Swinburne University of Technology

Doctoral Thesis

Spatial Partitioning of Road Traffic Networks and their Temporal Evolution

Author:
Tarique Anwar

Supervisors:
Prof. Chengfei Liu
Prof. Hai L. Vu

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Centre for Computing and Engineering Software Systems
Faculty of Science, Engineering and Technology

May 2017

Abstract

Urban areas generally attract people from all interior areas. According to the current global trend, people are rapidly migrating from rural towards urban areas for several reasons that include availing better livelihood services and seeking better employment opportunities. Consequently, the population of cities all over the world is increasing significantly, and thereby raising the mobility demands manyfold. This strongly motivates the research areas of urban planning and urban computing to develop innovative technologies and move towards smart and more sustainable cities. As most of the urban population travel daily or frequently for their work or studies, traffic congestion has become a very important practical problem. It is affecting the urban population directly by incurring extra cost on the fuel and extra time spent, and indirectly in many ways. An important concern in smart urbanization of our societies is the avoidance of such congestions and maintenance of a smooth transportation. While the infrastructure development is one direction to deal with this problem, the analysis of spatial traffic data to discover the congestion formation and propagation patterns, and apply them to optimize the traffic flow is another direction. The research on road traffic networks data analysis is growing with the problems like fastest route computation, traffic clustering, traffic prediction, emerging event detection, anomaly detection and bottleneck identification. To discover the congestion patterns, the continuous tracking of the spatiotemporal evolution of the traffic load leading to congestions is an important problem. The research on development of methods to identify the congested partitions effectively and track their evolution efficiently has been very limited so far. In this thesis, we aim to capture the spatiotemporal evolution of urban road traffic networks. To this end, we propose technical methods to effectively partition road traffic networks in order to obtain the differently congested partitions at a point of time, and incrementally update those partitions in an efficient manner in order to track their evolution in real time.

We firstly present a scalable method for traffic-based spatial partitioning of urban road traffic networks. It is based on a spectral theory based novel graph cut (referred as α-Cut) to partition the supergraph (or any graph) constructed in our method. Using density based clustering concepts we propose density peak graphs, and use them along with α-Cut to develop a robust framework for partitioning large road traffic networks. It provides an option to select a suitable trade-off between efficiency and accuracy. We show that for large networks where efficiency is an important concern, we can opt for the settings to fasten its execution. This comes at the cost of compromising accuracy up to some extent. Then we present a comprehensive framework to track and capture the spatiotemporal evolution of road network partitions. This is done by incrementally updating the differently congested partitions available from the previous time point in an efficient manner. It consists of a physical layer that performs all the low-level computations to incrementally update a large number of small-sized road network building blocks, and a logical layer that performs high-level computations in order to serve as an interface to query the physical layer about the congested partitions. We also present an in-memory index to capture the historical information and keep them compactly saved. At last two important applications are shown to study the traffic congestion propagation patterns, which are the temporal tracking of congested partitions, and the traffic diffusion and influence estimation of road segments. These applications demonstrate the usefulness of our research in the real environments. We investigate real traffic data, present our application-specific experimental study, and show some interesting insights found in the study.

by Tarique Anwar

Acknowledgements

This thesis would not have been possible without the help, support and guidance of some very important people in my life. Firstly, I would like to thank my parents for their unconditional love, support and encouragement, and for being with me on each and every step. They are also my first teacher and a great source of energy in my life. Whatever I am today is only because of them.

I would like to express my deepest gratitude to my supervisors Prof. Chengfei Liu, Prof. Hai L. Vu, and Prof. Christopher Leckie for their visionary guidance, insightful comments and continuous support towards the completion of my PhD thesis. They were a great source of inspiration during the entire period of my candidature. Their guidance extremely helped me for becoming an independent researcher in my field. I have learned many things from them which were not only confined to my research work, but also included many other perspectives of life for becoming a strong person.

I would like to acknowledge Swinburne University of Technology for providing various facilities and support for conferences and trainings, to finish my PhD research successfully. I would also like to acknowledge Data61 (formerly NICTA) for funding my entire PhD and providing financial support for attending conferences.

I should take this opportunity to remember my previous university Jamia Millia Islamia (India), where I built my foundations in computer science. I am very thankful to my master's thesis supervisors Prof. Muhammad Abulaish and Dr. Jahiruddin, who encouraged me for research from a very early stage. I thank all of my teachers at Jamia Millia Islamia particularly Prof. Khurram Mustafa, Dr. Mansaf Alam, Dr. Syed Zeeshan Hussain, Dr. Syed Kazim Naqvi, and Dr. Rafat Parveen, for their moral support and encouragement. I am also indebted to all of my teachers from my primary and secondary schools. I am very thankful to my colleagues at King Saud University (Saudi Arabia) particularly Faraz Ahmed, M. Zubair Rafique, Arif Ali Mondal, Wazdy Essam, and Waleed Halboob.

I would like to thank and appreciate all members of our research group at Swinburne especially Dr. Saiful Islam, Dr. Jianxin Li, Mushfique Anwar, Lu Chen, Mehdi Naseriparsa, Ahmed Abbas AL-Shammari, and Amin Rigi, for their support and encouragement. I am also sincerely indebted to my friends in Melbourne particularly Siamak Nezami Doost Alamdari, MD Aquib, MD Shoeab Hassan, Humza Khan, Azimullah, Shimul Nath, ASM Kayes, Shibli Saleheen and Arup Sarker for always being there for all kinds of help. I feel blessed to have had such extremely talented and friendly people around me.

Lastly my love and appreciation go to my sisters for their endless love and prayers.

Declaration of Authorship

I, Tarique Anwar, declare that this thesis titled, 'Spatial Partitioning of Road Traffic Networks and their Temporal Evolution' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Contents

List of Figures

List of Tables

To my parents and sisters

Chapter 1

Introduction

These days there is a rapid global migration of people towards urban areas, and this is leading to a rapid increase in the traffic load on urban road networks. According to UN estimates, 70 per cent of the world's population will live in cities by 2050, and the trend in Australia is well advanced with almost 90 per cent of the population currently residing in urban areas [1]. This huge population load on urban areas is a major reason for frequent traffic congestions, specially in the peak office hours and around city centre areas. It imposes a direct loss of money spent on the extra fuel consumption and time spent while waiting in jams, and many times it may also adversely affect personal health, which are borne by the travellers. Variability of travel time due to varying congestion on the same route at different times and environmental pollution induced by the extra burnt fuel are some other affects of traffic congestion. According to the BTRE (Bureau of Transport and Regional Economics) Australia report on the urban traffic and congestion trends in Australian cities [2], Sydney, Melbourne, and Brisbane, are the most congested cities in terms of road traffic. The *avoidable cost*[1] of congestion for Australia is projected in the report as high as $20.4 billion by 2020. This huge cost caused by the overloaded urban road networks severely urges for technologically improved infrastructure to meet the rapidly growing traffic volume and traffic congestion.

[1]Where the benefits to road users of some travel in congested conditions are less than the costs imposed on other road users and the wider community

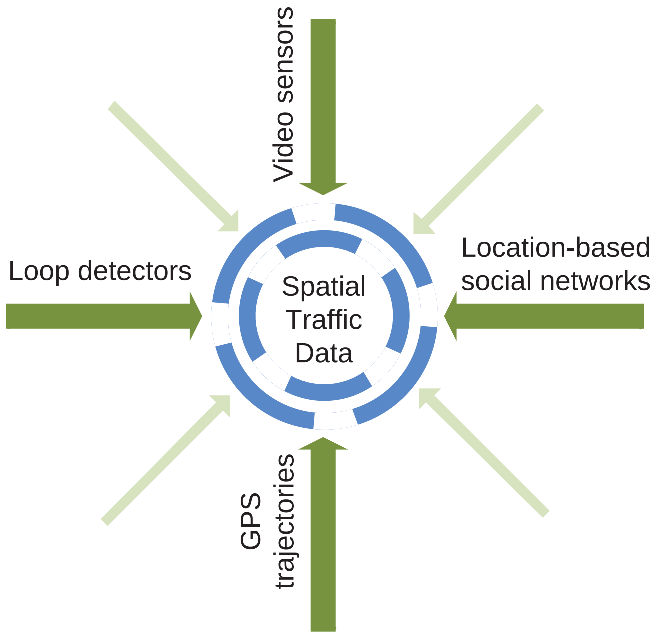


Figure 1.1: Collection of spatial traffic data

There are two possible ways to meet the growing demand of road traffic networks. The first obvious way is by improving the physical infrastructure by increasing the area of public spaces, widening of existing roads and building new roads, so that sufficient space is always available for congestion free movements of the urban population. However, many times there is a very limited scope for the government to go with this option. This is because the main city centre areas, where congestions are most likely to occur for most of the time, are generally occupied with buildings and settlements. So this way of treatment is not always an available option. The other way to deal with this problem is to intelligently optimize the traffic flow in the existing physical infrastructure. In a practical sense, what commonly happens is that the traffic does not get congested at all the places at the same time. Rather, congestion occurs when the traffic gets confined to a particular location, and the remaining locations become sparse. Thus the problem of congestion often arises due to the uneven distribution of traffic in the road network. This can be controlled up to some extent by evenly distributing the traffic throughout the network and thus optimizing the flow with the help of intelligent traffic control systems. This is possible only with the help of real-time traffic data. These days we continuously accumulate huge amounts of multidimensional geospatial traffic data or *travel footprints* from different sources, as shown in Figure 1.1. Almost all developed major cities today have loop detectors and video sensors installed at road intersection points, which log the traffic data automatically by detecting vehicles in real-time. The widely used smart phones and GPS-equipped vehicles capture the real-world movement trajectories. It is not just limited to these data. The most recent development in traffic data generation is the location based social media applications like Twitter[2], Facebook[3] and Foursquare[4], which log geospatial digital footprints of the social media users. Each source produces data with a different kind of traffic information and may or may not be directly related to others. As the data sources are available these days, it requires the development of data analysis techniques that are effective enough to understand, discover and control the evolving nature of congestion, and efficient enough to deal with the data of large urban road networks in real time.

1.1 Problem Statement

In this section, we present the research problem targeted by us in this thesis. As the urban areas are growing rapidly, urban planning and urban computing are emerging as important areas for multi-disciplinary research. Development of smart cities with smart transportation services and smart health care are some of the main concerns.

This thesis aims at contributing to an intelligent traffic (and/or congestion) management by developing traffic data analysis methods to aid the smart transportation services. Our problem at a high level is to develop mechanisms that capture the traffic congestion scenario in urban road networks and track the spatio-temporal evolution. There could be a range of possible statuses. Some parts of the network may be highly congested, while others with no congestion at all. At some points of time (for example, in the mid-night), the entire urban road network may be free of congestion, while at other points of time (for example, in the morning office-opening hours or evening office-closing hours), the entire urban road network may get congested. Thus the traffic congestion is marked by the characteristics of being *spatially diverse* and *temporally dynamic*.

[2]https://twitter.com/
[3]https://facebook.com/
[4]https://foursquare.com/

The problem considered in this thesis is to partition urban road traffic networks in order to obtain the differently congested partitions. These obtained partitions are considered as a good representative for the traffic congestion scenario at a point of time. The problem further extends to the tracking of the temporal evolution of the partitions over a period of time. This altogether captures the spatio-temporal evolution of the traffic congestion in urban road traffic networks.

1.2 Challenges

The road networks experience the traffic load on their road segments. The traffic of individual vehicles randomly originates at different locations in the network, and flows via the connected road segments. While the congestion on individual road segments is a naive approach of looking into the congestion scenario, the identification of the differently congested partitions gives a high level meaningful information from an urban-scale perspective. The main challenges in their identification and capturing their evolution are given below.

- **Structure of Urban Road Networks**

  The urban road networks are known to be unique in terms of their structure. Unlike social networks or other information networks, these are physical networks and associated with spatial properties. The dynamic traffic on them is another dimension to add more complexities. Some roads have the traffic flowing in both the oppositely directed road segments (called two-way roads), and others have the traffic flow in only one direction (one-way roads). Though the two directions of a two-way road are part of the same road but their traffic load may be completely different. For example, in the morning when the offices start opening, the traffic flows from the city outskirts towards the city centre areas. The city-inward traffic is much higher than the city-outward traffic at this time. This situation reverses itself in the evening, when offices in the city start closing and people start moving outwards back towards their home. This time the traffic load is more on the city-outward road segments. There are many other factors also on which the different direction of traffic flow is dependent. The urban road traffic networks need to be treated in such a way that the two different directions of traffic flow are considered differently. Thus one of the main challenges to work with these networks is to give a mathematical representation of the actual physical road network, which takes the mentioned issues into account and handles them wisely.

- **Identification of Traffic Congestion**

  One way to understand the traffic congestion is by looking into the traffic density of individual road segments at the micro level. This is an easy way that can be done by just going through the raw traffic data. But the question that still remains open is– ”Does it give the complete information about the spread of congestion?”. Practically a congestion occurs on the road networks as a block affecting multiple connected road segments. It originates from one or few specific road segments, grows through the roads connected to them, and thus spreads into other parts of the network. Similarly, when the traffic load starts reducing, the congestion starts shrinking by firstly freeing those affected roads that lie on the boundary. One after another the outer roads are freed and thus the congestion block keeps shrinking until it vanishes off. By considering only the micro level traffic density on each individual road segment, we miss the notion that a single congestion is actually formed by a set of multiple affected road segments. The second challenge in our research problem is the develop effective methods to identify the congestion spread or the differently congested partitions in such a way that captures all the associated properties from an urban-scale perspective.

- **Capturing the Evolution**

  In addition to the high complexity of urban road networks, they are also getting bigger in size rapidly. On the other hand, the dynamic traffic load on them, keeps changing continuously in every moment. Thus the *spatial expanse* and the *temporal frequency of traffic update* are the two factors that demand the treatment of the road traffic networks to be efficient. In the traffic control systems like SCATS, the traffic signals rotate for around one to three minutes to form a cycle, and the traffic data are logged in each signal cycle by the installed sensors. To process this data in real time, the computations for each cycle have to be completed before the beginning of the next cycle. In addition to this, in each cycle or time point the developed techniques would generate lots of information about the traffic congestion scenario. To analyze the temporal evolution, we need to store the required historical information generated by the method, in the memory, so that they can be referenced later whenever needed. This storage has to provide an efficient retrieval of the stored data and at the same time consume a minimum space in the memory. Thus, the challenges in capturing the evolution are two folds. First is that the developed methods need to be efficient enough to take into account the frequently changing traffic in real time, and the second is that the required historical information have to be compactly stored for their efficient retrieval.

1.3 Application Scenarios

Everyday almost all of the urban population make at least one journey. The most important thing that we are concerned about is *how congested is our route to destination.* The traffic management authorities and the adaptive traffic control systems try their best to disperse the congestion and maintain a smooth traffic flow. Capturing and analyzing the evolution of traffic congestion with the help of road network partitions can improve our understanding about the congestion scenario. There are two main application scenarios presented below. The first one is from the perspective of traffic management, whereas the second one is from the perspective of commuters, both of which are important for a pleasant journey.

- **Traffic Management**

  These days the developed urban areas are well equipped with urban traffic control systems (UTCS) like Sydney coordinated adaptive traffic system (SCATS) and split cycle offset optimisation technique (SCOOT). These traffic control (or management) systems log the aggregated traffic movement data on each of the road segments in real-time, and based on them the signal cycle is adaptively controlled. One major application area is to use the evolving road network partitions in the traffic control systems from a centralized database to adaptively control the signal cycle durations, and intelligently disperse the congestion wherever it is formed in the network.

- **Journey Planning**

  The commuters always want to plan their journey in advance, so that they can make it a pleasant and fastest one. We try to avoid the routes or destinations where there are chances to get stuck in a traffic jam. There are several websites and mobile applications that guide us in order to have a pleasant and fast journey. `Google Traffic`, a service of `Google Maps`, anonymously collects the traffic data from different sources (mostly smart mobile phones), and visualizes them on a map in real time. People can look into the traffic level on the road segments they are going to follow and plan their route and time of journey accordingly. Our research on the road network partitions and their evolution give a rich and meaningful information about the traffic congestion from an urban-scale perspective. Effectively visualizing the congestion scenario using the differently congested partitions in real time can aid the commuters to understand the spread of traffic congestion in a better way. In this way it can be applied to develop a smart journey planner.

1.4 Contributions of This Thesis

This thesis contributes towards the development of technical methods for capturing and analyzing the evolution of traffic congestion in real time. Our main contributions specifically are as follows.

