $\mathcal{I}(w, U, \mathtt{o})$, we simply aggregate the partial influences according to Eqn. 5.12 in Section 5.3.

## 5.5 Approximation Methods

Although the bound-based algorithms can find the accurate top-$k$ trajectories, they are not scalable to $k$ because the number of candidates grows exponentially with $k$. When $k$ is large, the computation becomes expensive and the memory cost is not affordable. In this section, we propose three approximation methods with performance guarantees to solve the problem.

### 5.5.1 Baseline Greedy Algorithm

We can extend existing greedy framework to support our problem [40], which starts by picking the trajectory with the maximum influence $\mathcal{I}(q, U, T)$. In the following $k-1$ iterations, it greedily selects the trajectory with the maximum incremental influence $\mathcal{I}(q, U, T|S)$ where $S$ contains the candidates selected in previous steps. The algorithm is more efficient than the accurate methods because it only involves $k$ iterations. The greedy algorithm guarantees an approximation ratio of $(1 - 1/e)$ based on the the following theorem.

**Theorem 5.2** *[40] For a non-negative, monotone submodular function $f$, let $S$ be a set of size $k$ obtained by selecting elements one at a time, each time choosing an element that provides the largest incremental increase in the function value. Let $S^*$ be a set that maximizes the value of $f$ over all $k$-element sets. Then $S$ provides a $(1 - 1/e)$-approximation.*

However, the greedy algorithm has a limitation that in each iteration it computes the incremental influence for every unvisited trajectory which is expensive. Thus, we optimize the greedy algorithm by avoiding examining all the unvisited trajectories, which is achieved by scanning the trajectories in order and terminating as early as possible, and avoiding calculating the incremental influence for each examined trajectory, which is achieved by utilizing the estimated incremental influence $\hat{\mathcal{I}}(q, U, T)$ proposed in Eqn. 5.17 in Section 5.4.2. We assume that the trajectories in $\mathcal{L}$ are visited in descending order of their $\mathcal{I}(q, U, T)$. We also maintain a max-heap $\mathcal{H}$ which sorts the trajectories in descending order of their estimated incremental influence $\hat{\mathcal{I}}(q, U, T)$. In each iteration, we use Algorithm 5.3 to select the trajectory $T_{max}$ with the maximum incremental influence. The procedure is as follows. If the influence score $\mathcal{I}(q, U, T)$ of a trajectory $T$ in $\mathcal{L}$ is larger than the estimated incremental influence of the top trajectory in $\mathcal{H}$, we calculate the estimated incremental influence of $T$ instead of its exact incremental influence and insert it into $\mathcal{H}$ (lines 5-7). Only when

**Algorithm 5.3:** Select($\mathcal{L}, S$)

```
1  Initialize a max-heap H ← ∅ and T_m with I(q, U, T_m|S) ← 0
2  i = |S| + 1
3  while i ≤ |L| do
4      T ← L[i]
5      if I(q, U, T) > H.top().Î then
6          Insert ⟨T, Î(q, U, T|S)⟩ into H
7          i ← i + 1
8      else
9          T' ← H.top().T
10         H.pop()
11         if I(q, U, T'|S) > I(q, U, T_m|S) then
12             T_m ← T'
13             if I(q, U, T_m|S) > max(I(q, U, T), H.top().Î) then
14                 return T_m
15 return T_m
```

$\mathcal{I}(q, U, T)$ is smaller than the estimated incremental influence of the top trajectory, do we need to calculate the exact incremental influence $\mathcal{I}(q, U, \mathcal{H}.\text{top}()|S)$ and check if this is a better candidate than the best trajectory $T_{max}$ ever found (lines 9-12). The algorithm can terminate early if $\mathcal{I}(q, U, T_{max}|S)$ is larger than the estimated incremental influence of all the trajectories in $\mathcal{H}$ and the influence of all the unvisited trajectories in $\mathcal{L}$ (lines 13-14).

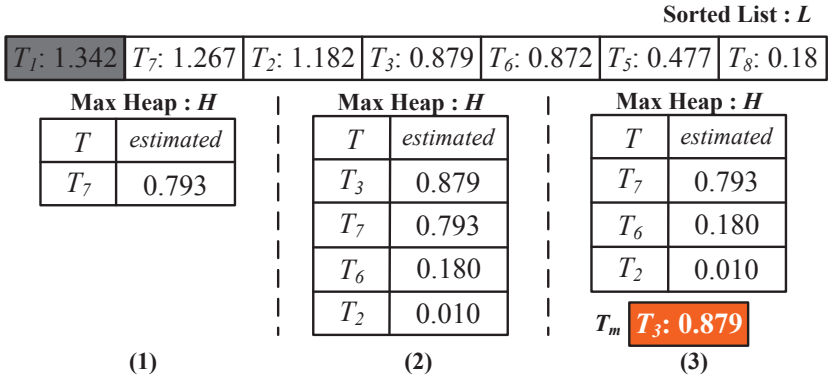



Figure 5.4: baseline greedy method.

**Example 5.3** *Fig. 5.4 shows a snapshot of the greedy algorithm in the* 2*nd iteration. In the* 1*st iteration,* $T_1$ *has the largest influence and is selected as the first seed. (1). In this iteration, we start from* $T_7$*. Since its influence is larger than the estimated incremental influence of the top trajectory (i.e.,* $\emptyset$*) in* $\mathcal{H}$*, we calculate* $\hat{\mathcal{I}}(q, U, T_7|S) =$

*0.793 and insert* $\langle T_7, 0.793\rangle$ *into* $\mathcal{H}$*. (2). Similarly,* $T_2$*,* $T_3$ *and* $T_6$ *are inserted into* $\mathcal{H}$*. (3). Next* $T_5$ *is traversed. Since its influence is smaller than* $\hat{\mathcal{I}}(q, U, T_3|S)$*, we calculate* $\mathcal{I}(q, U, T_3|S) = 0.879$ *and use* $T_3$ *as* $T_{max}$*.* $\mathcal{I}(q, U, T_{max}|S)$ *is larger than the estimated incremental influence of all the trajectories in* $\mathcal{H}$ *and the influence of all the unvisited trajectories in* $\mathcal{L}$*. We can terminate this iteration and select* $T_3$ *as the second seed.*

### 5.5.2 Cluster-based Method

The baseline greedy algorithm has two limitations. First, to find the trajectory $T_{max}$ with the maximum incremental influence in each iteration, it always scans the sorted list $\mathcal{L}$ in the same order. It neglects the fact that the incremental influence $\mathcal{I}(q, U, T|S)$ for a trajectory $T$ varies in different iterations, because a trajectory overlapped with $T$ may be inserted into $S$. Thus, following the same accessing order in different iterations may result in examining more trajectories. To improve the efficiency, we propose a cluster-based algorithm that first partitions the trajectory database into clusters and always accesses the cluster with the maximum estimated incremental influence $\hat{\mathcal{I}}(q, U, T|S)$ in each iteration so that a more promising trajectory $T_m$ can be found earlier. Second, it pushes all the trajectories with an influence larger than $\mathcal{I}(q, U, T_m|S)$ into the heap $\mathcal{H}$. To reduce the number of trajectories pushed into $\mathcal{H}$, we utilize $T_m$ to prune the insignificant trajectories.

#### 5.5.2.1 Trajectory Clustering

In our data model, a trajectory is represented by a sequence of POIs. If two trajectories share more POIs, they have higher chance to influence the same group of audience and demonstrate similar amount of influence increase when calculating the incremental influence. Formally, we define the distance measure used in our trajectory clustering as follows:

**Definition 5.2 (Overlap Distance)**

$$OD(T_i, T_j) = \frac{|T_i| + |T_j| - |T_i \bigcap T_j|}{|T_i| + |T_j|},$$

*where* $|T|$ *is the number of POIs contained in* $T$ *and* $|T_i \bigcap T_j|$ *is the number of matching POIs between* $T_i$ *and* $T_j$*.*

To partition the trajectories into $n$ clusters based on the Overlap Distance, we design a simple clustering method with $n$ fixed seeds. We first sort the trajectories

**Algorithm 5.4:** Cluster-Select($\{\mathcal{L}_i\}, S$)

```
1  Initialize a max-heap H ← ∅
2  Calculate Î(q, U, T_i|S) for the first T_i in each L_i
3  L_m ← the trajectory list with the largest Î(q, U, T_i|S)
4  T_m ← Select(L_m, S)
5  for each of the other trajectory lists L_i do
6  |  Insert ⟨T', Î(q, U, T'|S)⟩ with Î(q, U, T'|S) > I(q, U, T_m|S) to H
7  while I(q, U, T_m|S) < Î(q, U, H.top()|S) do
8  |  T' ← H.pop()
9  |  if I(q, U, T'|S) > I(q, U, T_m|S) then
10 |  |  T_m ← T'
11 return T_m
```

based on the number of potential audience to influence, i.e., the intersection between the audience and the trajectory in the spatial and temporal attributes. We then use $n$ iterations to select $n$ seeds. In the *i-th* iteration, we select the trajectory which can maximize the number of audience influenced by the selected $i$ seeds as the *i-th* seed. Next, each trajectory $T$ is assigned to the cluster whose seed is closest to $T$. An example is shown in Fig. 5.5 which clusters the trajectories in Fig. 4.1 into 3 clusters. We do not use conventional k-means clustering because we found that k-means would generate lots of insignificant clusters in which the trajectories have small influence. For the trajectories in a cluster, we build the same inverted index as in Section 5.3 to store trajectories whose partial influence is not zero for each keyword.

#### 5.5.2.2 Cluster-based Method

We propose a more efficient algorithm after the trajectory clustering and index construction. As discussed, the greedy algorithm has no idea on which trajectory may have a large incremental influence and can only traverse the trajectories in the same order. As a result, a large number of unnecessary trajectories are pushed into the heap $\mathcal{H}$. Thus we define a new order of accessing trajectories. The purpose is to initialize a good $T_m$ for pruning.