- **Spatial Partitioning of Urban Road Traffic Networks**

  As mentioned earlier, road traffic networks are unique in their kind. To identify the different congestions (in the form of blocks or partitions) occurring in the road network at a time point, spatial partitioning of the network needs to be applied. Till date, there has been very little work on this problem. Our first contribution is the development of effective and scalable methods for partitioning urban road networks in such a way that the different resulting partitions have homogeneous level of traffic congestion inside, and are heterogeneous to the other partitions outside. We developed two methods, both of which start with transforming the real road network into a graph representation called *road graph*

i) The first method is a two-stage procedure that firstly condenses the large road graph into a well-structured supergraph via clustering and link aggregation based on traffic density and adjacency connectivity, respectively. Based on spectral theory, we developed a novel partitioning algorithm called $\boldsymbol{\alpha}$-Cut, to partition the supergraph. Our experimental results show that this method outperforms the existing normalized cut based partitioning algorithm. This research is presented in detail in Chapter 3. It has been published as a paper in the proceedings of the *17th International Conference on Extending Database Technology* (**EDBT**) held in Athens (Greece) in 2014 [3].

ii) The second method is actually a robust framework that is based on both density based and spectral based clustering. This framework has a two-stage algorithm (referred as FaDSPa) that first transforms the large road graph into a well-structured and condensed density peak graph (DPG) via density based clustering. Thereafter our spectral theory based $\boldsymbol{\alpha}$-Cut (developed in the previous method) is applied on the DPG to partition and obtain the different sub-networks. This framework allows to select the trade-off between efficiency and accuracy. This research is presented in detail in Chapter 4. It has been published as a paper in the journal **Information Systems** in 2017 [4].

- **Efficient Incremental Update of Partitions**

  In the temporal dimension, the dynamic traffic keeps changing the load continuously. Partitioning the road network at each time point incurs heavy computation cost and takes a long time in execution. Generally this change occurs gradually, and the difference of the traffic level in two successive time-points is small. In such case, a logical and more efficient solution is to incremental update the already available partitions from the previous time point with respect to the change in traffic. Our second contribution is the development of a comprehensive framework to capture the spatiotemporal evolution of road network partitions. It consists of a method that incrementally updates the partitions in an efficient manner, and an in-memory index that compactly stores the historical information, detailed below.

  i) The method to incrementally update the available road network partitions at each new time point is based on a two-layer approach. The physical layer maintains a large number of small-sized road network building blocks, and performs low-level computations to incrementally update them. The logical layer interacts with the physical layer, and performs high-level computations in order to serve as an interface to query the physical layer about the congested partitions. This research has been published as a paper in the proceedings of the *25th International Conference on Information and Knowledge Management* (**CIKM**) held in Indianapolis (USA) in 2016 [5].

ii) We also develop an in-memory index called Bin that compactly stores the historical sets of building blocks with no information loss and facilitates their efficient retrieval. Experimental results are presented to demonstrate the effectiveness and efficiency of the framework.

This research is presented in detail in Chapter 5. We are in the process to submit this complete framework as a paper in the journal **IEEE Transactions on Knowledge and Data Engineering** for publication.

- **Application-specific Experimental Study using Real Data**

  The continuous evolution of the real road traffic networks undergo different kinds of situations that are generally not reflected in the synthetically generated datasets. Therefore it is very important to study the applications of the technical methods using real data. Our third contribution is an experimental study using real data on two specific evolutionary applications. It also demonstrates the usefulness of our previous contributions for real environments.

  i) In the first application, using our $\boldsymbol{\alpha}$-Cut partitioning algorithm we develop a simple and efficient framework for identifying the spatial congested partitions, and dynamically tracking the temporal change in their location and structure. We identify the congested partitions based on real traffic measures (volume and green time utilization) available from the traffic signal control system. We conducted extensive experiments on Melbourne (Australia) road networks, and came up with some interesting insights about the way congestion forms and propagates in the network. This research has been published in the proceedings of the *Transport Research Board 95th Annual Meeting* (**TRB**) held in Washington D.C. (USA) in 2016 [6], and in the journal **Transportation Research Record: Journal of the Transportation**

Research Board in 2016 [7]. It was also nominated for the Kikuchi-Karlaftis best paper award.

ii) In the second application, we study the diffusion of traffic from one road segment to other connected road segments. This traffic diffusion phenomenon makes some roads more influential than others in terms of congestion propagation. We develop an algorithm called `RoadRank` that computes the influence scores of each road segment in an urban road network, and rank them based on their overall influence. It also updates the influence scores incrementally with time based on the latest traffic measures. In our experimental study, we found some interesting results from the Melbourne (Australia) road networks. This research has been published as a paper in the proceedings of the *24th International Conference on Information and Knowledge Management* (**CIKM**) held in Melbourne (Australia) in 2015 [8].

Our experimental study on both the applications mentioned above, is performed using the real SCATS data of Melbourne road network provided by VicRoads[5]. This research is presented in detail in Chapter 6.

1.5 Structure of This Thesis

The rest of this thesis is organized as follows:

- **Chapter 2 - Literature Review** presents a review of the literature related to this thesis. We cover different aspects of the research on spatiotemporal evolution of traffic on road networks. The chapter starts with the basics of clustering and graph partitioning, and goes on to the research advancements on spatial data, clustering on spatial data, spatial network partitioning. Then moving on to dynamic and evolving networks, it presents the related works on the evolution of events in the form of clusters and influence propagation.

- **Chapter 3 - Spatial Partitioning of Urban Road Traffic Networks** presents a scalable method for traffic-based spatial partitioning of large urban road networks.

[5]http://www.vicroads.vic.gov.au/

It is a two-stage procedure that first transforms the large road graph into a well-structured and condensed supergraph via clustering and link aggregation based on traffic density and adjacency connectivity, respectively. We devise a spectral theory based novel graph cut (referred as $\boldsymbol{\alpha}$-Cut) to partition the supergraph and compare its performance with that of an existing method for partitioning urban networks. Our results show that the proposed method outperforms the normalized cut based existing method in all the performance evaluation metrics for small road networks and provides good results for much larger networks where other methods may face serious problems of time and space complexities.

- **Chapter 4 - Fast Partitioning of Road Traffic Networks Using Density Peak Graphs** presents a robust framework for spatial partitioning of large urban road traffic networks using density peak graphs. A two-stage algorithm (referred as FaDSPa) embedded in the framework, first transforms the large road graph into a well-structured and condensed density peak graph (DPG) via density based clustering and link aggregation using traffic density and adjacency connectivity, respectively. Thereafter we apply our spectral theory based $\boldsymbol{\alpha}$-Cut to partition the DPG and obtain the different sub-networks. Experimental results are presented to demonstrate the effectiveness and efficiency of the framework.

- **Chapter 5 - Tracking and Capturing the Spatio-temporal Evolution of Congestion** presents a comprehensive framework to track and capture the spatiotemporal evolution of road network partitions by incrementally updating them in an efficient manner. It is based on a two-layer approach. The physical layer maintains a set of small-sized road network building blocks, and performs low-level computations to incrementally update them, whereas the logical layer performs high-level computations in order to serve as an interface to query the physical layer about the congested partitions. We also present an in-memory index called Bin that compactly stores the historical sets of building blocks with no information loss and facilitates their efficient retrieval. Experimental results are presented to demonstrate the effectiveness and efficiency of the framework.

- **Chapter 6 - Applications using Real Traffic Data** investigates real traffic data and presents some application-specific experimental study. To discover the congestion propagation patterns, we study two applications: ***i***) temporal tracking of congested partitions, and ***ii***) traffic diffusion and influence estimation. We present some interesting insights using real SCATS data. Through this study, we demonstrate the importance of our research contributions in practical aspects.
- **Chapter 7 - Conclusion and Future Work** provides the concluding summary of this thesis, the possible extension of the works presented in this thesis and other unexplored areas as future research direction.

Chapter 2

Literature Review

The area of network partitioning and network evolution has received wide attention from different research communities. In the context of road traffic networks, the attention has been very recent and limited. It involves multiple research areas including clustering and graph partitioning as a core concept, treatment of spatial data and spatial networks, and evolution of partitions and influence propagation in the dynamic networks. In this chapter, we present a detailed survey related to our research. We firstly start with a review of the area of clustering in Section 2.1, including the foundational algorithms and the latest developments. It is followed by the works dealing with spatial data in Section 2.2, including basic index structures and the common problems that people have worked on these data. Then in Section 2.3, we present the advancements in the area of clustering on spatial data and spatial networks. Section 2.4 presents a review of the research on dynamic and evolving networks, focusing on the specific areas of evolution of clusters, influence propagation, and traffic congestion. Finally we compare our work with the existing works in Section 2.5.

2.1 Clustering

Clustering is one of the most important and ever growing research area in data mining. It refers to the task of grouping objects with similar properties together, while separating from those with dissimilar properties. It has some important applications in spatial data mining.

2.1.1 Foundations of Clustering

Clustering and graph partitioning apply to a wide variety of research problems, and many solutions have been proposed in the past. Some of the basic and foundational clustering algorithms including $\boldsymbol{k}$-means, DBSCAN, and SCAN are briefly explained below. Nowadays these basic algorithms are being used to develop further advanced and hybrid clustering techniques.

- ***K*-means**: The k-means clustering algorithm [9] is one of the simplest and most widely used unsupervised method to identify the grouping patterns in a data set. It works to group the data items into $\boldsymbol{k}$ clusters based on similarity of their feature values, where the value of $\boldsymbol{k}$ needs to be pre-determined. Let us suppose, we have a set of $\boldsymbol{n}$ data items $\boldsymbol{\mathcal{D} = \{d_1, d_2, \ldots, d_n\}}$, where each $\boldsymbol{d_i}$ has $\boldsymbol{m}$ feature values forming an $\boldsymbol{m}$-dimensional vector $\boldsymbol{< f_1, f_2, \ldots, f_m >}$. The $\boldsymbol{k}$-means clustering algorithm aims to find out the subsets $\boldsymbol{\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k\}}$ that minimizes $\boldsymbol{\sum_{l=1}^{k} \sum_{d_i \in \mathcal{D}_l} dist(d_i, \mu_l)}$, where $\boldsymbol{\mu_l}$ is the mean value of subset $\boldsymbol{\mathcal{D}_l}$ and $\boldsymbol{dist()}$ is the function that computes the distance between the two vectors provided as input. Euclidean distance $\boldsymbol{||d_i - \mu_l||^2}$ has been found to be the simplest and most suitable for distance measurement in most of the cases. The method starts with randomly picking $\boldsymbol{k}$ data items for the $\boldsymbol{k}$ means. After that all the data items are compared with the means of each group and assigned to the closest one, and the mean values are updated after all the data items have been assigned. The distance computing of data items with the updated means and their assignment to the closest group continues iteratively until convergence.

  The $\boldsymbol{k}$-means [9] and expectation-maximization [10] clustering algorithms work well for finding ellipsoidal or convex shaped clusters, but fail to find non-convex clusters. Density based clustering algorithms [11] are able to find clusters of arbitrary shapes. Areas of higher density are considered as clusters, and the nodes in the sparse areas separating the dense areas are considered as noise or boundary nodes.

- **DBSCAN**: The density based spatial clustering of applications with noise (DBSCAN)[12] algorithm is the most popular of this kind. It identifies a cluster by looking into the neighborhood of each object within a predefined radius of ϵ distance. With each *minpts* (predefined) objects in the ϵ-neighborhood, a new cluster is formed. However, this algorithm is highly sensitive to the thresholds ϵ and *minpts*, and choosing an appropriate threshold is very important to get accurate clusters [13].

- **SCAN**: Extending the DBSCAN concepts, SCAN (structural clustering algorithm for networks) [14] partitions a graph based on its structure to detect the clusters, hubs and outliers.

- **Density-based clustering using density peaks**: The authors in [13] recently proposed a fast density based clustering method by finding the density peaks locally. They assume that the cluster centers are surrounded by neighbors with lower local density and they are at a relatively large distance from any point of higher local density.

- A recently proposed algorithm SCAN++ [15] uses a new data structure called directly two-hop-away reachable node set (DTAR) to efficiently partition a graph in order to get the same results as produced by SCAN. Spectral clustering algorithms like minimum cut and normalized cut have remained quite popular [16, 17]. In [18], the authors proposed a spectral cut based on the min-max clustering principle for graph partitioning in a data clustering point of view. In [17], White and Smith proposed a spectral clustering based solution to find communities in graphs by partitioning. Their objective function is based on network modularity (defined later in Section 2.1.4). The modularity of a set of graph partitions is defined as the difference between the observed and the expected fraction of links within a partition. Larger modularity values are correlated with better graph partitioning. The minimization of our $\boldsymbol{\alpha}$-Cut (proposed in Chapter 3) approximately maximizes the modularity, and thus it gives an indication of good performance of our $\boldsymbol{\alpha}$-Cut.

2.1.2 Graph Partitioning

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{v_1, v_2, \ldots, v_{n_v}\}$ is the set of nodes, and $\mathcal{E} = \{e_1, e_2, \ldots, e_{n_e}\}$ is the set of edges. The problem of graph partitioning is the decompose $\mathcal{G}$ into multiple disjoint subgraphs or graph partitions $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, such that each subgraph $\mathcal{P}_i$ is a connected component in itself, and satisfy the required properties. These properties often come from numerical weights associated with the nodes and edges in the graph. This weighted graph can be denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W}_\mathcal{V}, \mathcal{W}_\mathcal{E})$ in complete detail, where $\mathcal{W}_\mathcal{V} = \{wv_1, wv_2, \ldots, wv_nv\}$ is the set of weights associated with each node v_i, and $\mathcal{W}_\mathcal{E} = \{we_1, we_2, \ldots, we_ne\}$ is the set of weights associated with each edge e_i.

Partitioning $\mathcal{G}$ would generate a set of partitions $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, where each $\mathcal{P}_i$ consists of a set of nodes $\mathcal{V}_i$, edges $\mathcal{E}_i$, and associated weights, if any. More generally, the partitioning of $\mathcal{G}$ divides the set $\mathcal{V}$ into k disjoint sets.

$$\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k\} \leftarrow \mathcal{V} \tag{2.1}$$

such that,

1. Nodes inside each $\mathcal{V}_i$ form a connected component;
2. $\mathcal{V}_i \cap \mathcal{V}_j = \phi$; and
3. $\{\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_k\} = \mathcal{V}$

Due to the wide applications of graph partitioning [19], including distributed graph processing and image segmentation, it has been studied in different contexts, with application-specific properties required in the partitioned subgraphs. Traditionally, this partitioning was mainly based on the associated weights, but there are also other emerging applications including community detection in social networks, where the partitions are formed based on the linkage structure in the graph [20]. This kind of partitioning lies into the category of structural graph partitioning.

Typically graph partitioning is an NP-hard problem, and the solutions are generally based on heuristics and approximation algorithms. There are two approaches to solve this problem- the local approach [21, 22] and the global approach [16, 23]. The local approach looks into the properties inside the graph locally to identify the possible partitions initially from whom the final partitions are generated. It has been found that they suffer from the problem of arbitrary initial clustering of the set of nodes [21, 22], which greatly affect the final partitions. On the other hand, global approaches look into the whole graph globally to identify the portions from where the graph links can be cut to generate the final partitions. Spectral clustering [16, 24, 25], widely used in image segmentation, has remained very popular for solutions in this category. The final partitions are derived from spectrum of the graph adjacency matrix.

2.1.3 Spectral Clustering

Spectral clustering is a class of clustering algorithms which has its foundations in graph theory and linear algebra, and clusters the set of data using eigenvectors and eigenvalues of matrices derived from the data [23]. The algorithms in this class treat clustering as a graph partitioning problem by optimizing different graph-based measures, and ignoring assumptions based on cluster shapes. Thus their performance and success depends heavily on how intelligently these measures, a.k.a. *graph cuts*, are devised. Maximizing *average association*, minimizing *average cut*, and minimizing *normalized cut* are the three most popular measures for spectral optimization [16].

For a given a set of $\boldsymbol{n}$ data items, these algorithms construct a graph $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{W}})$ where the set of nodes $\boldsymbol{\mathcal{V}}$ is the set of $\boldsymbol{n}$ data items, $\boldsymbol{\mathcal{E}}$ is the set of links between them, and the weights $\boldsymbol{\mathcal{W}}$ of links are defined by the affinity measure between the pair of nodes. The graph is represented in the form of its weighted adjacency matrix $\boldsymbol{A_{n\times n}}$. The spectral clustering algorithms aim to find $\boldsymbol{k}$ disjoint partitions $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \ldots, \boldsymbol{\mathcal{P}_k}\}$ of $\boldsymbol{\mathcal{G}}$, where $\bigcup_{l=0}^{k} \boldsymbol{\mathcal{P}_l} = \boldsymbol{\mathcal{V}}$, at the same time optimizing a certain *cut* (cost) function.

Let $\boldsymbol{A}$ be the weighted adjacency matrix of graph $\boldsymbol{G}$. The degree matrix of $\boldsymbol{G}$ is a diagonal matrix, derived from $\boldsymbol{A}$ by adding up the entries in each row together as shown in Equation 2.2.

$$\boldsymbol{D} = \begin{pmatrix} \sum_{j=1}^{n} \boldsymbol{a_{1j}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \sum_{j=1}^{n} \boldsymbol{a_{2j}} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \sum_{j=1}^{n} \boldsymbol{a_{nj}} \end{pmatrix} \tag{2.2}$$

The *Laplacian matrix* $\boldsymbol{L}$ of $\boldsymbol{G}$ is computed by subtracting the adjacency matrix from the degree matrix as shown in Equations 2.3, 2.4, and 2.5.

$$\begin{aligned} \boldsymbol{L} &= \boldsymbol{D} - \boldsymbol{A} \quad (2.3) \\ &= \begin{pmatrix} \sum_{j=1}^{n} \boldsymbol{a_{1j}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \sum_{j=1}^{n} \boldsymbol{a_{2j}} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \sum_{j=1}^{n} \boldsymbol{a_{nj}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{a_{11}} & \boldsymbol{a_{12}} & \cdots & \boldsymbol{a_{1n}} \\ \boldsymbol{a_{21}} & \boldsymbol{a_{22}} & \cdots & \boldsymbol{a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{a_{n1}} & \boldsymbol{a_{n2}} & \cdots & \boldsymbol{a_{nn}} \end{pmatrix} \quad (2.4) \\ &= \begin{pmatrix} \sum_{j\neq 1} \boldsymbol{a_{1j}} & -\boldsymbol{a_{12}} & \cdots & -\boldsymbol{a_{1n}} \\ -\boldsymbol{a_{21}} & \sum_{j\neq 2} \boldsymbol{a_{2j}} & \cdots & -\boldsymbol{a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{a_{n1}} & -\boldsymbol{a_{n2}} & \cdots & \sum_{j\neq n} \boldsymbol{a_{nj}} \end{pmatrix} \quad (2.5) \end{aligned}$$

A function $\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_j)}$ is defined in Equation 2.6 as the sum of weights associated with all the links having their nodes at one end in $\boldsymbol{\mathcal{P}_i}$ and the nodes at the other end in $\boldsymbol{\mathcal{P}_j}$.

$$\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_j)} = \sum_{e_r \in \left\{ links(\mathcal{P}_i, \mathcal{P}_j) \right\}} \boldsymbol{w_r} = \sum_{v_p \in \mathcal{P}_i, v_q \in \mathcal{P}_j} \boldsymbol{A(p,q)} \tag{2.6}$$

Minimum Cut

Definition 2.1. (**Cut**) For a given partition set $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \ldots, \boldsymbol{\mathcal{P}_k}\}$ the *cut* of a partition $\boldsymbol{\mathcal{P}_i}$ is defined as the summation of weights associated with all the links having their nodes at one end in $\boldsymbol{\mathcal{P}_i}$ and nodes at other end in any partition other than $\boldsymbol{\mathcal{P}_i}$, i.e., $\boldsymbol{W}(\boldsymbol{\mathcal{P}_i}, \overline{\boldsymbol{\mathcal{P}_i}})$. ■

The *cut* value of a partition $\boldsymbol{\mathcal{P}_i}$ gives a measure of connectivity strength between $\boldsymbol{\mathcal{P}_i}$ and the rest of the partitions, and thus quantifies the loss incurred in cutting those link connections while partitioning the graph. The summation of the cut of all the partitions in $\boldsymbol{\mathcal{P}}$ gives the total cut value. A *minimum cut* based partitioning aims partition the graph in such a way that the total cut is minimized. Thus it focuses on identifying partitions that are highly heterogeneous to each other.

Association Cut

Definition 2.2. (**Association**) For a given partition set $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \ldots, \boldsymbol{\mathcal{P}_k}\}$ the *association* of a partition $\boldsymbol{\mathcal{P}_i}$ is defined as the summation of weights associated with all the links having nodes at both ends in $\boldsymbol{\mathcal{P}_i}$, i.e. $\boldsymbol{W}(\boldsymbol{\mathcal{P}_i}, \boldsymbol{\mathcal{P}_i})$. ■

The *association* value of a partition $\boldsymbol{\mathcal{P}_i}$ gives a measure of connectivity strength between nodes inside $\boldsymbol{\mathcal{P}_i}$. The summation of the association of all the partitions in $\boldsymbol{\mathcal{P}}$ gives the total association value. An *association cut* based partitioning aims to partition the graph in such a way that the total association is maximized. Thus it focuses on identifying partitions that have high homogeneity inside.

Normalized Cut

It has been found that both minimum and association cuts behave in a biased manner, focusing only on the heterogeneity and the homogeneity, respectively [16, 26]. For example, minimum cut favors cutting small sets of isolated nodes in the graph and as a result the produced partitions are highly imbalanced. A good partitioning can be obtained by taking both the factors into account. *Normalized cut* (Ncut) defined in Equation 2.7 avoids this unnatural biasness by considering both the factors simultaneously. It computes the cut cost as a fraction of the edge connections between all the nodes in the graph. As its name suggests, the cut value is normalized by the association. A normalized cut based partitioning aims to partition the graph in such a way that the $\boldsymbol{Ncut(\mathcal{P})}$ value is minimized. Thus it focuses on identifying partitions that have both high homogeneity inside and high heterogeneity with others outside.

$$\boldsymbol{NCut(\mathcal{P}) = \sum_{i=1}^{k} \frac{cut(\mathcal{P}_i)}{association(\mathcal{P}_i)}} \tag{2.7}$$

The average association, average cut, and normalized cut based $\boldsymbol{k}$-way optimizing cut functions are shown in equations 2.8, 2.9, and 2.10 respectively, where $\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_j)}$ is defined in Equation 2.6.

$$\max_{\mathcal{P}} \boldsymbol{AvgAssoc(\mathcal{V})} = \max_{\mathcal{P}} \sum_{l=0}^{k} \frac{\boldsymbol{W(\mathcal{P}_l, \mathcal{P}_l)}}{|\mathcal{P}_l|} \tag{2.8}$$

$$\min_{\mathcal{P}} \boldsymbol{AvgCut(\mathcal{V})} = \min_{\mathcal{P}} \sum_{l=0}^{k} \frac{\boldsymbol{W(\mathcal{P}_l, \overline{\mathcal{P}_l})}}{|\mathcal{P}_l|} \tag{2.9}$$

$$\min_{\mathcal{P}} \boldsymbol{NCut(\mathcal{V})} = \min_{\mathcal{P}} \sum_{l=0}^{k} \frac{\boldsymbol{W(\mathcal{P}_l, \overline{\mathcal{P}_l})}}{\boldsymbol{W(\mathcal{P}_l, \mathcal{P})}} \tag{2.10}$$

There exist many more kinds of possible cuts with different desired properties. For very small graphs the optimal cut or association value for partitioning can be found by trying all the different possibilities, but for large sized graphs it is almost impossible to partition this way with the computing environments available today. The solutions to the objectives of all such kinds of cuts are based on the spectral theory, which produce approximate results [16, 24, 25, 27].

Solving any of these cut functions to find the optimal partitions set $\boldsymbol{\mathcal{P}}$ is an NP-complete problem [16]. Therefore the spectral clustering techniques apply an approximate solution by considering the $\boldsymbol{k}$ eigenvectors of a derived matrix as relaxed cluster indicator vectors, which are used to form an $\boldsymbol{n \times k}$ row-normalized data matrix. Nodes from the same cluster are expected to be close together while nodes from different clusters are expected to be well separated in the $\boldsymbol{k}$-dimensional eigenspace. Finally the simple $\boldsymbol{k}$-means clustering algorithm is applied on the $\boldsymbol{n \times k}$ data matrix, considering the $\boldsymbol{n}$ rows as different data items, where each data item is a vector of size $\boldsymbol{k}$. The obtained clusters are accepted as output of the spectral clustering algorithm.

Recently, people have adopted different heuristic, hybrid, and multilevel techniques to develop efficient techniques for graph partitioning based on different criteria and in different context [23, 28]. In [20], Zhou et al. aimed to obtain graph partitions in which the nodes inside a partition are structurally close to each other and have similar feature values, and followed a random walk based approach. Sun et al. [29] integrated the problems of ranking and clustering in heterogeneous information networks and proposed the algorithm RankClus that produces clusters with rank information of the objects in the network. There is a wide application of graph partitioning for network community detection. In [30], the authors explored some graph partitioning based community detection methods and evaluated their relative performances. However, most works on graph partitioning face time and space complexity issues with large networks.

2.1.4 Modularity

The modularity of a set of graph partitions $\boldsymbol{Q(\mathcal{P})}$ [17] is defined as the difference between the observed and expected fraction of links within a partition, and is formulated as Equation 2.11. ■

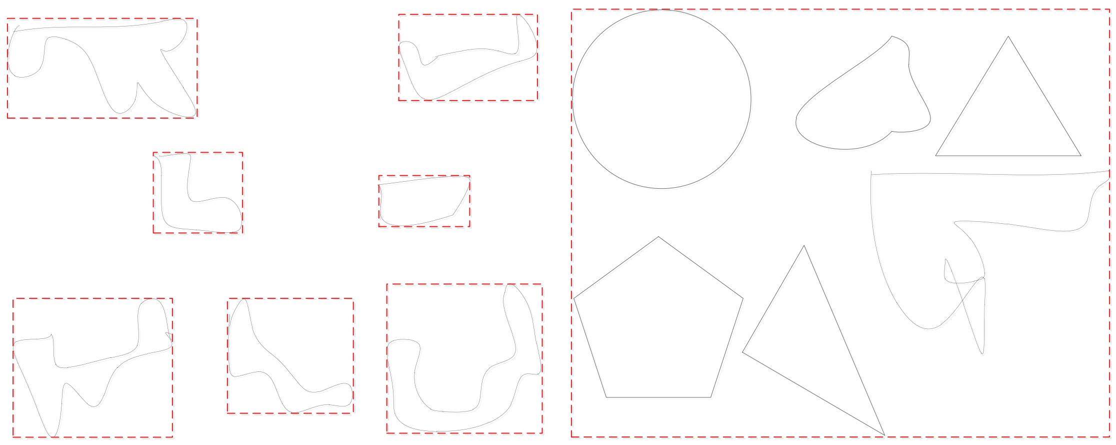
$$\boldsymbol{Q(\mathcal{P}) = \sum_{i=1}^{k} \left[\frac{W(\mathcal{P}_i, \mathcal{P}_i)}{W(\mathcal{V}, \mathcal{V})} - \left(\frac{W(\mathcal{P}_i, \mathcal{V})}{W(\mathcal{V}, \mathcal{V})} \right)^2 \right]} \tag{2.11}$$

Larger modularity values are correlated with better graph partitioning. To maximize modularity while partitioning a graph, in [17] the authors presented a spectral clustering solution. They showed that the partitioning can be obtained by using the $\boldsymbol{k}$ largest eigenvalues and corresponding eigenvectors obtained after eigen-decomposition of the modularity matrix $\boldsymbol{L_Q}$.

2.2 Spatial Data

Spatial data refers to spatial objects comprising geographic data like points, lines, regions, surfaces, volumes, and even data of higher dimension which includes time [31]. Examples of spatial data include cities, rivers, roads, counties, states, and such other geographical landmarks, which are marked by some geographic latitude and longitude coordinates. Spatial databases are able to efficiently store and query spatial data objects using suitable data structures and indexing methods. Some popular index structures for spatial data, described in the following section, are R-tree [32], R^+-tree [33], and R*-tree [34].

2.2.1 Basic Index Structures

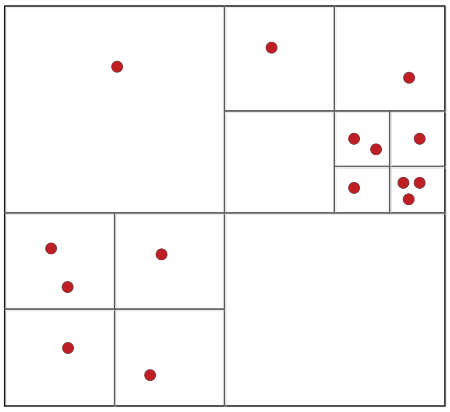


(a) MBRs of each object (b) MBR of a set of objects

Figure 2.1: MBR

MBR: The geometry of the spatial objects in a two dimensional space can range from a very simple structure like a point to complex structures like polygons or any other random shape. In spatial query processing, the evaluation of the spatial predicates is very inefficient when applied directly on the exact shapes of the objects. For an efficient solution, the shape is approximated by considering the smallest rectangle that bounds the entire object by the coordinates min(x), min(y), max(x), and max(y) in the (x,y) coordinate system. This rectangle is called a *minimum bounding rectangle* (MBR) [35, 36] for that particular object (or for a set of objects if there are multiple objects bounded by a single rectangle).

If an MBR does not qualify for the spatial query, it implies that the object in that MBR also does not qualify. In this way it improves the efficiency of this filtering step. Figure 2.1 shows some examples of MBR.


(a) Space partitioning for objects

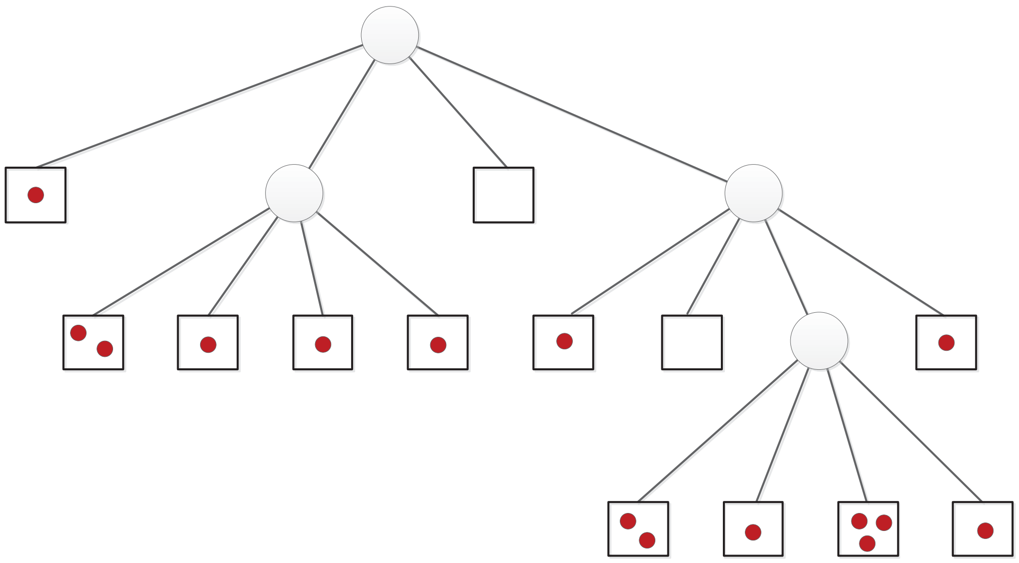
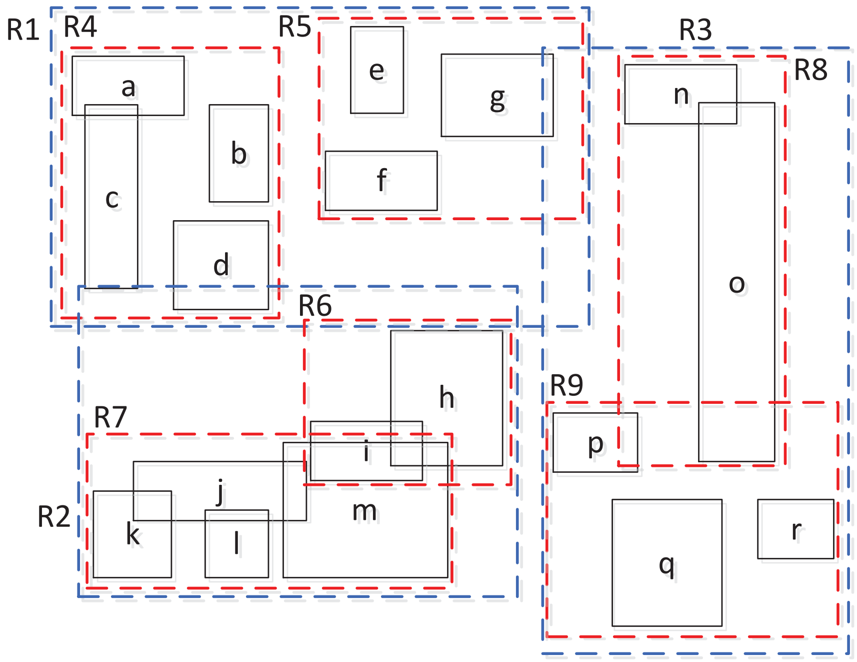
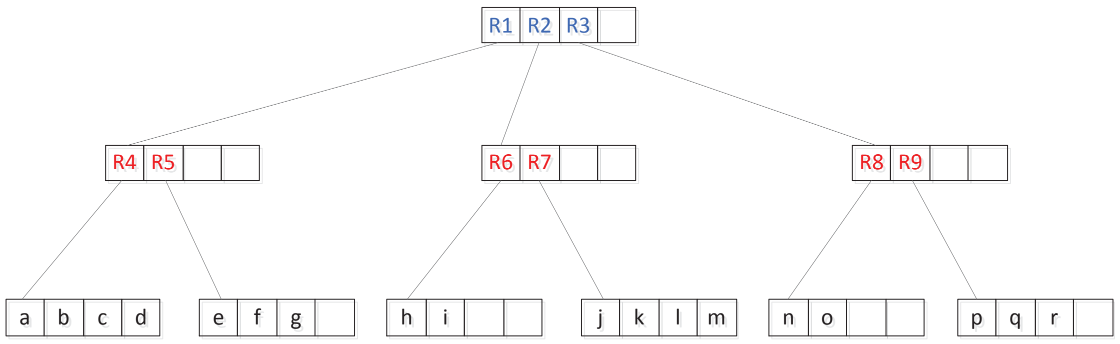

(b) Quadtree creation

FIGURE 2.2: Quadtree

Quad Tree: Quad tree [31, 35–37] is a tree structure where each internal (or non-leaf) node has exactly four children. It is used to index a two-dimensional space by recursively partitioning the space into four quadrants until the spatial objects are evenly distributed into the quadrants. The root node corresponds to the entire two-dimensional space, which is divided into four quadrants corresponding to its four children. The subdivision of quadrants in this way goes on at each level for child nodes of all the internal nodes. The data associated with the leaf nodes represent a single unit of spatial interest. Figure 2.2 illustrates the creation of a quadtree by recursive partitioning of the two-dimensional space. A quadtree is not necessarily balanced.