Given a query $q$, for each cluster $C_i$, we compute the influence of the trajectories in $C_i$ and build a sorted list $\mathcal{L}_i$. In each iteration, we call the Algorithm 5.4 to get the trajectory with the maximum incremental influence w.r.t. the current advertiser set $S$. As shown in Algorithm 5.4, given an incomplete candidate $S$ with top-$|S|$ trajectories, our algorithm starts by calculating an estimated incremental influence $\hat{\mathcal{I}}(q, U, T_i|S)$, where $T_i$ is the first unvisited trajectory in the sorted list $\mathcal{L}_i$ (line 2). Then, we find the

sorted list $\mathcal{L}_i$ with the maximum $\hat{\mathcal{I}}(q, U, T_i|S)$ and call the function **select** to get the trajectory $T_m$ with the maximum incremental influence in $\mathcal{L}_i$ (line 4). The intuition is that if a trajectory has a high incremental influence w.r.t. $S$, the trajectories belonging to the same cluster may also have a high incremental influence, because they share many identical POIs. For the remaining clusters, we can filter all the trajectories whose $\hat{\mathcal{I}}(q, U, T|S) \leq \mathcal{I}(q, U, T_m|S)$ (lines 5-6). The valid candidates are pushed into a heap $\mathcal{H}$ and accessed in decreasing order of $\hat{\mathcal{I}}(q, U, T|S)$. For each popped trajectory from $\mathcal{H}$, if its estimated incremental influence is smaller than $\mathcal{I}(q, U, T_m|S)$, we calculate the exact incremental influence and update $T_m$ when necessary (lines 7-10). Otherwise, the algorithm can be terminated and $T_m$ is selected as the seed.

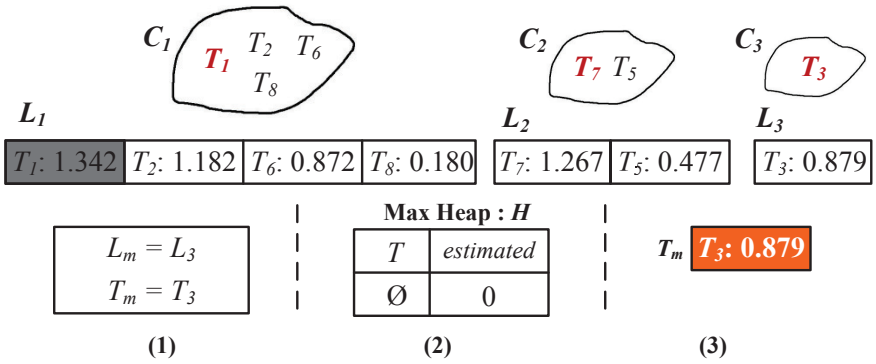



Figure 5.5: cluster-based method.

**Example 5.4** *Fig. 5.5 shows a snapshot of the 2nd iteration of our cluster-based method. (1) In the 2nd iteration, we calculate $\hat{\mathcal{I}}(q, U, T|S)$ for the first trajectory $T$ in each sorted list, and select the trajectory list with the largest $\hat{\mathcal{I}}(q, U, T|S)$ as $\mathcal{L}_m$, i.e., $\mathcal{L}_3$. Next we use the **select** function to get the trajectory $T_m$ that has the largest incremental influence in $\mathcal{L}_3$, i.e., $T_3$. (2) After that, we insert all the trajectories whose $\hat{\mathcal{I}}(q, U, T|S)$ are larger than $\mathcal{I}(q, U, T_3|S)$ into $\mathcal{H}$. In this example, we can see that no trajectory is inserted into $\mathcal{H}$. (3) Since $\mathcal{H}$ is empty, $T_3$ is selected as the second seed. As shown, in this iteration, we access $\mathcal{L}_3$ instead of $\mathcal{L}_1$. The advantage is that we can get a good trajectory $T_3$ with large incremental influence earlier. With the help of $T_3$, $T_7$, $T_6$ and $T_2$ are avoided inserting into $\mathcal{H}$ compared to the greedy method.*

### 5.5.3 Threshold-based Method

The tradeoff between efficiency and accuracy provided by the above algorithms may not be satisfactory in certain applications. The exact algorithms return accurate

top-$k$ results with much longer computation time, while the approximate algorithms can only guarantee a $(1-1/e)$-approximation ratio. To provide a flexible means to adjust the tradeoff between the efficiency and the approximation ratio $\epsilon \in (0,1]$, we propose our threshold-based algorithm. In the exact algorithms (Algorithm 5.1), the termination condition is $max(\overline{UB_0}, \overline{UB'_0}) \leq \mathcal{I}_{opt}$ which requires the current best optimal influence $\mathcal{I}_{opt}$ to be larger than the upper bound of all the candidates in $\mathbb{C}$ and unvisited trajectories. To terminate earlier, we revise the termination condition to be $max(\overline{UB_0} \cdot \epsilon, \overline{UB'_0} \cdot \epsilon) \leq \mathcal{I}_{opt}$ in our threshold-based algorithm and prove that the revised algorithm achieves a $\epsilon$-approximation ratio.

**Lemma 5.7** *The threshold-based method achieves an approximation ratio of $\epsilon$ for any $\epsilon \in (0,1]$.*

**Proof 5.8** *Let $C^+$ and $C^*$ denote top-k trajectories returned by threshold-based algorithm and the accurate algorithm, respectively. We know that $C^*$ is the exact answer and $\mathcal{I}_{opt} = \mathcal{I}(q, U, C^*)$. Suppose the threshold-based method terminates after evaluating a trajectory set $C^+$ such that $\mathcal{I}(q, U, C^+) < \mathcal{I}_{opt} \cdot \epsilon$. In the threshold-based algorithm, since $\overline{UB_0}$ is the upper bound for all the candidates in $\mathbb{C}$ and $C^*$ has not been found, we have $\overline{UB_0} \geq \mathcal{I}_{opt}$. Then, $\overline{UB_0} \cdot \epsilon \geq \mathcal{I}_{opt} \cdot \epsilon > \mathcal{I}(q, U, C^+)$. The termination condition is not satisfied and the threshold-based algorithm will continue to explore more candidates until find one candidate satisfying $max(\overline{UB_0} \cdot \epsilon, \overline{UB'_0} \cdot \epsilon) \leq \mathcal{I}_{opt}$. Thus, our algorithm guarantees $\mathcal{I}(q, U, C^+) \geq \mathcal{I}_{opt} \cdot \epsilon$ when terminated.*

## 5.6 Influence Maximization for Advertisement Group

In this section, we consider how to find $k$ trajectories to maximize the influence for a group of advertisements $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$. For example, a company may have a promotion campaign for multiple products at the same time. We assume that the number of buses $k_i$ to carry each advertisement $q_i$ has been determined and $\sum_{1 \leq i \leq |Q|} k_i = k$. Our goal is to find $k$ trajectories to maximize the total influence $\mathcal{I}(Q, U, S)$ of $Q$, which is defined as follows.

$$\mathcal{I}(Q, U, S) = \sum_{1 \leq i \leq |Q|} \mathcal{I}(q_i, U, S_i) \tag{5.18}$$

where $S_i$ denotes the set of trajectories allocated to $q_i$.

To solve the problem, we cannot directly select the specified number of trajectories for each advertisement $q_i \in Q$ separately by using the previous proposed methods.

The reason is that a trajectory may be selected multiple times but in reality a bus or vehicle cannot hold two advertisements at the same time. A naive method is to find the $k_i$-trajectory set for each advertisement $q_i$ separately, pick the set with the maximum average influence score and mark the trajectories in this set as selected. This process is repeated for the remaining advertisements. After $|Q|$ iterations, we can find a trajectory set for each advertisement. However, this method has to find a set for each advertisement repeatedly and has no performance guarantee. Thus, we propose a group greedy method which selects the trajectories for all the advertisements simultaneously and guarantee an approximation ratio of $(1-1/e)$ by extending our proposed approximate methods (i.e., Greedy and Cluster).

The group greedy method is based on Theorem 5.2 in Section. 5.5.1. There are $k$ iterations in our group greedy method. At each iteration, we select a seed that has the largest incremental influence w.r.t. $S$. After $k$ iterations, we can get a $k$-trajectory set $S$. According to Theorem 5.2, $S$ provides a $(1-1/e)$-approximation ratio.

Next we introduce how to select the seed in each iteration based on the following lemma.

**Lemma 5.8** *Given a trajectory $T_i$ attached with advertisement $q_i$ in $Q$, let $S_i$ denote the trajectory set allocated to $q_i$, we have*

$$\mathcal{I}(q_i, U, T_i|S_i) = \mathcal{I}(Q, U, T_i|S). \tag{5.19}$$

The procedure of the group selection method is as follows. We maintain a max heap $\mathcal{H}$ to access the trajectories in descending order of their incremental influence $\mathcal{I}(q_i, U, T|S_i)$. We also maintain a status for each trajectory to indicate whether it has been selected as a seed. In the first iteration, we select the trajectory $T_i$ with the largest $\mathcal{I}(q_i, U, T_i|S_i)$ for each advertisement $q_i$ separately using **Select** in the greedy method or **Cluster-Select** in the cluster-based method and insert it into $\mathcal{H}$. Then we select the trajectory $T$ on top of $\mathcal{H}$ as the first seed, because $T$ also has the largest $\mathcal{I}(Q, U, T|S)$ according to Lemma 5.8. In the meantime, we mark the status of $T$ as a seed. Suppose $T$ is allocated to $q$, we need to select the next trajectory with the largest incremental influence for $q$ and insert it into $\mathcal{H}$. In the following iterations, we select the trajectory $T_i$ on top of $\mathcal{H}$. If the status of $T_i$ has been marked, we ignore it, select the next trajectory for $q_i$ and insert it into $\mathcal{H}$. Otherwise, $T_i$ is selected as the next seed, its status is marked and the next trajectory for $q_i$ is inserted into $\mathcal{H}$. In addition, if we have selected the specified number of trajectories for $q_i$, we ignore $q_i$ and select the seeds from the remaining advertisements. The process repeats until we have $k$ trajectories with status marked.

## 5.7 Complexity Analysis

In this section, we present the complexity analysis for the trajectory index $\mathcal{N}_t$ and the approximation algorithms.