(a) Object and higher level MBRs



(b) R-Tree created from the MBRs

FIGURE 2.3: R-tree

R-Tree: R-Tree, proposed by Antonin Guttman in 1984 [32], is a height-balanced tree structure similar to B$^+$-tree. It indexes the MBRs of the spatial objects instead of their exact extents, and is used for spatial access methods. The main idea is that the objects are represented by their MBRs and all the closely located MBRs are grouped together and represented by their common MBR as node in the next higher level. Thus the root node consists of MBRs of the highest level, and the leaf nodes are MBRs of the individual spatial objects. Figure 2.3 shows an example of object MBRs and the created R-Tree. R-Tree is the most popular spatial index structure, and has been significantly in use for different theoretical and practical problems. An application where it is widely used is storing spatial objects like restaurants and applying nearest neighbor queries on them.

Variants of R-Tree: R-Tree is a dynamic data structure, where the tree is also updated using the insertion and deletion operations while performing the search operations. The performance of the tree is greatly affected by these alternating update operations. Altogether the performance is dependent on the coverage (the entire area to cover all related rectangles) and overlap (the entire area which is contained in two or more nodes). A minimal coverage means that the amount of *dead space* (or empty area in the nodes) is minimal, and a minimal overlap means that the set of search paths to the leaves is minimal. Both of these factors are critical to the search performance. Minimal coverage and minimal overalp are the key to achieve a good performance. R^+-Tree [33] and R*-Tree [34] are two optimized variants of the basic R-Tree, proposed in order to achieve a better performance. In R^+-Tree, there is no overlapping of MBRs of the internal nodes. The objects in the overlapping portions are inserted into multiple leaves if required. On the other hand, the R*-tree attempts to reduce both coverage and overlap by using a revised node split algorithm and the concept of forced reinsertion of nodes. It achieves a well clustered groups in the node entries, and thereby reduces the node coverage.

2.2.2 Query Processing

Processing spatial queries in spatial databases and geographic information systems (GIS) have received wide attention [38, 39]. Some popular spatial queries are range search, nearest neighbour search, closest pairs search, and spatial joins. To efficiently process such queries, effective index structures are employed to aid search algorithms in efficiently retrieving the target data objects. In range queries, the range usually corresponds to a rectangular window or a circular area around a query point [38]. R-trees efficiently process these queries. ***k***-nearest neighbor (kNN) queries retrieve ***k*** data point(s) closest to a query point. The first popular approach to process these queries on R-trees is the branch-and-bound R-tree traversal algorithm [40]. It traverses the R-tree while keeping a priority queue of the ***k*** potential nearest neighbours, and at the same time searching and pruning the unnecessary subtrees by applying the distance metrics. The reverse nearest neighbor (RNN) query, which is retrieves the m the set of points that have the query point as their nearest neighbor, was first proposed by Korn and Muthukrishnan [41]. In [42], the authors proposed the "time-interval all fastest path (allFP)" query and a solution method for its processing. Given a user-dened leaving or arrival time interval, a source node and an end node, allFP query asks for a set of all fastest paths from the source node to the end node, one for each sub-interval. Various other related problems including identifying top-k influential objects [43] and finding optimal locations [44] have also been studied in the past.

With the increasing popularity of location-based services, spatial queries including some textual keywords called spatial keyword queries have found their way. These queries are able to handle new problems like mapping textual documents or webpages to specific geographic regions. In [45], Chen et al. worked on improving performance of these queries on search engines. Top-***k*** spatial keyword queries retrieve ***k*** most suitable ranked objects on the basis of a combination of their distance to a query location and relevance to the associated query keywords in text. Efficient processing of these queries is a standard problem these days. In [46], the authors processed these queries by integrating R-trees and signature files in a hybrid index structure called IR$^{\mathbf{2}}$-tree. To improve the performance of these queries, Rocha-Junior et al. proposed a new indexing structure based on aggregated R-tree (aR-tree) in [47], and recently they proposed a technique for spatial keyword search in road networks [48]. Some more recent works on this problem are [49–51]. This problem has also expanded into various other similar problems [52].

2.2.3 Knowledge Discovery

With the advent of digitization and technological developments, in the present days we continuously accumulate huge amounts of geospatial traffic data or travel footprints from different sources. The most common of them to capture the realtime traffic are video sensors and loop detectors installed at different road network points which log the data automatically by detecting vehicles in real-time, GPS-equipped devices capturing the real-world trajectories, and location-based social media applications like Twitter[1] and Foursquare[2] logging geospatial digital footprints of urban social media users. Each source produces data with a different kind of traffic information and may or may not be directly related to others. This huge amounts of spatial data collected through various sources have the potential to deal with crucial travel and transportation problems by applying different data mining techniques. Several works of varied kind have been done in this area, including pattern mining [53, 54], trajectory clustering [55, 56], hot and popular route discovery [57, 58], fastest path computation [42, 59], mining interesting locations [60], and congestion-based spatial partitioning of road networks [61].

- **Pattern mining**: There are often some obscured or trendy patterns in the spatial or spatio-temporal data. An obvious example is the sudden increase in traffic congestion in the office opening and closing hours in the city center areas. The area of pattern mining covers the development of techniques to identify the hidden as well as obvious patterns in large spatio-temporal databases. Mamoulis et al. [53] proposed a framework to analyze, manage and query object movements to discover hidden period patterns from spatiotemporal data. Gianotti et al. [54] extended the sequential pattern mining area to introduce trajectory pattern mining to discover frequent hidden behaviors in both space and time.
- **Hot and popular route discovery**: There are often some routes more popular or preferred than others. There could be several obvious or unobvious reasons for the personal preferences. This area covers development of techniques to efficiently identify the hot routes considering the different factors. Marketing companies are often interested in identifying the hot routes of a city or particular area. In [57], Li et al. proposed a new density-based algorithm named FlowScan to discover hot routes in a road network. Instead of clustering the moving objects, they applied the clustering on road segments based on the density of common traffic they share.

[1]https://twitter.com/
[2]https://foursquare.com/

Trajectory clustering is a similar problem. Lee et al. proposed a partition and group framework for clustering trajectories to partition a trajectory into a set of line segments and then group similar line segments together into a cluster. A work on online hot route discovery [62] searches and maintains hot motion paths that are travelled by at least a certain number of moving objects. In [58], Chen et al. proposed a method to discover popular routes from trajectories. They designed a *coherence expanding* algorithm to retrieve a transfer network from raw trajectories to indicate all the possible movements between locations. The absorbing Markov chain model is then applied to derive the transfer probability from each node in the network, and finally the most popular routes are discovered from the transfer network using the their *maximum probability product* algorithm.

- **Fastest path computation**: With the emergence of digitized transportations maps and increase in our travel patterns, online journey planning has become a frequently performed and important task. Several online journey planning applications like `Google Maps`[3] provide this service. While planning a journey we are generally interested in the route that takes minimum time to reach the destination. Therefore fastest path computation is on of the most important and widely studied problems. Efficient retrieval of the path is a major concern, as the results have to be returned in real-time. In [59], Gonzalez et al. proposed an adaptive fastest path algorithm, that efficiently uses a large set of historic traffic data to mine the important driving and speed patterns, and make the algorithm adaptive to these patterns. The authors in [63] perform an experimental study on routing algorithms and acceleration methods for point-to-point shortest path computations in time-dependent directed graphs in which the link weights vary over a period of time. They found that the A* is the fastest algorithm with an enhanced heuristic estimate. It is up to 400 times faster than Dijkstras original algorithm on short routes, and the speed up compared to Dijkstra's algorithm diminishes with the increase in the length of the route. Some more interesting works on this problem are [64–68].

[3]https://maps.google.com.au

- **Mining interesting locations**: The tour and travel is a big commercial industry. Several companies like `tripadvisor`[4] are often interested in identifying the interesting destinations in a country or at a particular time. To boost the tourism industry, sometimes the government tourism departments also are interested in mining these locations. Zheng et al. [60] proposed to identify interesting locations and classic travel sequences in a give geospatial region by mining multiple users GPS trajectories. Their method uses a tree-based hierarchical graph (TBHG) to model the user location histories and is based on HITS (Hypertext Induced Topic Search) ranking model. In [69], Majid et al. developed a system to recommend interesting tourist locations and travel sequences from a collection of geo-tagged photos from platforms like `flickr`[5]. Their method takes into account the environment context including time, date and weather, and also the collective wisdom of people. The paper [70] exploits massive amounts of GPS records of multiple individual users for identifying the top-k significant semantic locations, e.g., shopping malls, restaurants, or tourist attractions.

- **Socio-spatial data**: The knowledge discovery from socio-spatial databases is a new and rapidly growing research area, having many unexplored problems [71]. Some of the works done in the past include query processing [71–73], user location based personalized recommendation [74–76], and crowd management [77, 78]. In socio-spatial recommendation research, the system learns users' interests from their location history and uses them along with the social data to make recommendations. In [74, 76], a user-location matrix is generated based on user location histories, where each entry in the matrix is the number of visits of a particular user to a physical venue. Then a user's interest on unvisited venues is inferred by following a collaborative filtering method. Cosine similarity is used to determine user similarities. The system of Bao et al. [75] follows two stages, offline modeling and online recommendation, to recommend top-$\boldsymbol{k}$ ranked location to a user. The offline modeling considers personal preferences and infers the expertise of each user in a city with respect to different category of locations, whereas the online recommendation selects candidate local experts in a geospatial range that matches the users preferences and then rank the

[4]https://www.tripadvisor.com.au/
[5]https://www.flickr.com/

locations based on an inferred score. Lee et al. [77, 78] worked to develop a geo-social event detection system by monitoring crowd behaviors indirectly via twitter. The geographical regularities are deduced from the usual behavior patterns of crowds with geo-tagged microblogs. The unusual geo-social events are detected by comparing the regularities with the estimated patterns. Li and Chen [79] performed quantitative analysis of the socio-spatial data. Zhang et al. [80] determined user influence based on spatial and social criteria.

For any kind of query processing or knowledge discovery, just like the spatial databases, it needs an efficient data indexing structure. Yang et al. [72] proposed Social R-tree as a new index structure and used it for socio-spatial group query (SSGQ) processing. The query finds the set of most suitable candidates for a planned activity by taking into account certain spatial and social constraints. Liu et al. [73] proposed k-Geo-Social Circle of Friend Query (k-gCoFQ), in which for a given a weighted graph, a user u, and a positive integer k, the query finds the group g of k + 1 users, which is connected, contains u, and minimizes the maximum distance between any two of its members. Both SSGQ and k-gCoFQ are NP-Hard, and their authors present approximation algorithms. In [71], the authors proposed two basic queries, range friends (RF) and nearest friend (NF), and one novel query, nearest start group (NSG). RF finds the friends of a user within a given geographic range, NF finds the nearest friends of a user within a given range, and NSG, for a given geographic query point $\boldsymbol{q}$ and an integer $\boldsymbol{m}$, finds a user group of size $\boldsymbol{m}$ which star subgraph of the social network and minimizes the aggregate distance of its members to $\boldsymbol{q}$.

In addition to the above recent works dealing with spatial traffic data, there are many other similar knowledge discovery problems being studied these days in different perspectives.

2.3 Clustering on Spatial Data

In clustering general data objects, there is only one objective, and it is to group the objects that have have similar properties. But when clustering is applied on spatial data, the associated spatial properties bring in some spatial constraints. Therefore in most of the problems related to clustering spatial data or partitioning spatial networks, some additional technical methods need to be developed on top of the traditional clustering algorithms. In the following sections, we present a review of the works related to spatial clustering and spatial network partitioning.

2.3.1 Spatial Clustering

Spatial clustering is an important component of spatial data mining, which groups similar spatial objects into classes using basic clustering algorithms [81–83]. In the recent years, it has been studied from different perspectives for different kinds of data including spatial trajectories, traffic data, and spatial streaming data. It is an important component in mining different traffic patterns. Trajectory clustering has remained an important problem for mining people movement patterns [55, 84]. In [55], Lee et al. presented a partition-and-group framework for clustering trajectories. It starts with optimally partitioning each trajectory into a set of line segments using the minimum description length (MDL), and then the grouping phase groups the similar line segments into clusters using a density based clustering method. Recently Hung et al. [84] used clues based on movement behavior to cluster similar trajectories into groups, which leads to find partial trajectory routes. Thereafter they do a clue-aware trajectory aggregation to derive the complete trajectory pattern and route. *FlowScan* proposed by Li et al. in [57] finds the hot routes in a road network by clustering the road segments based on the density of commonly shared traffic. The authors in [85] proposed an efficient incremental algorithm to cluster the spatial data streams collected from sensors. Their method first predicts the clusters roughly using the previous clustering results, and then refines them further in the next stage. In [86], the authors discover the spatial co-clustering patterns in traffic collision data by identifying the sets of non-spatial attribute-value pairs of collision data, e.g., weather conditions and day of the week, that together contribute significantly to the spatial clustering of corresponding collisions.

2.3.2 Spatial Network Partitioning

Spatial network partitioning refers to the task of partitioning a spatial network into multiple disjoint sub-networks or partitions in such a way that the partitions have homogeneous properties inside them. In the context of urban road networks, this task is specially important to identify the differently-congested partitions in an urban area. These partitions, also regarded as clusters of congestion in the network, help us to understand the congestion spread at a point of time in an informative way.

Though graph partitioning in general has been well studied, not much work have been done on the spatial partitioning of road networks. In [61], the authors proposed a normalized cut based method for spatial partitioning of transportation networks. They tried to achieve three predefined criteria of small variance of within-partition traffic density values, small number of partitions, and spatially near-compact partitions. Their method starts by excessive partitioning of the road network using normalized cut, followed by merging smaller partitions, and then locally adjusting the road segments lying on partition boundaries by replacing them into the neighboring partitions. It works well for small road networks, but suffers from high time and space complexities for large networks. In [87] the authors proposed two heuristic methods to partition the road network into a set of subnetworks that are balanced in terms of their size. The first method follows a recursive approach to find the sparsest cuts that lead to balanced partitions in terms of their size. The second method applies a greedy-based coarsening iteratively along the high-flow links, and terminates when the number of nodes in the coarsened network is equal to the required number of partitions. The authors have reported that their method works fine for small-sized networks, but the running time increases significantly with the increasing the network size because of the expensive computations in determining the optimal maximum concurrent flow (MCF).

There exist some other works that treat spatial network partitioning as a secondary problem to solve some other problem of primary concern. Some works suggest to partition the network into small subnetworks and use distributed computing in parallel to efficiently solve different transportation related problems in large road networks [88]. The authors in [88] used existing graph partitioning techniques to form a hierarchy of nodes in a spatial network and proposed an index structure called a *partition tree* that can be used for efficient spatial query processing. Some works partition the road networks in a way that suits their application, including monitoring proximity relations [89], point to point shortest path query indexing [90], traffic prediction [91], and finding distance-preserving subgraphs [92]. In [93], the authors partition a sensor network such that the data dissimilarity between any two nodes inside a partition is at most $\boldsymbol{\delta}$. They proposed a distributed clustering algorithm called *ELink* that works for both synchronous and asynchronous networks.

2.4 Dynamic and Evolving Networks

While there has been significant research on static networks, the research on dynamic and evolving networks is still in its early phase. In today's world, where people travel more frequently than ever before, dynamic traffic networks are becoming an important area of study. In these networks, the spatial road network is static but the continuously changing traffic makes it dynamic. Social networks are the other popular dynamic networks that have received significant attention. The common problem in dealing with all such dynamic networks is the limited ability to efficiently capture and process the frequent change in the network. Below we discuss on the developments in two specific area related to these networks.

2.4.1 Evolution of Clusters in Dynamic Networks

Traditional clustering methods directly apply only on static data. With the growth of dynamic data and dynamic networks, the attention in the recent years diverted towards developing clustering methods that can handle the changing behavior of the data. A naive way to deal with such dynamic data is by applying the traditional clustering algorithms on the dataset repeatedly whenever there is a change. The major drawback of this approach is that the algorithm has to perform all the computations again from the scratch each time, even when the change is very small. For clustering large data or partitioning large graphs, it may take very long execution times. This property makes the traditional clustering algorithms unsuitable to deal with dynamic data in real time. In dynamic evolving data, the change is generally a gradual process, and small at a time. For example, in road networks, the traffic volume on a road segment can not become very high all of a sudden. Rather, the traffic will gradually move in from connected road segments. Similarly in social networks the relationships and their strengths grows and shrinks gradually in the global perspective. The concept of *incremental clustering* is a solution for such environments, which reuses the already available information in the form of clusters from the previous time point. It looks into the most recent change, and heuristically updates the clusters or partitions based on the change. Thus it avoids all the unnecessary computations. Incremental clustering techniques are known to be more efficient than the traditional clustering algorithms, and this is achieved at the cost of sacrificing the quality of results up to some extent.

Evolution of specific events or characteristics in complex dynamic networks is studied in different kinds of networks from different perspectives [61, 94, 95]. These events in the networks are often captured in the form of clusters of the network. The temporal evolution of traffic congestion is an important problem for a smartly managed transportation, but has not been studied very well. Existing works consider an urban road network as a set of differently congested individual subnetworks, which are identified by partitioning the road network [61]. [95] investigates the spatiotemporal relation of congested road segments, and performs an empirical observation on the propagation of congestion. In [85] the authors incrementally cluster the spatial data streams collected from sensors. They firstly predict the clusters roughly using the previous clustering results, and then refine them further in the next stage. Li et al. [86] discover the spatial co-clustering patterns in traffic collision data by identifying the sets of non-spatial attribute-value pairs of collision data, e.g., weather conditions and day of the week, that together contribute significantly to the spatial clustering of corresponding collisions.

These days the evolution of events is increasingly being studied in the domain of social networks [96–99]. In [96], the authors incrementally update the evolving clusters in a dynamic network by doing bulk updates, with the aim to track the evolution of events in social networks. [99], [97] focus on identifying the emerging stories from social networks in real-time. [99] considers the network of keywords from the social texts, from which the dense clusters are identified based on approximate quasi cliques and temporally maintained. [97] considers a network with streaming edge weight updates, and with the help of an index they maintain the dense subgraphs. Community evolution in social networks is generally tracked with the help of different events in community development, including birth, death, merge, split, expansion and contraction [100]. The social networks are highly dynamic in nature, and the graph representations on this platform continuously change in both the nodes and links. Unlike these networks, the road networks are static in terms of its topology, with the continuously changing bi-directional traffic. These differences make our problem different than that studied in social networks.

2.4.2 Influence Propagation

Identification of important nodes in a network is a problem common to many different kinds of networks [101, 102]. Some application examples are identification of influential persons in social networks, key components in different information networks, key infrastructure nodes in Internet, and important road segments in transportation networks. Centrality measures including *degree centrality*, *closeness centrality*, and *betweenness centrality* give a good indication of important nodes based on the structural properties of a graph [103–105]. These measures have been found quite effective and popular in social networks. In the recent years, information propagation and diffusion has been a hot area of research in social networks [106–108]. Kempe et al. [107] devised a greedy discrete-optimization model to maximize the spread of influence through social network. Gruhl et al. [106] modeled information diffusion through blogosphere as an infectious process among users. Several existing works focus on estimating the user influences in different types or platforms of social networks [109–112]. Song et al. proposed InfluenceRank to identify opinion leaders in blogosphere based on the users importance and the novelty of the content being diffused. Goyal et al. [110] devised various probabilistic models of influence between users in social networks and also showed that influence is genuinely happening in real-world social network. Silva et al. [111] proposed an information diffusion model called ProfileRank to identify influential users and content relevance based on random walks over a user-content graph. Recently, Herzig et al. [112] devised an influence model to detect topic-based influencers in social media. Information diffusion and influence propagation have mostly been studied as applications in social networks. Though some of their fundamental concepts can be directly applied, not much work have been done on road traffic networks for an in-depth study.

2.4.3 Traffic Congestion in Transportation Networks

Transportation networks are physical spatial networks, in which the network is static but the traffic is dynamic. Crowd management and dealing with traffic congestions are problems of great importance in transportation networks [77, 113, 114]. Wang et al. [113] developed an interactive system for visual analysis of urban traffic congestion based on GPS trajectories. They clean the trajectories and match them to the road network, based on which they detect the traffic jams. They also studied the traffic-jam propagation using graphs over a period of time. Their work mainly shows congestion on individual road segments having no concrete information about how the congestion is linked in the whole network globally. Traffic congestions are known to often originate from bottleneck areas. Bottlenecks are those sections of a road network where the traffic supply capacity is lower than the traffic demand. Bottlenecks could be either in static form (e.g., on-ramp, off-ramp, curve of road, and lanes merging) or in dynamic form (e.g., moving bottlenecks due to low speed vehicles, incidents or traffic accidents) [115, 116]. Analysis of static and dynamic traffic bottlenecks and development of models to identify and understand their nature are crucial problems for dealing with traffic congestions and improving transportation operation [117]. There are often some unusual movement patterns on road networks, and finding such patterns leads to the area of anomaly detection [118–120]. Traffic jam detection and traffic pattern monitoring are its important applications [121]. Besides road networks, such problems are also being studied in other forms of transportation networks [122]. In [123], the authors demonstrated the applicability of cloud computing technique for using the large traffic data to provide traffic information requirements to the general public and traffic management organizations. Some recent works on data analytics to deal with traffic congestion are [124, 125]. [124] proposed an algorithm for automatic prediction of congestion at a future point of time by learning from the traffic history, so that effective measures could be taken in advance. The authors in [125] argued that the generation and propagation of congestion has a close relation with the network topology.

2.5 Our Work vs. Existing Work

In recent years, the growth in multidimensional geospatial datasets has attracted the attention of spatial database researchers to address the problems of transportation systems [58, 59]. This thesis is on spatial partitioning of road traffic networks and tracking their evolution over a period of time. There are not many works done so far in this research area. However, there exists few works that are partly related to some sections of our research. In this Section, we present the works related to our research, and show how we differ from them.

2.5.1 Spatial Partitioning of Urban Road Networks

Chapter 3 of this thesis presents a spectral clustering based algorithm for partitioning road traffic networks. On the same problem, but for a more efficient solution for large road networks at the cost of compromising with accuracy up to some extent, Chapter 4 presents a framework that is based on both density and spectral based clustering.

The most closely related work on this problem are [61, 87, 126]. Ji and Geroliminis [61] proposed a normalized cut based method for spatial partitioning of transportation networks. They tried to achieve three predefined criteria of small variance of within-partition traffic density values, small number of partitions, and spatially near compact partitions. Their method works in three steps, starting with excessive partitioning of the road network using normalized cut, followed by merging smaller partitions up to a certain level, and then locally adjusting the road segments lying on partition boundaries by replacing them into the neighboring partitions, if doing so increases the segment uniformity. Their method suffers from time and space complexity for large urban road networks. [126] showed that an efficient real-time traffic control for a large-scale urban traffic network could be done by dividing the large complex network into multiple small simple subnetworks controlled in a hierarchical structure. They proposed a partitioning method based on the community detection theory to maximize the modularity of a partition, and follow an agglomerative approach to check each possible combination and compare their modularity. It is highly computational and not scalable to check each possible combination, and therefore faces serious limitations for large networks. In [87] the authors proposed two heuristic methods to partition the road network into a set of subnetworks that are balanced in terms of their size. The first method follows a recursive approach to find the sparsest cuts that lead to balanced partitions in terms of their size. The second method applies a greedy-based coarsening iteratively along the high-flow links, and terminates when the number of nodes in the coarsened network is equal to the required number of partitions. The authors have reported that their method works fine for small-sized networks, but the running time increases significantly with the increasing the network size because of the expensive computations in determining the optimal maximum concurrent flow (MCF). [127] partitioned the transportation network to develop a macroscopic fundamental diagram (MFD) of Brisbane, Australia, based on data fusion from multiple sources.

In contrast to the works mentioned above, both of our methods (Chapters 3 and 4) follow a two level-clustering to make the algorithm scalable for large urban road networks. Our method in Chapter 3 applies $\boldsymbol{k}$-means in the first level to group the road segments locally in a bottom-up manner to form supernodes, and then applies a novel partitioning algorithm (referred as $\boldsymbol{\alpha}$-Cut) that works in a top-down manner in the second level. Similarly, our method in Chapter 4 applies density based clustering in the first level to construct a well structured and condensed density peak graph. Then in the second level it applies our $\boldsymbol{\alpha}$-Cut partitioning algorithm. The $\boldsymbol{\alpha}$-Cut algorithm helps us in achieving more accurate results, and dividing the computations into two levels, helps us in improving our efficiency.

This problem is much related to general graph partitioning, which is a well studied problem. It has applications to a wide variety of areas, and many solutions have been proposed in the past. As mentioned earlier in Section 2.1.3, spectral clustering algorithms like minimum cut and normalized cut have remained quite popular [16, 17]. In [18], the authors proposed a spectral cut based on the min-max clustering principle for graph partitioning from a data clustering point of view. In [17], White and Smith proposed a spectral clustering based solution to find communities in graphs by partitioning. Their objective function is based on network modularity.

This modularity matrix [17] (described earlier in Section 2.1.4) actually equals to the negative of our $\boldsymbol{\alpha}$-Cut matrix derived in Equation 3.10 of Chapter 3. As we obtain the partitioning by selecting the $\boldsymbol{k}$ smallest eigenvalues and corresponding eigenvectors, both the techniques result in the same set of eigenvalues and eigenvectors, and thus the same partitioning. It means that the minimization of $\boldsymbol{\alpha}$-Cut approximately maximizes the modularity.

More works related to general graph partitioning are covered in detail in Section 2.1.2.

2.5.2 Tracking and Capturing the Spatio-temporal Evolution of Congestion

The temporal evolution of traffic congestion is an important problem for a smartly managed transportation, but has not been studied very well. Existing works consider an urban road network as a set of differently congested individual subnetworks, which are identified by partitioning the road network [61]. Chapter 5 of this thesis presents a comprehensive framework to track and capture the spatiotemporal evolution of road network partitions by incrementally updating them in an efficient manner. In road networks, there exists no such work that incrementally updates the partitions for capturing the spatiotemporal evolution. [95] investigates the spatiotemporal relation of congested road segments, and performs an empirical observation on the propagation of congestion. Unlike our framework, this work is more of an experimental analysis. In [85] the authors incrementally cluster the spatial data streams collected from sensors. They firstly predict the clusters roughly using the previous clustering results, and then refine them further in the next stage. Li et al. [86] discover the spatial co-clustering patterns in traffic collision data by identifying the sets of non-spatial attribute-value pairs of collision data, e.g., weather conditions and day of the week, that together contribute significantly to the spatial clustering of corresponding collisions.

These days the evolution of events is increasingly being studied in the domain of social networks [96–99]. In [96], the authors incrementally update the evolving clusters in a dynamic network by doing bulk updates, with the aim to track the evolution of events in social networks. [99], [97] focus on identifying the emerging stories from social networks in real-time. [99] considers the network of keywords from the social texts, from which the dense clusters are identified based on approximate quasi cliques and temporally maintained. [97] considers a network with streaming edge weight updates, and with the help of an index they maintain the dense subgraphs. Community evolution in social networks is generally tracked with the help of different events in community development, including birth, death, merge, split, expansion and contraction [100]. The social networks are highly dynamic in nature, and the graph representations on this platform continuously change in both the nodes and links. Unlike these networks, the road networks are static in terms of its topology, with the continuously changing bi-directional traffic. These differences make our problem different than that studied in social networks.

Chapter 5 of this thesis also presents an in-memory index structure called `BIN` to compactly store the historical congestion-related information in the form of small subgraphs (a set of *building blocks*) generated by the method. The storage of historical graph data is an old and well studied problem [128]. There has been a lot of works on temporal relational databases [129] [130]. With the rise of temporally evolving networks and problems related to efficient snapshot retrieval, index structures have also become important for main memory-resident graphs [131]. However, none of the prior works focus on memory-resident structures for maintenance of the road network partitions.

2.5.3 Applications using Real Traffic Data

Chapter 6 of this thesis presents an experimental study to discover the congestion propagation patterns. There are two specific applications considered. First one is the temporal tracking of congested partitions, and the other one is the traffic diffusion and influence estimation.

The first application of temporal tracking of congested partitions is much related to our work presented in Chapter 5. However, the objective here is to analyze the real data to discover the unknown traffic propagation patterns. We defined certain measures to identify the congested partitions, and track their evolution with time. One closely related work is that of Wang et al. [113]. They developed an interactive system for visual analysis of urban traffic congestion based on GPS trajectories. They clean the trajectories and match them to the road network, based on which they detect the traffic jams. They also studied the traffic-jam propagation using graphs over a period of time. Their work mainly shows congestion on individual road segments having no concrete information about how the congestion is linked in the whole network globally. In contrast our congested partitions give the information about how congestion is spread in the network globally and how the different congested partitions are linked. In another work [95], the authors investigate the spatiotemporal relation of congested road segments, and performs an empirical observation on the propagation of congestion. The `Google Traffic` service on `Google Maps`[6] anonymously collects the traffic data from different sources mostly from a large number of mobile phone users. Based on this data, it visualizes the traffic condition using a simple heat map in real time. The information here again is local to the referred road segment. In contrast our congested partitions give the global information about the whole network, particularly, how the congestion is spread in the network and how the different congested partitions are linked.

The second application of traffic diffusion and influence estimation is a novel application for road networks. We could not find any existing work talking about road traffic diffusion. However, there exists the concept of information diffusion since a long time. Identifying important nodes in a network is a problem common to many different kinds of information networks [101, 102]. In the recent years, information propagation and diffusion has been a hot area of research in social networks [106–108]. Kempe et al. [107] devised a greedy discrete-optimization model to maximize the spread of influence through social network. Gruhl et al. [106] modeled information diffusion through blogosphere as an infectious process among users. Several existing works focus on estimating the user influences in different types or platforms of social networks [109–112]. Song et al. proposed InfluenceRank to identify opinion leaders in blogosphere based on the users importance and the novelty of the content being diffused. Goyal et al. [110] devised various probabilistic models of influence between users in social networks. Silva et al. [111] proposed an information diffusion model called ProfileRank to identify influential users and content relevance based on random walks over a user-content graph. Recently, Herzig et al. [112] devised an influence model to detect topic-based influencers in social media.