### 5.7.1 Index Complexity and Update

**Complexity and Update.** Let $K$ denote the total number of tags in $\mathcal{N}_t$ and $|\mathcal{T}_i|$ denote the average number of trajectories in the inverted list under the tag $w_i$. The space complexity of $\mathcal{N}_t$ is $O(\sum_{1\leq i\leq K}|\mathcal{T}_i|)$, where $|\mathcal{T}_i| = |\mathcal{T}|$ in the worst case. In order for efficient update of $\mathcal{N}_t$, we maintain a POI-trajectory index in which each POI $\mathtt{o}$ is followed by the trajectories that pass $\mathtt{o}$ and a POI-audience index in which each each POI $\mathtt{o}$ is followed by the audience that visit $\mathtt{o}$. To update $\mathcal{N}_t$ due to the update of the profile or spatial-temporal patterns of a certain audience $u$, we first find the trajectories $\mathcal{T}_e$ that could influence $u$ using the POI-trajectory index. The complexity is $O(N_u)$, where $N_u$ is the number of POIs contained in the motion patterns of $u$. We then find the partial influence $\mathcal{I}_p(w, U, T)$ for each trajectory $T \in \mathcal{T}_e$ and each tag $w$ of $u$, and update $\mathcal{I}_p(w, U, T)$ by subtracting the original influence of $u$ and adding the new influence of $u$. The complexity is $O(|\mathcal{T}_e|\mathcal{M}_u)$, where $\mathcal{M}_u$ denotes the cost of calculating $\mathcal{I}_p(w, u, T)$. Thus, the total complexity is $O(N_u + |\mathcal{T}_e|\mathcal{M}_u)$. It is similar to update $\mathcal{N}_t$ due to the insertion or deletion of a user. To update $\mathcal{N}_t$ due to the insertion or deletion of a trajectory $T$, we first find the users $U_e$ that could be influenced by $T$ using the POI-audience index. The complexity is $O(N_t)$, where $O(N_t)$ is the number of POIs passed by $T$. Let $\mathtt{t}_e$ denote the set of tags contained in $U_e$. Then for each tag $w$ in $\mathtt{t}_e$, we calculate $\mathcal{I}_p(w, U_e, T)$ and insert $\langle T, \mathcal{I}_p(w, U_e, T)\rangle$ into the inverted list of $w$. The complexity is $O(|\mathtt{t}_e|\mathcal{M}_u)$, where $\mathcal{M}_u$ denotes the cost of calculating $\mathcal{I}_p(w, U_e, T)$. Thus, the total complexity is $O(N_t + |\mathtt{t}_e|\mathcal{M}_u)$

### 5.7.2 Algorithm Complexity Analysis

We first discuss the approximate methods for one single advertisement. The baseline greedy algorithm consists of two major steps: 1) initialization of the sorted list $\mathcal{L}$. 2) $k$ iterations of finding a trajectory with the maximum incremental influence. In the initialization step, we use our inverted lists of partial scores to calculate the influence for each trajectory and sort them according to the value. The cost is $O(m|\mathcal{T}| + |\mathcal{T}|log(|\mathcal{T}|))$, where $m$ is the number of tags in the advertisement. In the following $k$ iterations, we use a heap to help find the trajectory with the maximum incremental influence from a sorted list. In the worst case, the algorithm examines all

the trajectories and the cost is $O(k|\mathcal{T}|\mathcal{M}_e)$, where $\mathcal{M}_e$ denotes the cost of calculating incremental influence $\mathcal{I}(q, U, T|S)$, which is an expensive operation because it needs to scan the affected audience to get the accurate score. The clustering algorithm has the same worst case performance as the baseline algorithm. However, it takes advantage of trajectory similarity and provides a flexible order in accessing the trajectories to find a promising trajectory. With the help of the promising trajectory, a large number of insignificant trajectories can be pruned from the heap. In our experiments, the results show that it can achieve better performance than the baseline greedy algorithm. The complexity analysis of the threshold-based algorithm is not provided because it depends on the value of $\epsilon$. Its performance can be arbitrarily bad when $\epsilon$ is close to 1.

Next, we discuss the analysis of the approximate method for a group of advertisements $Q$. The group greedy algorithm also consists of two major steps: 1) initialization of the sorted list $\mathcal{L}_i$ for each advertisement $q$ in $Q$. The cost is $O(|Q|m|\mathcal{T}| + |Q||\mathcal{T}|log(|\mathcal{T}|))$. 2) $k$ iterations of finding a trajectory with the maximum incremental influence from the $|Q|$ sorted lists. In the worst case, the algorithm examines all the trajectories in the $|Q|$ sorted lists. The cost is $O(\sum_{1\leq i\leq|Q|} k_i|\mathcal{T}|\mathcal{M}_e) = O(k|\mathcal{T}|\mathcal{M}_e)$. The complexity is the same with selecting $k$ trajectories for a single advertisement, which shows the superiority of the group greedy method.

## 5.8 Experimental Study

In this section, we show how to apply the trajectory influence maximization problem to the moving advertisement management field and report results of extensive experiments conducted to evaluate both the efficiency and effectiveness of our proposed methods using real datasets.

**Audience Dataset.** In a modern city, it is common for people to take the bus to their destinations (e.g., workplace, shopping mall or home). They need to wait for a bus at the bus stations and have a high probability to be influenced by the passing vehicles. We modeled such important spatial-temporal patterns of the audience group from the EZLink dataset in Singapore. The Singapore EZLink system works as follows. Each user has an EZLink card with some stored value. When an user boards a bus, he/she will tap on an on-board EZLink device. This essentially records the check-in station and check-in timestamp. Likewise, when the user alights from the bus, he/she will tap on another on-board EZLink device, which records the check-out station and check-out timestamp. The amount corresponding to the fare of

the journey will be deducted from the stored value. Each record of the dataset thus contains the check-in station, check-in timestamp, check-out station and check-out timestamp of a certain audience with a certain bus. Our dataset is provided by the Land Transport Authority company in Singapore. In total, we have three months of EZlink records, which correspond to hundreds of millions of records. To model the spatial-temporal patterns, we represented a POI by a bus station and partitioned a day into 144 time periods (i.e., each time period corresponds to 10 minutes). For each audience $u$, we extracted the records of $u$ and mapped each record into the corresponding bus station and time period. For each bus station $\mathtt{o}$, we calculated the probability that $u$ occurs in $\mathtt{o}$ as $N_o/N_t$, where $N_o$ represents the number of records mapped to $\mathtt{o}$ and $N_t$ represents the total number of records of $u$. In total, we extracted the spatial-temporal patterns for 5 million audience. There are several possible ways to model the textual profile for an audience. One way is to perform an online questionnaire for the audience to collect their interest. The other way is to link an audience in the EZlink dataset with the same user in some online social networks (e.g., Twitter and facebook) where there are rich interest information for the audience by using some profile linking technique. Profile linking is the ability to connect profiles of a user on different social networks and has been studied extensively in recent years [53, 52, 45]. In our experiments, we crawled a large number of geo-tweets in Singapore from Twitter. Then, we modeled the geo-profile of each bus stop by aggregating the tweets nearby. Since the probability of a user visiting a bus stop has been derived, we can aggregate the geo-profiles of the visited bus stops based on the probability to generate the profile for the user.

**Trajectory Dataset.** We constructed two trajectory datasets from bus schedules and GPS logs of taxies in Singapore. The bus trajectory dataset contains $10,000$ trajectories and each trajectory is represented by a sequence of bus stations associated with timestamp. The raw taxi dataset contains logs of two-dimensional geo-coordinates and timestamps reported periodically from the taxies. We need some pre-processing to convert them into a sequence of bus stations. Hence, we define a visual distance threshold to filter those locations in the raw taxi dataset whose distance to the nearest bus stations is larger than the threshold. In total, we extracted 1 million non-empty trajectories, which is large enough for a trajectory database.

**Algorithms.** For the exact methods, we implemented and compared three variants of the expansion-based framework with three different upper bound estimation techniques, denoted by $\texttt{Expansion-UB}_0$ (Eqn. 5.8), $\texttt{Expansion-UB}_1$ (Eqn. 5.14) and

`Expansion-UB`$_2$ (Eqn. 5.15), respectively. For the approximate methods, we compared the greedy method (`Greedy`), the cluster-based method (`Cluster`) and the threshold-based method (`Threshold`). We also compare our group greedy method (`Group`) with the naive method (`Naive`). All the indexes are memory resident and implemented in C++.

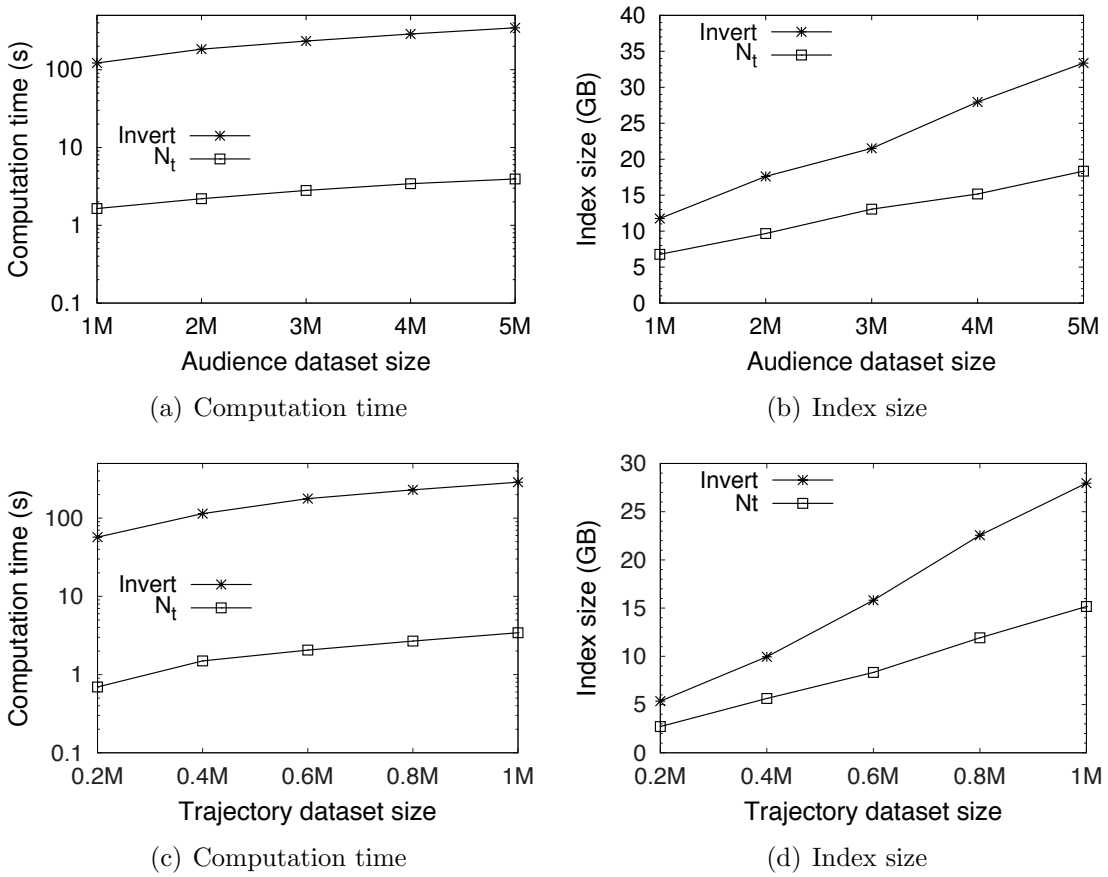
| | | |
|---|---|---|
| Number of tags $m$ | 6, 7, **8**, 9, 10 | |
| Audience dataset size $\|U\|$ | 1M, 2M, 3M, **4M**, 5M | |
| Trajectory dataset size $\|\mathcal{T}\|$ | Bus | 2K, 4K, 6K, 8K, **10K** |
| | Taxi | 0.2M, 0.4M, 0.6M, 0.8M, **1M** |
| Exact Methods | | |
| $k$ | 10, **20**, 30, 40, 50 | |
| Approximate Methods / `Naive` v.s. `Group` | | |
| $k$ | Bus | 100, **200**, 300, 400, 500 |
| | Taxi | 200, **400**, 600, 800, 1000 |
| `Threshold` v.s. `Cluster` | | |
| $k$ | 60, 70, **80**, 90, 100 | |
| `Naive` v.s. `Group` | | |
| Number of advertisements $\|Q\|$ | **10**, 20, 30, 40, 50 | |