[6]https://maps.google.com.au

Chapter 3

Spatial Partitioning of Road Traffic Networks

The rapid global migration of people towards urban areas is multiplying the traffic volume on urban road networks. As a result these networks are rapidly growing in size, in which different sub-networks exhibit distinctive traffic flow patterns. In this chapter, we propose a scalable framework for traffic congestion-based spatial partitioning of large urban road networks. It aims to identify different sub-networks or partitions that exhibit homogeneous traffic congestion patterns internally, but heterogenous to others externally. To this end, we develop a two-stage procedure within our framework that first transforms the large road graph into a well-structured and condensed supergraph via clustering and link aggregation based on traffic density and adjacency connectivity, respectively. We then devise a spectral theory based novel graph cut (referred as $\boldsymbol{\alpha}$-Cut) to partition the supergraph and compare its performance with that of an existing method for partitioning urban networks. Our results show that the proposed method outperforms the normalized cut based existing method in all the performance evaluation metrics for small road networks and provides good results for much larger networks where other methods may face serious problems of time and space complexities.

3.1 Introduction

Traffic flow patterns in urban road networks have been found to vary significantly in different sub-networks depending on two critical factors– ***i***) *spatial* importance, and ***ii***) *temporal* importance. Usually roads of each locality, say inside a suburb or part of a suburb in a city, experience a specific traffic flow pattern regardless of the global flow. For example, roads inside the city centre or any area having popular venues like a monument or hospital, usually remain more congested than others without any such significance. Additionally, the congestion on roads connecting important places of human gathering like airports, train stations, hospitals, bus stops, etc., remains comparatively higher than others. Similarly in the temporal perspective, roads usually remain busier and more congested in peak hours (normally 7 AM to 10 AM and 4 PM to 7 PM) than off-peak hours. As the sub-networks exhibit distinctive traffic flow patterns, the traffic management decisions for each sub-network need to reflect these differences.

It urges for an intelligent and effective traffic management to make crucial decisions on issues, like flow smoothening and streamlined infrastructure deployment, which would treat the varying traffic patterns in different sub-networks accordingly. To aid in these decision making, the transport department authorities might be very keen to identify the different sub-networks or partitions which exhibit homogeneous traffic flow patterns and properties inside them locally whereas heterogeneous to others globally in the urban road network [61]. Then instead of looking into the whole network globally, each partition with homogeneous congestion patterns within can be considered as a unit, and traffic monitoring strategies can be developed individually. The identification of the differently congested partitions further aids in understanding and analyzing the congestion and its evolving nature with respect to time. These sub-networks can also be helpful for travelers to get information about the differently congested areas at a point of time. Therefore, the congestion-based spatial partitioning of urban road networks from a large data perspective is becoming a problem of growing importance.

Although there exist many other kinds of information networks and the application of graph partitioning on such networks has been studied in the past [20, 28], the geospatial properties of a road network associated with traffic flow patterns make a unique kind of network [61]. The problem was recently raised in the intelligent transportation systems (ITS) community [61]. They contend that transportation networks have unique dynamic features and an arbitrary clustering algorithm may not produce the desired kind of partitions. They tried to achieve three predefined criteria of small variance of traffic density values inside a partition, small number of partitions, and spatially *near-compact* partitions. Their normalized cut based method works in three steps, starting with excessive partitioning of the road network, followed by merging smaller partitions up to a certain level, and then locally adjusting the road segments lying on partition boundaries by replacing them into the neighbouring partitions, if doing that increases the segment uniformity.

In this chapter, we present a method for traffic-based spatial partitioning of large urban road networks. Same as [61], we do not focus much on the dynamic nature of traffic congestion in this work, and consider partitioning the network repeatedly at regular intervals of time using static traffic measures. As we focus on large road networks, scalability of the method for real-world applicability is also a major concern.

The partitioning framework is based on traffic density measures on a road network defined by the count of vehicles per unit distance on each road segment. Its objective is to identify the different heterogeneous regions of an urban network which internally exhibit homogeneous traffic congestion patterns. The method starts with transforming the actual road network into a *road graph*, which is followed by mining the *road supergraph*. Finally the supergraph is subjected to a spectral theory based novel graph cut called $\boldsymbol{\alpha}$-Cut to obtain the set of road segments partitioned into several subsets called *road network partitions*. As the partitioning algorithm is applied on a supergraph with much reduced order(number of supernodes), the framework becomes more scalable by managing the computational and space complexity. In summary, we make the following contributions in this chapter.

– We mine a well-structured and condensed road supergraph from the road graph. Its main advantage is in making the partitioning method applicable on large road networks where the number of road segments is large.

– To identify the optimal number of clusters for $\boldsymbol{k}$-means while forming the condensed supergraph, we devise a measure to help find the optimal clustering.

– We devise a spectral theory based novel graph cut called **α**-Cut to partition the road supergraph, which outperforms normalized cut in our empirical study.

– Extensive experiments are performed on both small and large road networks to establish its efficacy.

The rest of the chapter is organized as follows. Section 3.2 presents some preliminary theories followed by the problem definition. Section 3.3 presents the framework briefly. The complete methodology is described in Sections 3.4 and 3.5. Experimental results are shown in Section 3.6, followed by the chapter summary in Section 3.7.

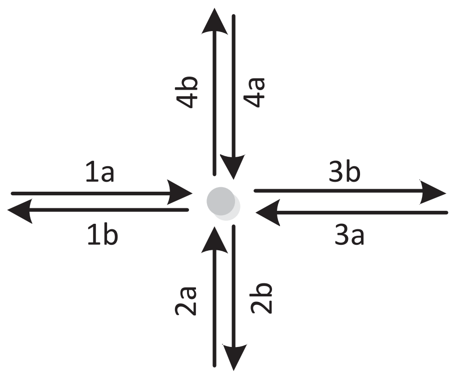
3.2 Preliminaries

In this section, we present some preliminary theories on road networks and then formulate the problem.

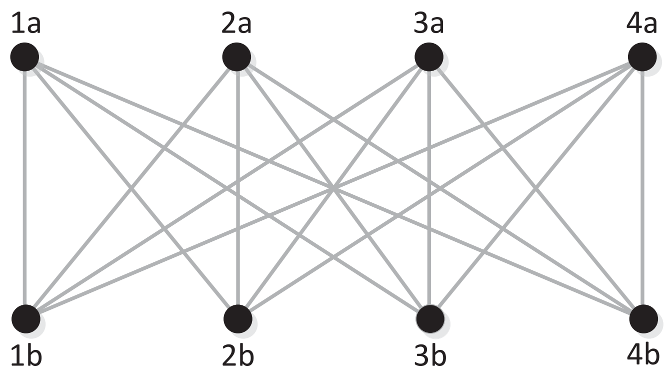
3.2.1 Road Networks and their Mathematical Representation

Urban roads exist in the form of a physical network spatially spread over a large urban area. To make it a machine-interpretable network, we need to give it a mathematical representation in the form of a graph, which we name as a *road graph.* The unique features associated with this kind of network, like varying spatial importance of different roads and the traffic flow being unidirectional on some roads whereas bidirectional on others, make it a challenging task to give a realistic mathematical representation. Previous works have represented it in different graph-based structures that suited the application area [42, 59].

Unlike the previously attempted problems, the focus of spatial partitioning of road networks is on the road segments, not on the intersection points. A trivial representation in the form of a graph by considering roads as links and their intersection points as nodes is not suitable as its partitioning results into subsets of intersection points, which is not the objective. To make the representation applicable to spatial partitioning, we transform the actual road network into its dual, which forms an undirected road graph. This transformation is an improved version of that used in our earlier work [3].



(a) Intersection point



(b) Bipartite formation

FIGURE 3.1: Star topology to bipartite formation

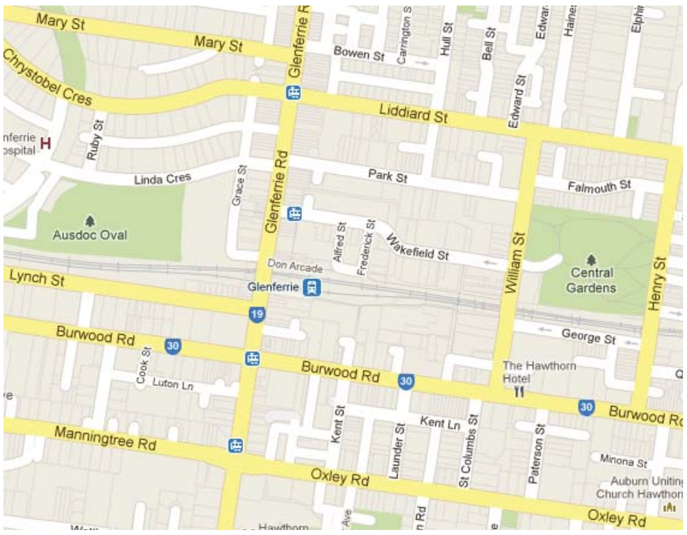
Definition 3.1. (**Road Network**) A real urban road network is defined as $\mathcal{N} = (\mathcal{I}, \mathcal{R})$ comprising a set of intersection points $\mathcal{I} = \{\iota_1, \iota_2, \ldots, \iota_{n_\iota}\}$ as nodes that are connected among themselves by the set of directed road segments $\mathcal{R} = \{r_1, r_2, \ldots, r_{n_r}\}$ as its links, where each road segment r_i associates the traffic density $r_i.d$ with itself. ■

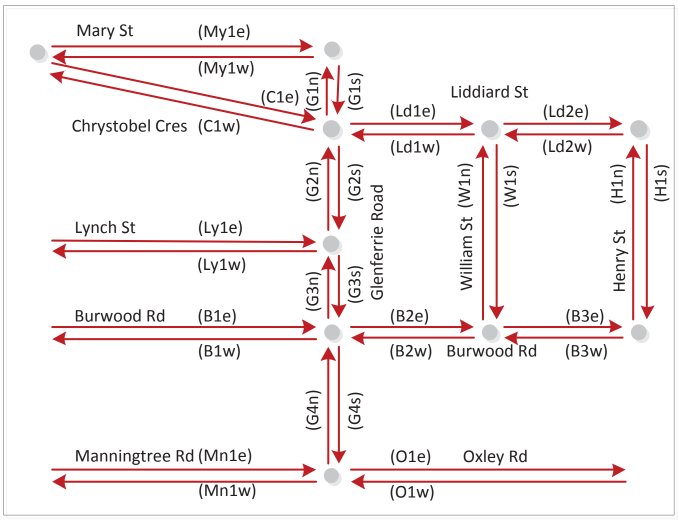
Definition 3.2. (**Road Graph**) Given a road network $\mathcal{N}$, the corresponding road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed by adding each road segment $r_i \in \mathcal{N}$ as a node v_i, and establishing an undirected link e_i between each possible node pair (v_j, v_k) if there exists at least one intersection point ι_l which is a common intersection for the roads r_j and r_k, and the traffic can flow either from r_j to r_k or vice versa, as shown in Equation 3.1.

$$\begin{aligned}\boldsymbol{\mathcal{V}} &= \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_{n_r}}\}\text{, where } \boldsymbol{v_i} = \boldsymbol{node}(\boldsymbol{r_i}) \\ \boldsymbol{\mathcal{E}} &= \{\boldsymbol{link}(\boldsymbol{v_j}, \boldsymbol{v_k}) : \exists\, \boldsymbol{\iota_l} \text{ as the commnon intersection} \\ &\quad \text{point for } \boldsymbol{r_j} \text{ and } \boldsymbol{r_k}\}\end{aligned} \tag{3.1}$$

Thus the links stand for the adjacency relationships among the road segments. In this manner, the road network components in a star topology form bipartites in the road graph, as shown in Figure 3.1, where each partite stands for either incoming flow to or outgoing flow from a common intersection point. Each node $\boldsymbol{v_i}$ $(\boldsymbol{node}(\boldsymbol{r_i})) \in \boldsymbol{\mathcal{V}}$ associates with it a feature value $\boldsymbol{v_i.f}$ which is the road traffic density $\boldsymbol{r_i.d}$. ■

Most urban roads exist as two-way roads, which are two oppositely directed one-way parts separated from each other. Each of the two parts (directions) undergo different kinds of traffic flow patterns. For example, in the morning office hours, a road that connects outskirts with the city center would find more traffic heading towards the city center than the opposite direction. This feature of the urban network is accommodated in Definition 3.2 by considering the two traffic directions as separate road segments, if they share a common intersection point and thus are adjacent. Figure 3.2 shows an example of our road network representation in which Figure 3.2(a) is a sample road map, Figure 3.2(b) is the corresponding road network of those colored yellow in the map, and Figure 3.2(c) is the final representation called the road graph. When representing any kind of network in the form of a graph, normally the main objects of study in the network are considered as nodes and the links define the affinity between them. In the road network in Figure 3.2(b), we can see that nodes represent the intersection points of roads, which actually do not have much importance for the problem of spatial partitioning as compared to roads, which appear as links. The road graph in Figure 3.2(c) solves this problem as the objects of study are represented as nodes.


(a) Actual road map



(b) Road network

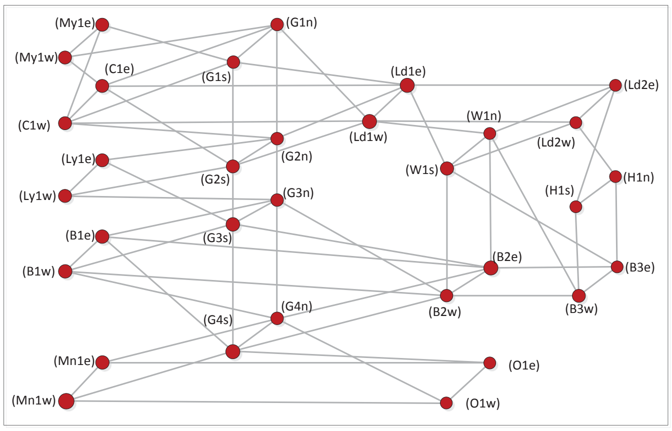

(c) Road graph

FIGURE 3.2: Mathematical representation of road networks

3.2.2 Problem Definition

The problem of *spatial partitioning* of large urban road networks is defined as splitting up a given large urban road network based on traffic congestion measures into several disjoint partitions, keeping intact the associated spatial properties. The different partitions exhibit the property of intra-partition congestion *homogeneity* and inter-partition congestion *heterogeneity.* Let us suppose we have a real urban directed road network $\mathcal{N}$, which is transformed into a *road graph* $\mathcal{G}$ by following the method described in Section 3.2.1. This graph is then subjected to congestion-based spatial partitioning. Before formally stating the problem on $\mathcal{G}$, we present three definitions.

Definition 3.3. (**Cost of Partitioning**) While partitioning the set of nodes $\mathcal{V}$ in a road graph $\mathcal{G}$ into different partitions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, the *cost of partitioning* is defined as the aggregation of *affinity values* of all possible node pairs (v_i, v_j) for which v_i and v_j lie in different partitions in the final result, where the affinity values are a measure of congestion similarity between the pair of nodes. ■

Definition 3.4. (**Partition Volume**) Given a set of road graph partitions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, *partition volume* is defined as the aggregation of *affinity values* of all possible pairs (v_i, v_j) for which v_i and v_j lie in the same partition, where the affinity values are a measure of congestion similarity between the pair of nodes. ■

Definition 3.5. (**Partition Connectivity**) A partition $\mathcal{P}_l$ is said to be *connected* if for any given node pairs $(v_i, v_j) \in \mathcal{P}_l$ there exists a path from v_i to v_j or vice versa. ■

The problem of congestion-based spatial partitioning of a road graph $\mathcal{G}$ is to split its node set $\mathcal{V}$ into k partitions (or subsets) $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ such that the following conditions hold.

C.1 $\bigcup_{i=1}^{k} \mathcal{P}_i = \mathcal{V}$ and $\mathcal{P}_i \bigcap \mathcal{P}_j = \phi$ for all $i \neq j$;

C.2 each $\mathcal{P}_i$ is *connected* and all adjacency relations, except the cross-partition relations, are maintained as in $\mathcal{G}$;

C.3 the *cost of partitioning* of $\mathcal{G}$ is the minimum; and

C.4 the *partition volume* of $\mathcal{G}$ is the maximum.

In the above conditions, **C.1** is a general condition of grouping the set of nodes (or road segments) into $\boldsymbol{k}$ non-overlapping subsets, **C.2** introduces the spatial connectivity (or linkage) of nodes, **C.3** enforces the condition of inter-partition traffic congestion heterogeneity, and **C.4** enforces intra-partition traffic congestion homogeneity. A partitioning may not satisfy **C.3** and **C.4** together simultaneously, and therefore the best possible trade-off has to be found.

3.3 Framework

The task of road network partitioning can also be viewed as similar to that of clustering road segments based on their traffic density values. However, the main drawback of this approach is that traditional clustering algorithms do not take care of the associated spatial connectivities (connectivity of road segments). Consequently we treat it as a 2-level partitioning problem, in which the first level follows a bottom-up approach considering only data in the form of density values, whereas the second level follows a top-down approach considering both the density data along with the road segment connectivities.