Table 5.2: Parameters evaluated in the experiments

**Parameters and Metrics.** Table 5.2 shows the main parameters and values used throughout the experiments with default values in bold. We tested larger $k$ values for the approximate methods than the exact methods to show the scalability to $k$. For `Cluster`, we generated 50 clusters for evaluation. For `Threshold`, we evaluated the performance when $\epsilon$ is set to $1 - 1/e$, 0.8 and 0.9. As for the methods for a group of advertisements $Q$, we randomly specify the number of trajectories for each advertisement in $Q$. In the experiments, we measure the following metrics: (i) *efficiency*, the amount of time an algorithm runs to process a top-$k$ query; (ii) *influence score*, the expected influence obtained by the returned top-$k$ trajectories for an algorithm.

### 5.8.1 Evaluation on Indexes

In the initialization steps of all the algorithms, we need to calculate the influence for each trajectory in the trajectory dataset. There are two types of indexes that can be built to accelerate the calculation, the naive inverted index (`Invert`) and $\mathcal{N}_t$ (see Section 5.3.3). In this section, we compare `Invert` and $\mathcal{N}_t$ in terms of index sizes and trajectory influence computation time on the taxi dataset which contains




(a) Computation time

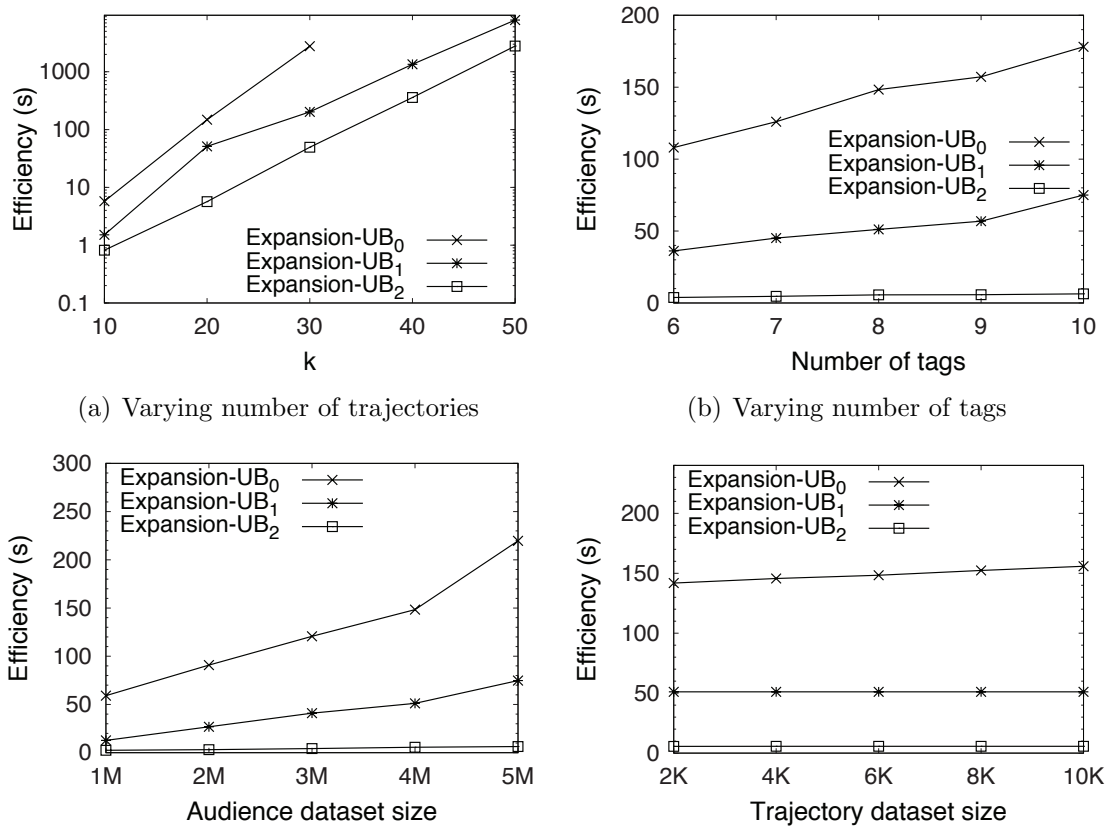
(b) Index size

(c) Computation time

(d) Index size

Figure 5.6: Evauation on the indexes.

1 million trajectories. The results are shown in Fig. 5.6. We can see that $\mathcal{N}_t$ beats `Invert` with better efficiency (orders of magnitude better) and smaller index size (50% smaller). This is due to the advantage of the partial influence $\mathcal{I}_p(w, U, T)$ proposed in Section 5.3.3. For the index size, $\mathcal{N}_t$ only needs to store the trajectories with their corresponding partial influence. However, `Invert` needs to store all the audience whose number is much larger than that of the trajectories. For the computation time, $\mathcal{N}_t$ only needs to perform simple linear computation for each trajectory according to Eqn. 5.12. However, `Invert` needs to traverse the corresponding users to calculate the influence according to Eqn. 5.7. In addition, $\mathcal{N}_t$ scales much better than `Invert` when the audience dataset or the trajectory dataset size increases with the help of the partial influence. In the following experiments, all the methods utilize $\mathcal{N}_t$ to calculate the influence for each trajectory.




(a) Varying number of trajectories

(b) Varying number of tags

(c) Varying audience dataset size

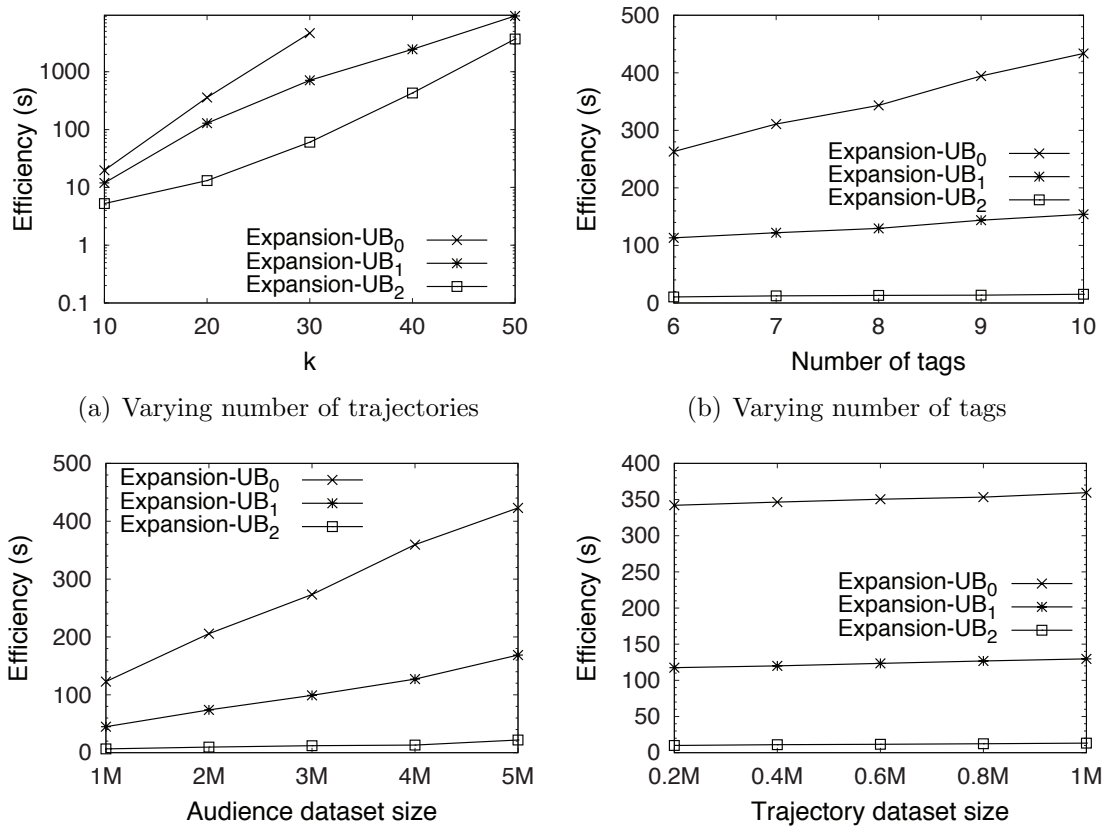
(d) Varying trajectory dataset size

Figure 5.7: Accurate methods on the bus dataset.

### 5.8.2 Evaluation on Accurate Methods

In our first set of experiments, we compare our proposed expansion-based framework with three different upper bounds $UB_0$, $UB_1$ and $UB_2$. The expansion-based framework can guarantee to find the accurate top-$k$ trajectories with the largest influence score.