The complete framework for spatial partitioning of road networks, shown in Figure 3.3, comprises three different modules– $\boldsymbol{i}$) road graph construction, $\boldsymbol{ii}$) road supergraph mining, and $\boldsymbol{iii}$) supergraph partitioning. The first module deals with transforming the real road network $\boldsymbol{\mathcal{N}}$ into a road graph $\boldsymbol{\mathcal{G}}$ to give it a mathematical representation, which is explained as a preliminary step in Section 3.2.1. Due to the large and rapidly expanding urban area, the size of an urban road network $|\boldsymbol{\mathcal{R}}|$ and the order of the corresponding road graph $|\boldsymbol{\mathcal{V}}|$ may become extremely large, which heavily affects the computational and space complexity for partitioning $\boldsymbol{\mathcal{G}}$. To address this problem, the framework follows a 2-level partitioning.

The first level (which is the second module described in Section 3.4) mines a road supergraph $\boldsymbol{\mathcal{G}_s}$ from the road graph $\boldsymbol{\mathcal{G}}$ with a much reduced order following a bottom-up approach. It goes through the steps of clustering feature values $\boldsymbol{v_i.f}$ using $\boldsymbol{k}$-means in Section 3.4.1 and constructing the road supergraph in Section 3.4.3.

Furthermore, an extended version of this module introduces the concept of *stable* supernodes. A supernode is considered to be *stable* if its *stability* measure, defined later, is above

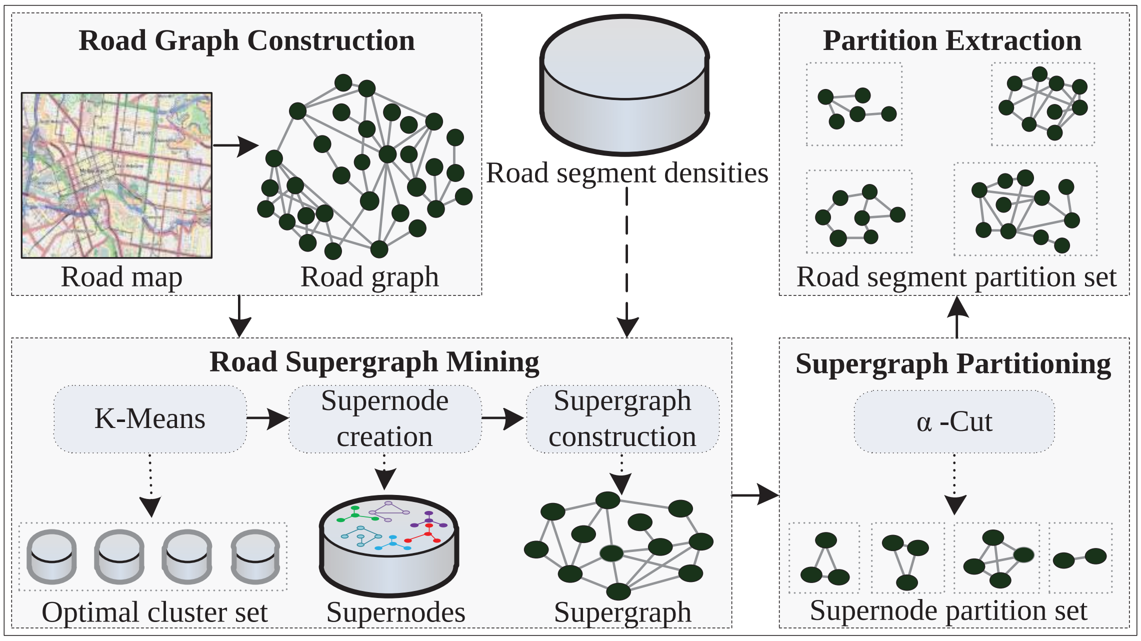


FIGURE 3.3: Proposed spatial partitioning framework

a predefined threshold. To have a *stable* supergraph, all the supernodes that are found unstable are further split up, which is repeated until they become stable. We can have the supergraph of different structures as per the application environment by varying the stability threshold. The stability threshold scale is also a trade-off between complexity and accuracy. A lower threshold value reduces the complexity by reducing the supergraph *order* while sacrificing some level of accuracy by presuming all nodes inside a supernode to belong to the same final partition. On the other hand a higher value can give more accurate results at the cost of computational and space complexity.

The last module of supergraph partitioning, described in Section 3.5, is the second level partitioning that follows a top-down approach to split up the supergraph into multiple heterogeneous partitions that are homogeneous within. It is achieved by approximately optimizing a measure called ***α-Cut***, by following a spectral clustering based solution. It produces supernode partitions, from which the road segment partitions are extracted.

3.4 Road Supergraph Mining

A naive approach to obtain road network partitions is to apply the partitioning algorithm on the road graph directly. However, we reduce the load of partitioning by following a 2-level partitioning. The first level mines a condensed *road supergraph*, before partitioning it in the second level.

The road segments inside a road sub-network or partition are linked together. Any vehicle entering into a partition through a road segment needs to go through the following segments to cross the partition or reach the destination. It makes the congestion pattern of a segment more likely to be similar to (or dependent on) other (following or preceding) segments inside the partition. Thus the similar spatial importance of road segments within a partition leads them to exhibit similar congestion patterns. That means if they are grouped based on their traffic density measures, most of the time, they could be expected to be grouped together. To capture this aspect, before applying the partitioning algorithm we group the segments based on their traffic density measures to find their clustering pattern, and use them to construct a condensed road supergraph. The following definitions present the idea of a supergraph used in this chapter.

Definition 3.6. (**Supernode**) Given a road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *supernode* ς_i having a feature value $\varsigma_i.f$, is defined as a set of nodes $\{v_j\}$ that exhibit the properties of being similar in terms of their feature values $\{v_j.f\}$ (density measures), and interlinked together. ■

Definition 3.7. (**Superlink**) Given a road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *superlink* ε_i is defined as a link between a pair of supernodes $(\varsigma_p, \varsigma_q)$, which exists only if there is at least one link $link(v_x, v_y) \in \mathcal{E}$ such that $v_x \in \varsigma_p$ and $v_y \in \varsigma_p$. ■

Definition 3.8. (**Road Supergraph**) Given a road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *road supergraph* $\mathcal{G}_s$ is defined as an ordered 3-tuple $(\mathcal{V}_s, \mathcal{E}_s, \mathcal{W}_s)$, where $\mathcal{V}_s = \{\varsigma_1, \varsigma_2, \ldots, \varsigma_{n_\varsigma}\}$ is the set of supernodes comprising the set of road segments, $\mathcal{E}_s = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{n_\varepsilon}\}$ is the set of superlinks, and $\mathcal{W}_s = \{\omega_1, \omega_2, \ldots, \omega_{n_\varepsilon}\}$ is the set of weights associated with each of the corresponding superlinks, which is defined as a measure of congestion similarity between the pair of supernodes connected by the superlink. ■

The task of mining the supergraph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s, \mathcal{W}_s)$ is done in two steps. The first step deals with the feature values $v_i.f$ associated with each node in $\mathcal{G}$ to group them into different clusters, whereas the second step uses these clusters to construct the supergraph. The complete algorithm is shown in Algorithm 1. The popular clustering algorithm k-means is used to cluster the feature values associated with the node set. A major problem with k-means is its requirement for a pre-determined number of clusters. We overcome this problem by designing a novel optimality measure called *moderated clustering gain* (MCG) to determine the optimal number[1] of clusters κ for a dataset (line 6). Instead of considering the optimal value of κ, we consider all those κ for which the MCG value lies above a threshold (lines 3–9), and the one that produces the least number of supernodes is finally selected as optimal (lines 10–16). After creating the supernodes and assigning their feature values as cluster means from the optimal cluster set (lines 17–20), the superlinks in between the supernodes are established and weighted (lines 21–25) to construct the supergraph (line 26).

3.4.1 Feature Value Clustering

This step looks into the feature values $v_i.f$ associated with the nodes in $\mathcal{G}$ without considering its adjacency relationships or connectivities with the intent to get a rough idea of the partitions, which are refined in subsequent steps to find the actual partitions. Let $\mathcal{F} = \{v_1.f, v_2.f, \ldots, v_{n_r}.f\}$ be the set of feature values associated with the set of nodes $\mathcal{V}$ in $\mathcal{G}$. The objective is to extract information about grouping patterns of the feature values, and therefore $\mathcal{F}$ is treated with the k-means clustering algorithm. It results in an organization of feature values in the form of clusters. There are a few limitations of k-means, and one of them is that it may result in a clustering configuration having a local maxima that may not be the global maxima. The outcome depends on the initialization of cluster means. As we have the feature values in a single dimension, we overcome this limitation by firstly sorting the feature values $v_i.f \in \mathcal{F}$ and then initializing the cluster means with feature values at equal intervals. That means, when we have n_r number of

[1] We use the Greek lowercase letter kappa (κ) to refer the number of clusters produced by k-means in Section 3.4.1, and the English lowercase letter k to refer the number of partitions produced by the framework.

Algorithm 1: Road supergraph mining (Road graph $\mathcal{G}$, optimality threshold ϵ_θ)

```
1  $A_\mathcal{G} \leftarrow$ adjacency matrix of $\mathcal{G}$;
2  $\mathcal{F} \leftarrow \{v_1.f, v_2.f, \ldots, v_{n_r}.f\}$;
   // shortlist cluster sets based on MCG threshold $\epsilon_\theta$
3  $\varrho \leftarrow \phi$;
4  for $\kappa \leftarrow 2$ to $(n_r - 1)$ do
5      $(\mathcal{C}^\kappa, \mu(\mathcal{C}^\kappa)) \leftarrow k\text{-}means(\mathcal{F}, \kappa)$;
6      $\Theta(\mathcal{C}^\kappa) \leftarrow$ MCG of $\mathcal{C}^\kappa$;
7      if $\Theta(\mathcal{C}^\kappa) \geq \epsilon_\theta$ then
8          $\varrho^\kappa \leftarrow$ cluster indicator vector of $\mathcal{C}^\kappa$;
9          $\varrho \leftarrow \varrho \cup \varrho^\kappa$;
   // select the optimal cluster set
10 $\varrho^\theta \leftarrow \phi$, $\mathcal{C}^\theta \leftarrow \phi$;
11 $min \leftarrow$ number of connected components in $(\varrho^1, A_\mathcal{G})$;
12 forall $\varrho^\kappa \in \varrho$ do
13     $comp \leftarrow$ number of connected components in $(\varrho^\kappa, A_\mathcal{G})$;
14     if $min > comp$ then
15         $min \leftarrow comp$;
16         $\varrho^\theta \leftarrow \varrho^\kappa$, $\mathcal{C}^\theta \leftarrow \mathcal{C}^\kappa$;
   // create supernodes and assign their feature values
17 $\mathcal{V}_s \leftarrow createSupernodes(\varrho^\theta, A_\mathcal{G})$;
18 forall $\mathcal{C}_i^\theta \in \mathcal{C}^\theta$ do
19     forall $\varsigma_j \in \mathcal{C}_i^\theta$ do
20         $\varsigma_j.f \leftarrow \mu(\mathcal{C}_i^\theta)$;
   // establish superlinks and assign their weights
21 $\mathcal{E}_s \leftarrow \phi$, $\mathcal{W}_s \leftarrow \phi$;
22 forall $link(v_x, v_y) \in \mathcal{E}$ do
23     if $v_x \in \varsigma_p$ and $v_y \in \varsigma_q$ and $p \neq q$ then
24         $\mathcal{E}_s \leftarrow \mathcal{E}_s \cup establishLink(\varsigma_p, \varsigma_q)$;
25         $\mathcal{W}_s \leftarrow \mathcal{W}_s \cup assignWeight(\varsigma_p, \varsigma_q)$;
26 $\mathcal{G}_s \leftarrow (\mathcal{V}_s, \mathcal{E}_s, \mathcal{W}_s)$;
27 return $\mathcal{G}_s$
```

sorted feature values, the mean of the jth cluster is initialized by $v_i.f$, where $i = \frac{n_r}{\kappa} \times j$, and the following steps remain the same as the standard k-means algorithm.

As stated earlier, k-means needs to have a predetermined number of clusters that has to be provided as input to the algorithm. We address this problem by applying k-means repeatedly with different values of κ producing the set of clusters as $\mathcal{C}^\kappa = \{\mathcal{C}_1^\kappa, \mathcal{C}_2^\kappa, \ldots, \mathcal{C}_\kappa^\kappa\}$. It starts with $\kappa = 2$ incrementally and at each value an optimality test is performed. The optimality test compares the MCG measure, described in Section 3.4.2, with that computed in the preceding iteration at $\kappa - 1$ and the following iteration at $\kappa + 1$. The point where the value found is higher than both its preceding and following points, represents a *local optimality maxima.* However there is no guarantee that it will serve as the *global optimality maxima.* Applying k-means repeatedly on a large dataset just to learn the optimal number of clusters makes the method computationally very expensive, particularly in situations when the dataset is extremely large. To overcome this problem, repetitive clustering is applied on a randomly generated sample dataset, much smaller than the actual dataset. Let θ be the value of κ that produces the clustering configuration having the global optimality maxima. Instead of considering only the clustering configuration at θ, we consider all values of κ for whom MCG lies above a predefined threshold ϵ_θ, which are passed on to the next step to create supernodes in Section 3.4.3.

The computational complexity of k-means is $\mathrm{O}(tnd\kappa)$, where t is the number of iterations needed to converge, n is the number of data items, d is the data dimension, and κ is the number of required clusters. In our case, $d = 1$, which makes it $\mathrm{O}(tn\kappa)$. Also κ is usually very small.

3.4.2 Optimality Measure

In this section, we design an optimality measure to learn the optimal number of clusters θ suitable for a data set. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_{(n_d)}\}$ be the dataset consisting of n_d data items, where each d_i has m_d feature values forming an m_d-dimensional vector $\langle f_1, f_2, \ldots, f_{(m_d)}\rangle$. The global mean μ^0 is a vector given by $\langle \mu^0_1, \mu^0_2, \ldots, \mu^0_{(m_d)}\rangle$ where $\mu^0_p = \frac{1}{n_d}\sum_{i=1}^{n_d} d_i.f_p$ corresponding to each feature f_p. Let $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\kappa\}$ be the set of clusters generated by k-means, then the mean μ^q of each cluster $\mathcal{C}_q$ is a vector of feature means given by $\langle \mu^q_1, \mu^q_2, \ldots, \mu^q_{(m_d)}\rangle$ where $\mu^q_p = \frac{1}{|\mathcal{C}_q|}\sum_{d_i \in \mathcal{C}_q} d_i.f_p$ corresponding to each feature f_p. A measure called *clustering balance* $\mathcal{E}(\mathcal{C})$, defined in [132], has been found to be a good indicator for the optimal number of clusters, outperforming previous measures. Another comparable and computationally efficient measure called *clustering gain* $\Delta(\mathcal{C})$ is defined in the same work. The optimal clustering configuration is achieved at that value of κ where the clustering balance reaches its minimum, whereas clustering gain reaches its maximum. Although these measures were proposed for identifying optimality in hierarchical clustering, they were shown to be suitable for $\boldsymbol{k}$-means too. In our analysis we found that when these measures are applied with $\boldsymbol{k}$-means they produce a smaller number of sparse clusters. In this work we extend and improve clustering gain to make the clusters compact and far apart from others, named as *moderated* clustering gain, denoted by $\boldsymbol{\Theta(\mathcal{C})}$. Shown in Equation 3.2, it is the summation of a value over all clusters $\boldsymbol{\mathcal{C}_q}$, in which the value consists of two parts multiplied by each other. The first part $\boldsymbol{\Theta_1(\mathcal{C}_q)}$ is the clustering gain, whereas the second part $\boldsymbol{\Theta_2(\mathcal{C}_q)}$ is a function of the ratio of intra-cluster and inter-cluster error sums. For each $\boldsymbol{q \in [1, \kappa]}$, the value of $\boldsymbol{\Theta_2(\mathcal{C}_q)}$ lies in the range $\boldsymbol{[0, 1]}$ and it moderates the value of $\boldsymbol{\Theta_1(\mathcal{C}_q)}$ by reducing its effect accordingly.

$$\text{MCG}, \quad \boldsymbol{\Theta(\mathcal{C}) = \sum_{q=1}^{\kappa} \left(\Theta_1(\mathcal{C}_q) \times \Theta_2(\mathcal{C}_q)\right)} \tag{3.2}$$

$$\begin{aligned} \text{where,} \quad \boldsymbol{\Theta_1(\mathcal{C}_q)} &= \boldsymbol{(|\mathcal{C}_q| - 1) \left\| \mu^q - \mu^0 \right\|_2^2} \\ \boldsymbol{\Theta_2(\mathcal{C}_q)} &= \boldsymbol{\left(1 - \log_2 \left(1 + \frac{\sum\limits_{d_i \in \mathcal{C}_q} \left\| d_i - \mu^q \right\|_2^2}{|\mathcal{C}_q| \times \left\| \mu^q - \mu^0 \right\|_2^2} \right) \right)} \end{aligned}$$

The optimal number of clusters $\boldsymbol{\theta}$ is that value of $\boldsymbol{\kappa}$ where $\boldsymbol{\Theta(\mathcal{C})}$ attains the maxima.