**Effect of $k$.** Fig. 5.7(a) and Fig. 5.8(a) show the impact of $k$ in the top-$k$ query. We have the following observations. First, `Expansion-UB`$_0$ achieves the worst performance due to the rather loose upper bound $UB_0$. Even though $UB_0$ can be calculated efficiently, `Expansion-UB`$_0$ would generate a large number of incomplete candidates in $\mathbb{C}$ and need to calculate the influence for each incomplete candidates, resulting in a large computation cost. As shown in the figures, it takes hundreds of seconds to return a top-20 trajectories using `Expansion-UB`$_0$. It takes much more time when $k$ is larger than 30 and we did not report their results. Second, `Expansion-UB`$_1$ achieves much better performance than `Expansion-UB`$_0$. Even though $UB_1$ is computationally expensive, `Expansion-UB`$_1$ can provide a rather tight upper bound for each incomplete can-




(a) Varying number of trajectories

(b) Varying number of tags

(c) Varying audience dataset size

(d) Varying trajectory dataset size

Figure 5.8: Accurate methods on the taxi dataset.

didate and prune much more insignificant candidates than $\texttt{Expansion-UB}_0$, reducing the cost of calculating the influence of the incomplete candidates. Third, $\texttt{Expansion-UB}_2$ achieves the best performance. This is because $\texttt{Expansion-UB}_2$ can estimate a rather tight upper bound $\hat{\mathcal{I}}(q, U, T|C)$ for the incremental influence $\mathcal{I}(q, U, T|C)$ of a trajectory $T$ w.r.t. an incomplete candidate $C$ by considering the overlaps between $T$ and $C$. In addition, $\hat{\mathcal{I}}(q, U, T|C)$ can be calculated efficiently by precomputing the partial influence for each POI. As shown in the figures, $\texttt{Expansion-UB}_2$ achieves 10X better performance than the other two methods. It takes only about 5 seconds to find the top-20 trajectories for the bus dataset and about 10 seconds for the taxi dataset. However, we can also see that the accurate methods do not scale well w.r.t. $k$ because the number of incomplete candidates grows exponentially with $k$. Thus, we propose three approximate methods with performance gurantees to support the case when $k$ is large, which will be evaluated in Section 5.8.3.

**Effect of the number of tags $m$.** Fig. 5.7(b) and Fig. 5.8(b) depict the effect of the number of tags $m$ in the advertisement. As shown, the performance of all the

methods degrade when $m$ increases. This is because in our expansion-based method, we iteratively enumerate all the promising candidates. For each newly generated candidate, we need to calculate the influence score on the fly. We also need to calculate the influence score for a large number of incomplete candidates when calculating $UB_1$ for `Expansion-UB`$_1$. Such calculation is a frequent operator and increasing $m$ would make the operation more expensive because we need to scan more relevant users in order to calculate the exact influence score. `Expansion-UB`$_0$ scales worst in terms of increasing $m$ because it examines much more trajectories and generates much more candidates than the other two methods, leading to higher computation cost in total. In contrast, `Expansion-UB`$_2$ is more scalable w.r.t. to $m$ because it estimates a tight upper bound which prunes a large number of incomplete candidates from $\mathbb{C}$ and uses $\hat{\mathcal{I}}(q, U, T|C)$ to calculate $UB_2$ which does not involve any candidate influence computation.

**Effect of the audience dataset size $|U|$.** Next, we evaluate the effect of the audience dataset size $|U|$. The results are shown in Fig. 5.7(c) and Fig. 5.8(c). As expected, the running time increases as $|U|$ becomes larger. This is because it takes more time to scan the relevant audience to calculate the influence score for incomplete candidates. In other words, it takes the same effect as increasing the number of query tags $m$ in an advertisement. Still, `Expansion-UB`$_2$ achieves the best performance.

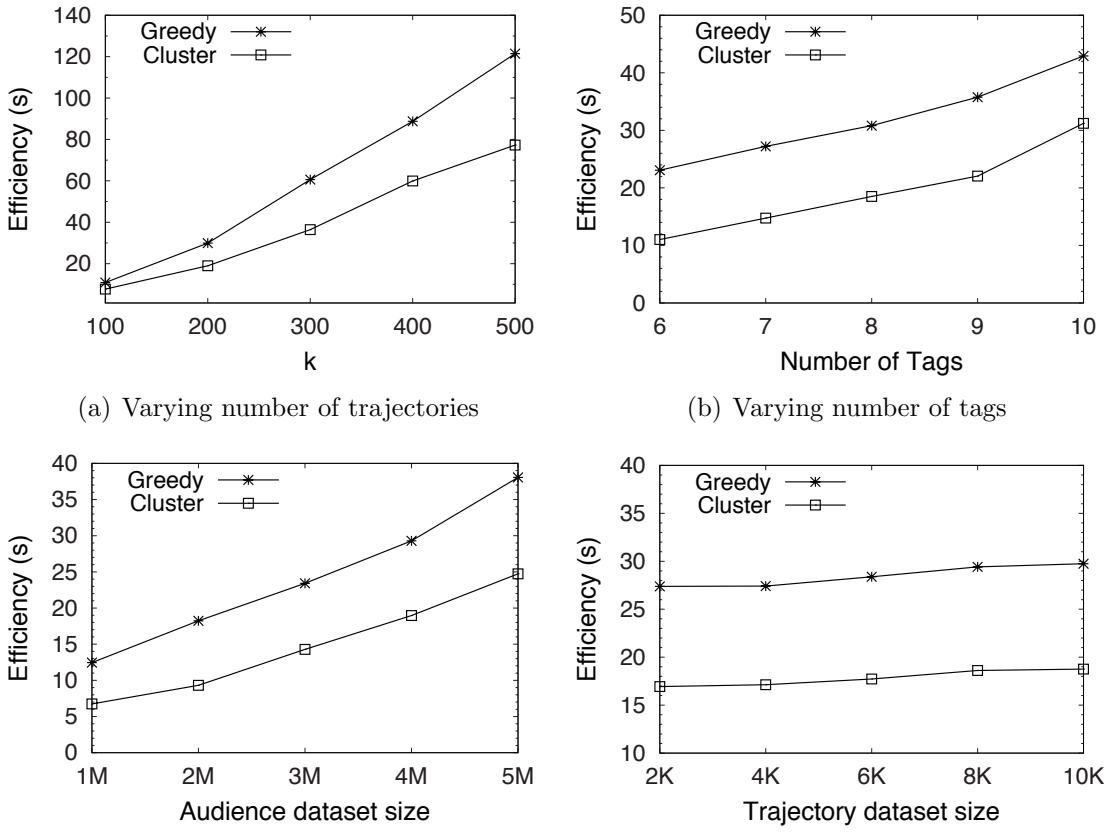
**Effect of the trajectory dataset size $|\mathcal{T}|$.** Lastly, we evaluate the efficiency when increasing the trajectory dataset size $|\mathcal{T}|$. The results are shown in Fig. 5.7(d) and Fig. 5.8(d). We can see that all the methods only increase slightly when $|\mathcal{T}|$ increases. The reason for such insensitivity is that our expansion-based method enumerates the trajectory in a best-first manner. With the help of our proposed upper bounds, the algorithm can be terminated early without enumerating all the trajectories. As a result, only a small part of the trajectories with high influence are traversed. In addition, with the help of our proposed index $\mathcal{N}_t$, the cost of calculating the influence of a trajectory is not sensitive to $|\mathcal{T}|$ as well.

### 5.8.3 Evaluation on Approximate Methods

For the approximate methods, we report both the running time of an advertisement and the expected influence score to measure both the efficiency and effectiveness.

#### 5.8.3.1 Methods with (1-1/e) approximation ratio

We first compare the efficiency of two approximate methods (i.e., `Greedy` and `Cluster`) that return the same top-$k$ trajectories and achieve the same approximation ratio.




(a) Varying number of trajectories

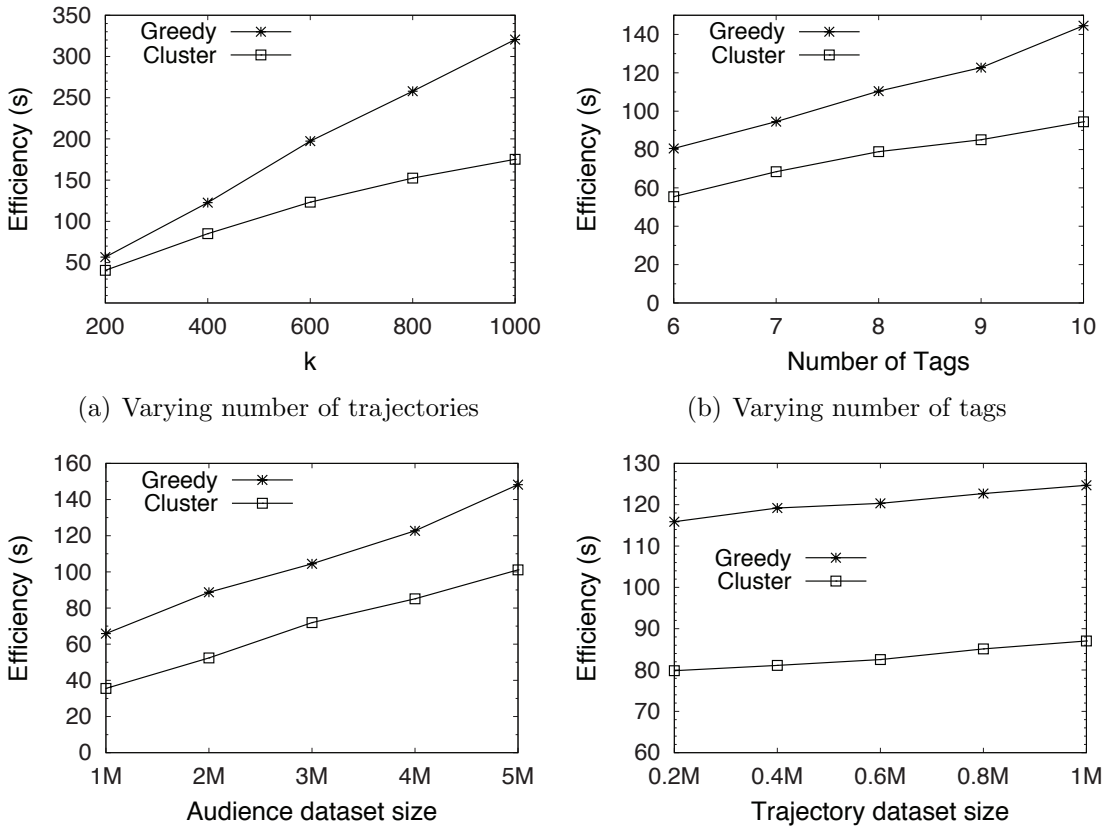
(b) Varying number of tags

(c) Varying audience dataset size

(d) Varying trajectory dataset size

Figure 5.9: Approximate methods with (1-1/e) apprimation ratio on the bus dataset.

**Effect of $k$.** Fig. 5.9(a) and Fig. 5.10(a) present the impact of $k$. We have the following observations. First, `Greedy` performs much more efficiently w.r.t. $k$ compared with the above accurate methods, because it only needs $k$ iterations to find the top-$k$ trajectory set. In each iteration, it adopts a best-first manner to traverse the trajectories and utilizes the upper bound $\hat{\mathcal{I}}(q, U, T|S)$ to avoid calculating the incremental influence for each trajectory pushed into the heap. It takes thousands of seconds to answer a top-50 query for exact methods in Fig 5.7(a) and Fig 5.8(a), but `Greedy` requires less than 50 seconds to answer a top-100 query. Second, `Cluster` achieves much better performance than `Greedy`. This is attributed to its pre-knowledge about the promising trajectory, which is obtained by clustering the trajectories first and accessing the clusters according to a different order related to the incremental influence. With the help of the promising trajectory, it can prune a large number of insignificant trajectories and only push a small number of trajectories into the heap. As shown in the figures, the superiority of `Cluster` compared with `Greedy` is more obvious with a larger $k$. It takes about 10 seconds to return top-100 trajectories.




(a) Varying number of trajectories

(b) Varying number of tags

(c) Varying audience dataset size

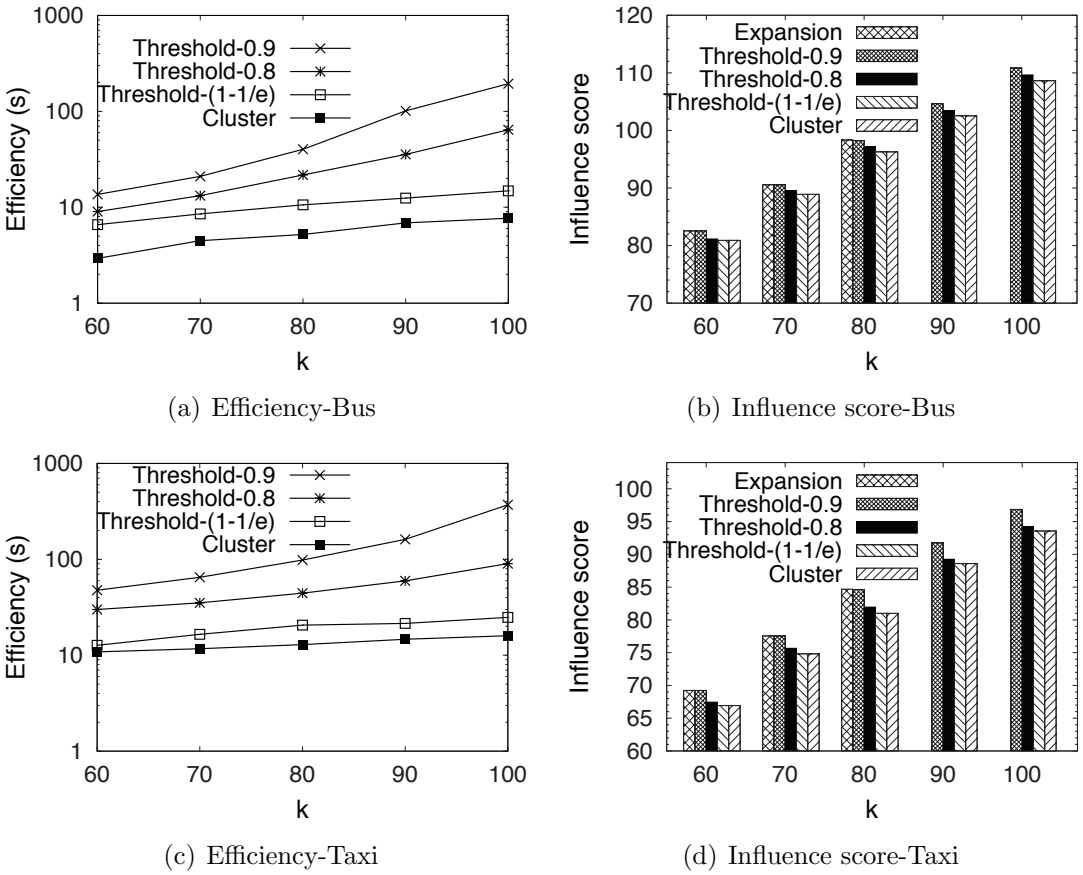
(d) Varying trajectory dataset size

Figure 5.10: Approximate methods with (1-1/e) approximation ratio on the taxi dataset.

**Effect of the number of tags $m$.** Fig. 5.9(b) and Fig. 5.10(b) depict the effect of the number of tags $m$ in the query. We can see that the computation cost of `Greedy` and `Cluster` increases when $m$ increases. This is because more users are relevant with the advertisement and it needs more time to calculate the incremental influence.

**Effect of the audience dataset size $|U|$.** Next, we evaluate the effect of the audience dataset size $|U|$. The results are shown in Fig. 5.9(c) and Fig. 5.10(c). $|U|$ has a similar impact to the case of increasing $m$. A larger $|U|$ results in more users involved in an advertisement and increases the computation cost.

**Effect of the trajectory dataset size $|\mathcal{T}|$.** Lastly, we evaluate the efficiency when increasing the trajectory dataset size $|\mathcal{T}|$. The results are shown in Fig. 5.9(d) and Fig. 5.10(c). We can see that all the methods are not sensitive to $|\mathcal{T}|$. This is because both `Greedy` and `Cluster` enumerate the trajectory in a best-first manner until they find a trajectory with the largest incremental influence w.r.t. $S$ in each




(a) Efficiency-Bus

(b) Influence score-Bus

(c) Efficiency-Taxi

(d) Influence score-Taxi

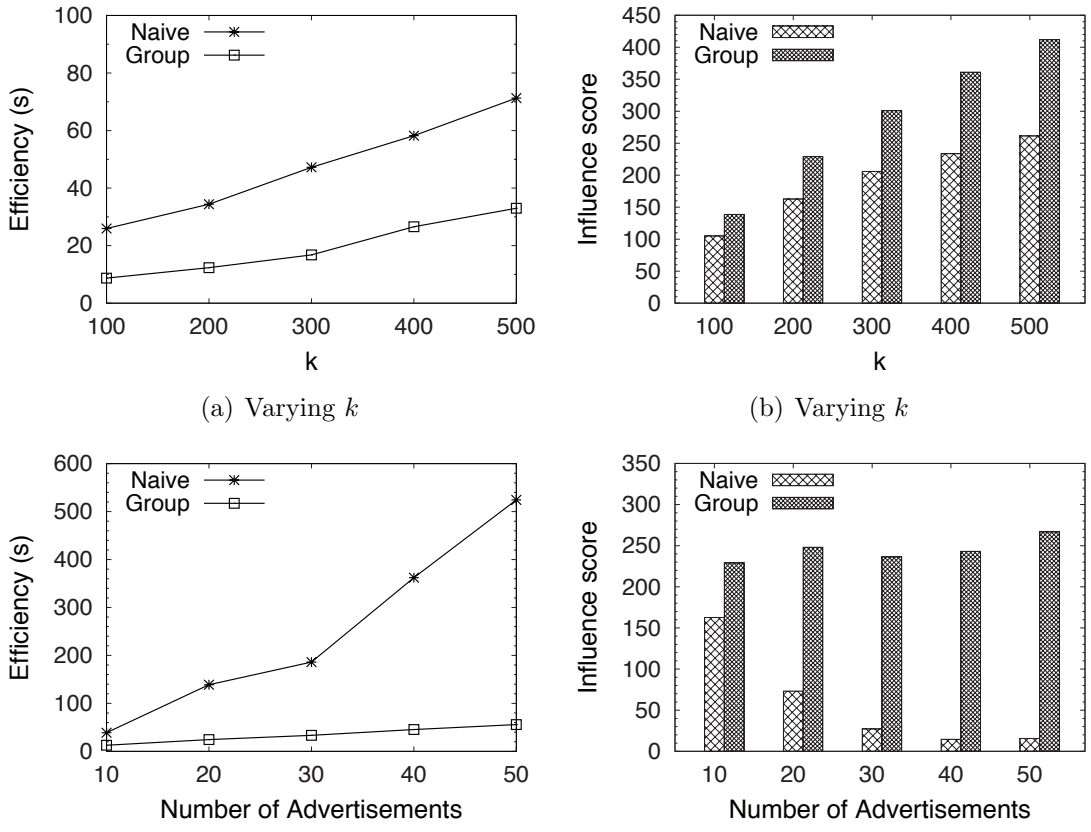
Figure 5.11: Comparison between Threshold and Cluster.

iteration. As a result, only a small part of trajectories in the front of the sorted list would be traversed.

#### 5.8.3.2 Comparison between Threshold and Cluster

In this section, we compare `Threhold` with `Cluster` on efficiency and influence score to see the influence of $\epsilon$. We did not compare with `Greedy` because `Cluster` finds the same results as `Greedy` and achieves better performance. For better comparison, we also report the influence score of `Expansion` when $k$ is smaller than 90. The efficiency of `Expansion` was not reported because it took very long time for `Expansion` when $k$ is large.

Fig. 5.11 shows the efficiency and influence score of the methods. We have the following observations. (1) When $\epsilon$ decreases from 0.9 to $1 - 1/e$, the running time of `Threshold` drops dramatically but the expected influence also decreases, leading to less accurate results. (2) `Threshold` demonstrates a good tradeoff between efficiency and accuracy. When $\epsilon$ is set to 0.9, it achieves almost the same influence score with




(a) Varying $k$

(b) Varying $k$

(c) Varying $|Q|$

(d) Varying $|Q|$

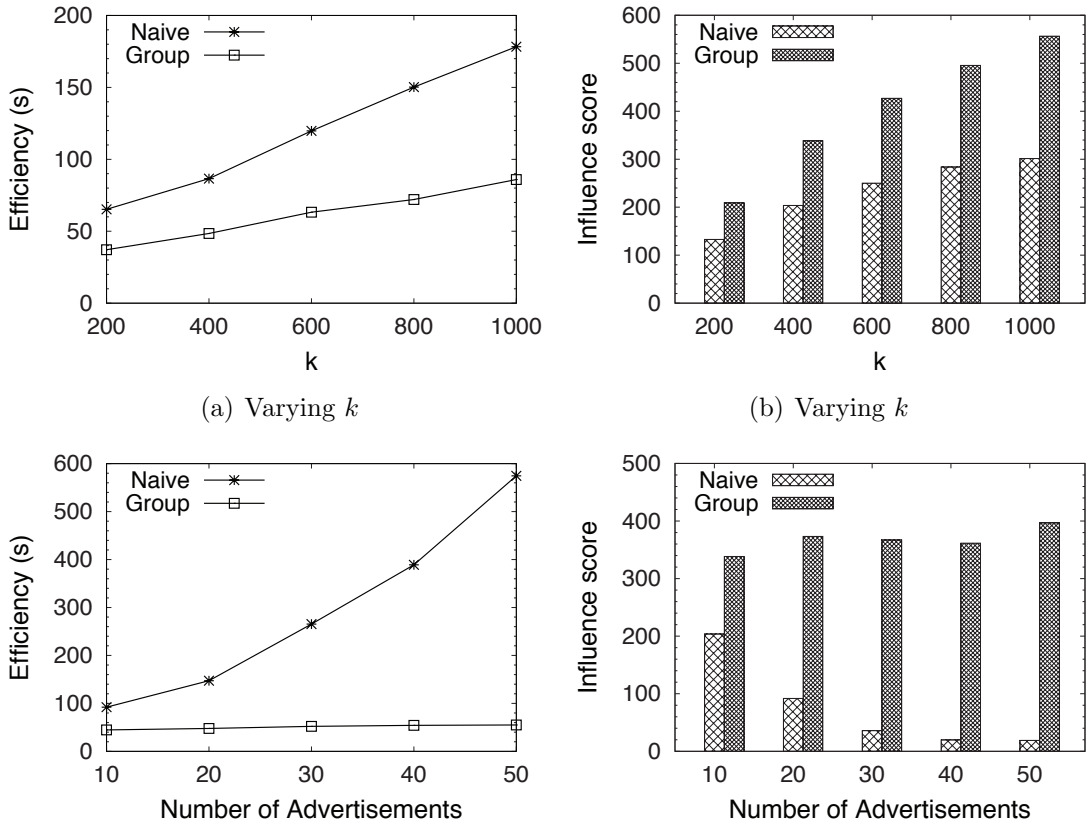
Figure 5.12: Methods for advertisement group on the bus dataset.

the exact method but the computation cost is reduced by more than two orders of magnitude. (3) When the approximation ratio is set to $(1-1/e)$, the influence scores of `Cluster` and `Threshold` are very close. However, it takes much less time for the `Cluster` to answer a query.

Based on the above observations, we recommend to use `Cluster` method when an approximation ratio of $(1-1/e)$ has met the requirement of applications. Otherwise, `Threshold` is recommended because it achieves a good tradeoff between efficiency and accuracy. When $\epsilon$ is set to be close to 1, it returns results similar to the exact solutions but with much less computation cost.

#### 5.8.3.3 Methods for Advertisement Group

We compare the efficiency and effectiveness of `Naive` and `Group`. The results are shown in Fig. 5.12 and Fig. 5.13. We have the following observations. (1) In terms of efficiency, `Group` performs much better than `Naive`. This is because `Group` selects the trajectory for the advertisements simultaneously to avoid repetitive processing. (2) In




(a) Varying $k$ (b) Varying $k$

(c) Varying $|Q|$ (d) Varying $|Q|$

Figure 5.13: Methods for advertisement group on the taxi dataset.

terms of effectiveness, `Group` can find a $k$-trajectory set with a higher influence score than that found by `Naive`, because `Group` can guarantee an approximation ratio of $(1 - 1/e)$ while `Naive` does not provide any performance guarantee. (3) `Group` scales much better than `Naive` w.r.t. the number of advertisements. As shown in Fig. 5.12(c) and Fig. 5.12(d), `Group` have a 10X better performance than `Naive` when $|Q| = 50$.

## 5.9 Summary

In this work, we formulate the trajectory influence maximization problem and prove it is NP-hard. To calculate the accurate results efficiently, we devise an expansion-based framework that enumerates the trajectory in a best-first manner. In addition, we propose two effective methods for the upper bound estimation. To support the trajectory influence maximization problem with large $k$, we propose three approximate methods with performance guarantees. Experimental results on real datasets show that our methods can solve the trajectory influence maximization problem efficiently.

# Chapter 6

# Conclusion and Future Works

## 6.1 Conclusions

With the rapid development of GPS-enabled devices and the explosive increase of geo-tagged information, new challenges have rise in location-based services where large-scale real-time locations of moving objects can be recorded and collected. It is challenging to devise efficient and effective algorithms to process moving location-based queries and extract valuable information buried in the trajectory data representing the mobility of moving objects. This thesis addresses these challenges by solving three types of problems: the moving spatial keyword queries on road networks, the moving spatial queries against dynamic event streams and the optimal trajectories queries for influence maximization.

The first problem deals with moving spatial keyword queries on road networks, where top $k$ ranked objects satisfying a moving query on a road network are continuously returned. We propose two efficient methods for query processing. QCA monitors the top-$k$ results of the moving point by examining the intersections the query encounters. It uses expansion tree to avoid repetitive traversing of some network edges. OCA incrementally retrieves the top-$k$ results according to textual relevance first and computes all or partial top-$k$ results of a subset of nodes on the network. It constructs a shortest path tree to facilitate subsequent processing. We compare the proposed methods with three baseline methods. Experimental results confirm the superiority of our two methods and revealed their relative advantages.

The second problem deals with moving spatial queries against dynamic event streams. In this problem, the server continuously monitors moving users subscribing to dynamic event streams, and notifies users instantly when there is a matching event nearby. We propose a new location-aware pub/sub system named Elaps to support continuous moving spatial queries against dynamic event streams. We exploit how to

use safe region together with a novel concept name impact region to reduce the communication cost. Based on a novel cost model, we propose two incremental methods to construct the safe region and impact region. To reduce the response time of Elaps, we propose a novel index BEQ-Tree which can support efficient spatial subscription matching over a collection of events in the dynamic event environment. Experimental results on real datasets show that Elaps can greatly reduce the communication overhead and disseminate events to users in real-time.

The last problem deals with optimal trajectories queries for influence maximization, which finds top $k$ trajectories to maximize the influence of an advertisement among a large number of audience. We formulate the influence maximization problem in trajectory databases and prove it is NP-hard. To calculate the accurate results efficiently, we devise an expansion-based framework that enumerates the trajectory in a best-first manner and proposed two effective methods for the upper bound estimation. To support the problem with large $k$, we propose three approximate methods with performance guarantees. In addition, we extend the problem to find $k$ best trajectories for a group of advertisements. Experimental results on real datasets show that our methods can solve the trajectory influence maximization problem efficiently.

## 6.2 Future Work

There are several directions that we would like to work on in the future.

In Chapter 3, we drive a safe segment within one edge to reduce the communication cost between the server and the clients. As long as a user stays within the safe segment, there is no need to communicate with the server. It should be noted that a limitation exists in this study. If we can build a safe segment crossing several edges, the user can stay within the safe segment with a longer time, resulting in less communication cost. A naive solution to solve this limitation is to find the top-$k$ objects for each vertex near the user and locate the points along the edges where the top-$k$ results of the query would change when the user overpasses the points. However, the naive method sacrifices the computation cost to reduce the communication cost. Future work should be conducted to study how to find the safe segment crossing several edges more efficiently.

In Chapter 4, while our system can monitor the moving subscribers efficiently, the publisher in our system are static. Future work is needed to process a more sophisticated scenario where both the subscribers and publishers are moving. In this case, a subscriber can be a publisher at the same time. For instance, two moving

users with similar interest can be notified when they stay close to each other. This problem can find many applications such as advertising and recommending friends. A naive solution for this problem is to build a safe region and impact region for each user and check whether there is a match when the user exits the safe region. However, the naive solution still suffers from the problem of communicating with the server frequently. This is because the publishers are also moving, resulting in the need to update the safe region and impact region frequently. Thus, future work is needed to consider how to maintain the safe region and impact region effectively to reduce the communication overhead. Another direction that can be worked on is to extend the problem in Chapter 4 to support the scenario where the subscribers and publishers are constrained on a road network. To solve this new problem, we can still adopt the idea of safe region and impact region to reduce the communication cost, and also construct a cost model to balance the tradeoff between safe region size and communication cost. However, future work is needed to redesign the methods to construct the cost model and safe region due to the constraint of the underlying road networks.

In Chapter 5, the spatial-temporal patterns of a user is represented by the likelihood to visit a POI in certain time period. Future work can extend the problem setting to represent the motion patterns by replacing a POI to a spatial region. Under the new problem setting, we can adopt some spatial indexes for efficient pruning when finding influenced audience w.r.t a trajectory or finding trajectories that can influence an audience.

The advances in location positioning and wireless communication technologies have led to a myriad of user-generated spatial trajectories. One promising direction is to utilize rich information about user behavior, interests, and preferences buried in these trajectories to further improve location-based services. For instance, we can understand the similarity between two different users with user-generated trajectories, thereby providing a user with personalized services and enabling friend recommendation and community discovery. We can also extract the correlations between two different locations based on upon the information from users, thereby offering users better personalized travel recommendations.

# Bibliography


[1] Robert Aboolian, Oded Berman, and Dmitry Krass. Efficient solution approaches for a discrete multi-facility competitive interaction model. *Annals of Operations Research*, 167(1):297–306, 2009.

[2] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. Vector-space ranking with effective early termination. In *SIGIR*, pages 35–42, 2001.

[3] Gennady Antoshenkov and Mohamed Ziauddin. Query processing and optimization in oracle rdb. *The VLDB Journal*, 5(4):229–237, 1996.

[4] Cigdem Aslay, Nicola Barbieri, Francesco Bonchi, and Ricardo A. Baeza-Yates. Online topic-aware influence maximization queries. In *EDBT*, pages 295–306, 2014.

[5] Bhuvan Bamba, Ling Liu, Arun Iyengar, and Philip S Yu. Safe region techniques for fast spatial alarm evaluation. *Georgia Institute of Technology*, 2008.

[6] Jie Bao, M.F. Mokbel, and Chi-Yin Chow. Geofeed: A location aware news feed system. In *ICDE*, pages 54–65, 2012.

[7] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. In *ICDM*, pages 81–90, 2012.

[8] Oded Berman, Dmitry Krass, and ChenWei Xu. Locating flow-intercepting facilities: New approaches and results. *Annals of Operations Research*, 60(1):121–143, 1995.

[9] F. Bonchi. Influence propagation in social networks: A data mining perspective. In *WI-IAT*, volume 1, pages 2–2, 2011.

[10] Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[11] Sergio Cabello, J. Miguel Diaz-Banez, Stefan Langerman, Carlos Seara, and Inma Ventura. Reverse facility location problems. In *CCCG*, pages 68–71, 2005.

[12] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. Continuous monitoring of distance-based range queries. *IEEE Trans. Knowl. Data Eng.*, 23(8):1182–1199, 2011.

[13] Lisi Chen, Gao Cong, and Xin Cao. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*, pages 749–760, 2013.

[14] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: an experimental evaluation. In *VLDB*, pages 217–228, 2013.

[15] Shuo Chen, Ju Fan, Guoliang Li, Jianhua Feng, Kian-lee Tan, and Jinhui Tang. Online topic-aware influence maximization. *PVLDB*, 8(6):666–677, 2015.

[16] Wei Chen, Tian Lin, and Cheng Yang. Efficient topic-aware influence maximization using preprocessing. *CoRR*, 2014.

[17] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.

[18] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[19] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, pages 88–97, 2010.

[20] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD*, pages 591–602, 2009.

[21] Zitong Chen, Yubao Liu, Raymond Chi-Wing Wong, Jiamin Xiong, Ganglin Mai, and Cheng Long. Efficient algorithms for optimal location queries in road networks. In *SIGMOD*, pages 123–134, 2014.

[22] Hyung-Ju Cho and Chin-Wan Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, pages 865–876, 2005.

[23] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[24] Gianpaolo Cugola and Alessandro Margara. High-performance location-aware publish-subscribe on gpus. In *Middleware*, pages 312–331, 2012.

[25] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[26] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, pages 115–126, 2001.

[27] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[28] Hakan Ferhatosmanoglu, Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Constrained nearest neighbor queries. In *SSTD*, pages 257–278, 2001.

[29] R.A. Finkel and J.L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[30] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1):73–84, 2011.

[31] Long Guo, Lu Chen, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, and Zhifeng Bao. Elaps: An efficient location-aware pub/sub system. In *ICDE 2015*, pages 1504–1507, 2015.

[32] Long Guo, Jie Shao, HtooHtet Aung, and Kian-Lee Tan. Efficient continuous top-k spatial keyword queries on road networks. *GeoInformatica*, 19(1):29–60, 2015.

[33] Long Guo, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, and Zhifeng Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD 2015*, pages 843–857, 2015.

[34] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[35] Mahady Hasan, Muhammad Aamir Cheema, Xuemin Lin, and Ying Zhang. Efficient construction of safe regions for moving knn queries over dynamic datasets. In *SSTD*, pages 373–379, 2009.

[36] Haibo Hu, Jianliang Xu, and Dik Lun Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*, pages 479–490, 2005.

[37] Weihuang Huang, Guoliang Li, Kian-Lee Tan, and Jianhua Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.

[38] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *TODS*, 30(2):364–397, 2005.

[39] Qingye Jiang, Guojie Song, Gao Cong, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated annealing based influence maximization in social networks. In *AAAI*, 2011.

[40] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[41] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *PKDD*, pages 259–271, 2006.

[42] Mohammad R. Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[43] Mohammad R. Kolahdouzan and Cyrus Shahabi. Alternative solutions for continuous k nearest neighbor queries in spatial network databases. *GeoInformatica*, 9(4):321–341, 2005.

[44] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[45] Nitish Korula and Silvio Lattanzi. An efficient reconciliation algorithm for social networks. *PVLDB*, 7(5):377–388, 2014.

[46] Takeshi Kurashima, Tomoharu Iwata, Go Irie, and Ko Fujimura. Travel route recommendation using geotags in photo sharing sites. In *CIKM*, pages 579–588, 2010.

[47] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[48] Chuanwen Li, Yu Gu, Jianzhong Qi, Ge Yu, Rui Zhang, and Wang Yi. Processing moving knn queries using influential neighbor sets. *PVLDB*, 8(2):113–124, 2014.

[49] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-lee Tan, and Wen-syan Li. Efficient location-aware influence maximization. In *SIGMOD*, pages 87–98, 2014.

[50] Guoliang Li, Yang Wang, Ting Wang, and Jianhua Feng. Location-aware publish/subscribe. In *KDD*, pages 802–810, 2013.

[51] Xiaohui Li, Vaida Čeikute, Christian S. Jensen, and Kian-Lee Tan. Trajectory based optimal segment computation in road network databases. In *SIGSPATIAL*, pages 396–399, 2013.

[52] Jing Liu, Fan Zhang, Xinying Song, Young-In Song, Chin-Yew Lin, and Hsiao-Wuen Hon. What's in a name?: An unsupervised approach to link users across communities. In *WSDM*, pages 495–504, 2013.

[53] Anshu Malhotra, Luam Totti, Wagner Meira Jr., Ponnurangam Kumaraguru, and Virgilio Almeida. Studying user footprints in different online social networks. In *ASONAM*, pages 1065–1070, 2012.

[54] Sarana Nutanong, Egemen Tanin, Jie Shao, Rui Zhang, and Kotagiri Ramamohanarao. Continuous detour queries in spatial networks. *IEEE Trans. Knowl. Data Eng.*, 24(7):1201–1215, 2012.

[55] Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. The v*-diagram: a query-dependent approach to moving knn queries. *PVLDB*, 1(1):1095–1106, 2008.

[56] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Ltd, Chichester, second edition, 2000.

[57] Atsuyuki Okabe, Toshiaki Satoh, T. Furuta, A. Suzuki, and K. Okano. Generalized network voronoi diagrams: Concepts, computational methods, and applications. *International Journal of Geographical Information Science*, 22(9):965–994, 2008.

[58] Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.

[59] Frank Ramsak, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, and Rudolf Bayer. Integrating the ub-tree into a database system kernel. In *VLDB*, pages 263–272, 2000.

[60] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.

[61] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222, 2011.

[62] João B. Rocha-Junior and Kjetil Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, pages 168–179, 2012.

[63] Nick Roussopoulos, Stephen Kelley, and Frdic Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[64] Mohammad Sadoghi and Hans-Arno Jacobsen. Be-tree: An index structure to efficiently match boolean expressions over high-dimensional discrete space. In *SIGMOD*, pages 637–648, 2011.

[65] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[66] Narushige Shiode, Chao Li, Michael Batty, Paul Longley, and David Maguire. The impact and penetration of location-based services. In *Telegeoinformatics*, 2004.

[67] Stefan Steiniger, Moritz Neun, and Alistair Edwardes. Foundations of location based services.

[68] Han Su, Kai Zheng, Jiamin Huang, Hoyoung Jeung, Lei Chen, and Xiaofang Zhou. Crowdplanner: A crowd-based route recommendation system. In *ICDE*, pages 1144–1155, 2014.

[69] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD*, pages 75–86, 2014.

[70] Henan Wang, Guoliang Li, Huiqi Hu, Shuo Chen, Bingwen Shen, Hao Wu, Wen-Syan Li, and Kian-Lee Tan. R3: A real-time route recommendation system. *PVLDB*, 7(13):1549–1552, 2014.

[71] Steven Euijong Whang, Hector Garcia-Molina, Chad Brower, Jayavel Shanmugasundaram, Sergei Vassilvitskii, Erik Vee, and Ramana Yerneni. Indexing boolean expressions. *PVLDB*, 2(1):37–48, 2009.

[72] Raymond Chi-Wing Wong, M. Tamer Özsu, Ada Wai-Chee Fu, Philip S. Yu, Lian Liu, and Yubao Liu. Maximizing bichromatic reverse nearest neighbor for lp-norm in two- and three-dimensional spaces. *The VLDB Journal*, 20(6):893–919, 2011.

[73] Raymond Chi-Wing Wong, M. Tamer Özsu, Philip S. Yu, Ada Wai-Chee Fu, and Lian Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009.

[74] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. Joint top-k spatial keyword query processing. *IEEE Trans. Knowl. Data Eng.*, 24(10):1889–1903, 2012.

[75] Dingming Wu, Man Lung Yiu, Christian S. Jensen, and Gao Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[76] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.*, 31(1):1–38, 2006.

[77] Wei Wu, Wenyuan Guo, and Kian-Lee Tan. Distributed processing of moving k-nearest-neighbor query on moving objects. In *ICDE*, pages 1116–1125, 2007.

[78] Wenjian Xu, Chi-Yin Chow, Man Lung Yiu, Qing Li, and Chung Keung Poon. Mobifeed: A location-aware news feed system for mobile users. In *SIGSPATIAL*, pages 538–541, 2012.

[79] Tak W. Yan and Héctor García-Molina. Index structures for selective dissemination of information under the boolean model. *TODS*, 19(2):332–364, 1994.

[80] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*, 2013.

[81] Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. An efficient publish/subscribe index for ecommerce databases. *PVLDB*, 7(8):613–624, 2014.

[82] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.

[83] Dongxiang Zhang, Kian-Lee Tan, and Anthony K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013.

[84] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee. Location-based spatial queries. In *SIGMOD*, pages 443–454, 2003.

[85] Jingbo Zhou, Anthony K.H. Tung, Wei Wu, and Wee Siong Ng. A semi-lazy approach to probabilistic path prediction in dynamic environments. In *KDD*, pages 748–756, 2013.

[86] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005.

[87] Zenan Zhou, Wei Wu, Xiaohui Li, Mong Li Lee, and Wynne Hsu. Maxfirst for maxbrknn. In *ICDE*, pages 828–839, 2011.

[88] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.