3.4.3 Supergraph Construction

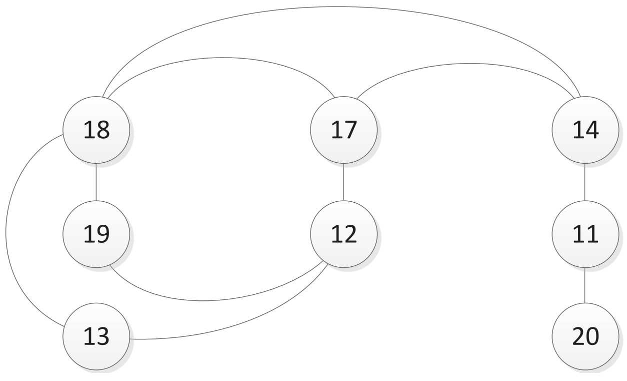
Supergraph construction starts with creating supernodes, then establishing weighted superlinks between them.

3.4.3.1 Supernode creation

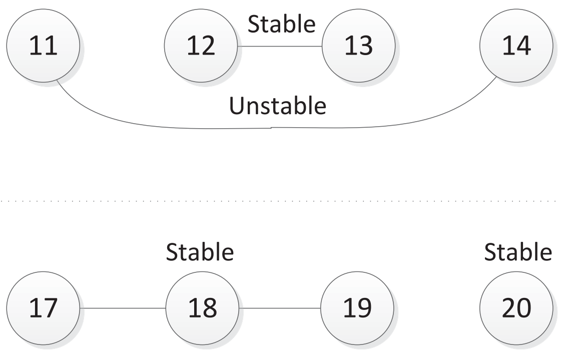
Once the MCG measure for all values of κ are computed for the sample data, all those κ for which the value lies above a predefined threshold value ϵ_θ, are considered for supernode creation. The k-means algorithm is now applied on the complete dataset with the shortlisted values of κ. Let $\varrho = \{\varrho^\kappa : \Theta(\mathcal{C}^\kappa) \geq \epsilon_\theta\}$ be the set of clustering configuration indicator vectors, where each ϱ^κ is of length n_r, and the value of $\varrho^\kappa(i)$ indicates the cluster to which the node v_i belongs. These vectors along with the adjacency matrix $A_\mathcal{G}$ give the connectivity information of nodes. Nodes v_i and v_j are considered as directly connected if they are grouped in the same cluster by k-means and are adjacent as well in the actual road network. Using this information the total number of connected components is computed for each ϱ^κ and the clustering configuration having the minimum number of connected components is finally selected as the optimal ϱ^θ. These components form the supernodes. A lesser number of supernodes makes the framework more scalable for large networks. Therefore in order to get fewer but informative supernodes, the method of supernode creation selects that clustering configuration among the short-listed ones as optimal, which leads to the lowest number of connected components. We apply the standard FIFO based connected components identification algorithm. Its computational complexity is $\mathbf{O}(\mathbf{max}(n_r, n_e))$, where n_r and n_e are the total number of nodes (road segments) and edges (adjacency relationships) in the road graph respectively.

All the connected components corresponding to ϱ_θ are then considered as supernodes to form the set $\mathcal{V}_s = \{\varsigma_1, \varsigma_2, \ldots, \varsigma_{n_\varsigma}\}$. Thus each cluster of nodes which is connected as well in $\mathcal{G}$ is accepted as a supernode. Feature value of each supernode is set as the mean of the cluster (given by k-means) to which they belong, i.e., $\forall \varsigma_i \in \mathcal{C}^\theta_j, \varsigma_i.f = \mu(\mathcal{C}^\theta_j)$.

Setting an appropriate optimality threshold value is crucial to the complexity of the overall algorithm. A lower value would lead to a large number of κ for which the MCG measure would be above the threshold, and would require computing and storing a large number of clustering configuration indicator vectors. It may sometimes lead to have fewer supernodes, but the cost of complexity for so many clustering indicator vectors has to be borne. On the other hand, a higher threshold would lead to fewer clustering indicator vectors but may result in producing more supernodes, which increases the complexity of the partitioning algorithm.



(a) Sample graph



(b) Supernodes

FIGURE 3.4: Supernode stability check

3.4.3.2 An extension for supernode stability check

Sometimes the clustering and adjacency patterns of the road graph $\boldsymbol{\mathcal{G}}$ may lead to form supernodes that do not guarantee the compactness and tightness of bonding that we want to impose. Figure 3.4 shows such an example, in which supernodes are formed from a sample graph after applying $\boldsymbol{k}$-means with $\boldsymbol{\kappa = 2}$ (which is optimal as per our MCG). The $\boldsymbol{\{11, 14\}}$ set is loosely bonded. Considering it as a supernode will bind them together to belong to the same final partition. However, as it is connected to both 17 and 18 of the other cluster, it may suit 14 more to be with them in the final partition. Taking this matter into concern, here we present an extended method that determines the stability of created supernodes and splits them to make them reach a desired stability. We define a measure called *stability* that determines how much the nodes inside a supernode deserve to be together by looking into the tightness of bonding. The closer the nodes in a supernode are, the higher will be its stability measure.

Algorithm 2: Supernode stability check(Supernode set $\mathcal{V}_s$, stability threshold ϵ_η)

```
1  $stack \leftarrow$ initialize a stack;
   // push all the supernodes to check their stability
2  forall $\varsigma_i \in \mathcal{V}_s$ do
3      push $\varsigma_i$ into $stack$;
   // split the unstable supernodes until made stable
4  while $stack$ is not empty do
5      $\varsigma_i \leftarrow$ pop from $stack$;
6      if $\eta(\varsigma_i) < \epsilon_\eta$ then
7          $\varsigma_{pre} \leftarrow$ instantiate an empty supernode;
8          $\varsigma_{post} \leftarrow$ instantiate an empty supernode;
9          forall $v_j \in \varsigma_i$ do
10             if $v_j.f \leq \mu(\varsigma_i)$ then
11                 add $v_j$ to $\varsigma_{pre}$ ;
12             else
13                 add $v_j$ to $\varsigma_{post}$ ;
14         $\mathcal{V}_s \leftarrow \mathcal{V}_s \setminus \varsigma_i, \mathcal{V}_s \leftarrow \mathcal{V}_s \cup \varsigma_{pre} \cup \varsigma_{post}$;
15         push $\varsigma_{pre}$ into $stack$, push $\varsigma_{post}$ into $stack$;
16 return $\mathcal{V}_s$;
```

Definition 3.9. (**Supernode Stability**) The *stability* measure $\eta(\varsigma_i) \in [0,1]$ of a supernode is defined in Equation 3.3, where $|\varsigma_i|$ denotes the number of nodes v_j in ς_i and $\mu(\varsigma_i)$ denotes the mean of feature values $v_j.f$ of all nodes in ς_i. The supernode ς_i is said to be *stable* if its stability measure is greater than or equal to a pre-defined stability threshold ϵ_η, else it is said to be *unstable.* ■

$$\eta(\varsigma_i) = \frac{1}{|\varsigma_i|} \times \sum_{v_j \in \varsigma_i} \exp\left(-\text{abs}\left(\frac{v_j.f + 1}{\mu(\varsigma_i) + 1} - 1\right)\right) \tag{3.3}$$

The above formulation looks into the distance of all nodes from the supernode centroid by the ratio of their feature values to their supernode mean values, and then takes their average value. The 1s are added in the numerator and denominator to avoid the zero values. It would yield its value as 1 when all the nodes inside the supernode have their feature values same as the supernode mean, and lower as much as the node feature values go far from the supernode mean up to 0. The LIFO based stability check algorithm is shown in Algorithm 2. All *stable* supernodes are accepted right away retaining their existing feature value $\varsigma_i.f$ (lines 5–6). However the *unstable* supernodes are further processed to split up into two parts from its centroid (lines 7–13). They are created as two independent supernodes and again checked to confirm their stability (lines 14–15). The interleaved steps of stability checking (lines 5–6) and their splitting into two supernodes from the centroid (lines 7–15) goes on indefinitely until all of them are made stable. The supernodes that were unstable earlier and made stable this way, their means become their new feature values, i.e. $\varsigma_i.f = \mu(\varsigma_i)$. Thus the newly formed set of supernodes is $\mathcal{V}_s = \left\{\varsigma_1, \varsigma_2, \ldots, \varsigma_{n'_\varsigma}\right\}$, where $n'_\varsigma \in [n_\varsigma, n_r]$. The exact gaps between n_ς and n'_ς, and n'_ς and n_r, depend on the imposed stability threshold ϵ_η. It has two extremes. When it is set to 1, the set of all nodes in the road graph $\mathcal{G}$ that have the same feature value $v_i.f$ as well as are linked directly by an edge, is accepted as a complete supernode. In extreme case n'_ς could be equal to n_r if no two nodes have the same feature value. On the other hand when it is set to 0, all connected components obtained using the optimal clustering configuration indicator vector ϱ_θ are considered as supernodes as if without any stability check where $n'_\varsigma = n_\varsigma$. Setting the stability threshold is based on the trade-off between quality and complexity. The worst case complexity of this task is $\mathbf{O}(2n_r - n_\varsigma)$ when all the supernodes are split up repeatedly until only single nodes are left in each supernode, whereas the best case complexity is $\mathbf{O}(n_\varsigma)$ when no supernode needs to be split.

3.4.3.3 Superlink establishment

For the obtained set of supernodes $\mathcal{V}_s$ and the available road graph $\mathcal{G}$, let $\mathcal{L}_{pq}$ be the set of links $\{e_j = link(v_x, v_y)\}$ existing between all $v_x \in \varsigma_p$ and all $v_y \in \varsigma_q$. A superlink ε_i is established between each pair of supernodes $(\varsigma_p, \varsigma_q)$, for which the condition $\mathcal{L}_{pq} \neq \phi$ is fulfilled (Algorithm 1, lines 21–25). The set of superlinks is denoted by $\mathcal{E}_s = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{n_\varepsilon}\}$. At the same time, each superlink ε_i is weighted by a value $\omega_i \in [0, 1]$ in Equation 3.4, where $|\mathcal{L}_{pq}|$ denotes the number of links in $\mathcal{L}_{pq}$, $\varsigma_p.f$ and $\varsigma_q.f$ are the feature values of ς_p and ς_q respectively, and $\sigma^2(\varsigma) = \frac{1}{n_\varsigma} \times \sum_{i=1}^{n_\varsigma} \left(\varsigma_i.f - \mu^0\right)^2$ is the variance of supernode features with respect to the global mean μ^0.

$$\boldsymbol{\omega_i} = \sqrt{\frac{\mathbf{1}}{|\boldsymbol{\mathcal{L}_{pq}}|} \times \sum_{e_j \in \mathcal{L}_{pq}} \left(\mathbf{exp} \left(\frac{-\left(\boldsymbol{\varsigma_p.f} - \boldsymbol{\varsigma_q.f}\right)^2}{\mathbf{2} \times \boldsymbol{\sigma^2(\varsigma)}} \right) \right)^{\mathbf{2}}} \tag{3.4}$$

The above formulation is in the form of a Gaussian function that assigns a similarity measure between the two supernodes of each linked pair, and takes their average value. The effect of individual links on the weight is high if the supernodes connected by the link have closer feature values, and low otherwise. The normalization by $|\boldsymbol{\mathcal{L}_{pq}}|$ normalizes the bias towards the supernode pairs having large number of links but highly dissimilar feature values. Thus the overall weight considers both the number of individual links between the participating supernodes and their feature values, where larger number of links and closer supernode feature values together lead to higher superlink weight. The set of computed weights associated with each superlink is denoted by $\boldsymbol{\mathcal{W}_s} = \{\boldsymbol{\omega_1}, \boldsymbol{\omega_2}, \ldots, \boldsymbol{\omega_{n_\varepsilon}}\}$. Hence the supergraph mining step becomes complete producing $\boldsymbol{\mathcal{G}_s} = (\boldsymbol{\mathcal{V}_s}, \boldsymbol{\mathcal{E}_s}, \boldsymbol{\mathcal{W}_s})$ (Algorithm 1, lines 26–27).

3.5 Road Supergraph Partitioning

Although a preliminary level of grouping of road segments has already happened in the form of supernodes, the number of supernodes still can be very large. Moreover, the linkages that represent the spatial associations have remained under-utilized, as they have just directly been employed in supernode creation until this stage. This step aims to group the set of supernodes into different partitions in a top-down manner by using the superlinks by which they are connected. These different supernode partitions are obtained as connected within, and this in turn achieves the ultimate objective of getting partitions as node partitions where spatial adjacencies are maintained.

3.5.1 Spectral Clustering for Partitioning

Spectral clustering treats clustering as a graph partitioning problem. In our case, we already have a graph that we want to partition. Among the existing graph cuts, normalized cut has been found to be comparatively effective for graph partitioning, due to the reason that it optimizes both the intra-partition homogeneity and inter-partition heterogeneity at the same point [16, 61]. Its optimization function $\min_{\mathcal{P}} \sum_{i=0}^{k} \frac{W(\mathcal{P}_i,\overline{\mathcal{P}_i})}{W(\mathcal{P}_i,\mathcal{P})}$ is a normalized summation of the cross-partition weighted links, where the normalization is done by all the weighted links having at least one end in the corresponding partition. In this function, both the numerator and denominator take into account just the weighted links, and no consideration is made for the node groupings (or node counts) in the resulting partitions. The links in our road graph are established only if they are adjacent in the road network, and thus the superlinks too are based on adjacency relationships. To partition the graph based on both weighted links and node counts in resulting partitions, in the next section we propose a novel k-way graph cut. Instead of repeated bipartitioning of the whole graph, it produces $k'(> k)$ partitions in just a single iteration, and then applies recursive bipartitioning to produce k partitions, which significantly improves its efficiency.

3.5.2 The k-way α-Cut

For a given weighted graph, which in our case is the supergraph[2] $\mathcal{G}_s$, let us suppose its supernode set is partitioned into k disjoint subsets or clusters as $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$. The *adjacency matrix* of $\mathcal{G}_s$ is denoted by A, the *degree matrix* is denoted by D, which is a diagonal matrix having row sums of A at the diagonal shown in Equation 3.5, and the *Laplacian matrix* $(D - A)$ is denoted by L.

$$D = \begin{pmatrix} \sum_{i=1}^{n_\varsigma} a_{1i} & 0 & \cdots & 0 \\ 0 & \sum_{i=1}^{n_\varsigma} a_{2i} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{i=1}^{n_\varsigma} a_{n_\varsigma i} \end{pmatrix} \tag{3.5}$$

[2]In this subsection the supergraph can be treated just like a weighted graph, and the terms supergraph, supernode, and superlink, could be read synonymously as graph, node, and link, respectively for the application of α-Cut in graph partitioning.

A function $\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_j)}$ is defined in Equation 3.6 as the sum of weights associated with all the superlinks having its supernode at one end in $\boldsymbol{\mathcal{P}_i}$ and the supernode at the other end in $\boldsymbol{\mathcal{P}_j}$.

$$\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_j)} = \sum_{\boldsymbol{\varepsilon_r \in \{SLinks(\mathcal{P}_i, \mathcal{P}_j)\}}} \boldsymbol{\omega_r} = \sum_{\boldsymbol{\varsigma_p \in \mathcal{P}_i, \varsigma_q \in \mathcal{P}_j}} \boldsymbol{A(p,q)} \tag{3.6}$$

Definition 3.10. (**Cut**) For a given partition set $\boldsymbol{\mathcal{P}} = \boldsymbol{\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}}$ the *cut* of a partition $\boldsymbol{\mathcal{P}_i}$ is defined as the summation of weights associated with all the superlinks having their supernodes at one end in $\boldsymbol{\mathcal{P}_i}$ and supernodes at other end in any partition other than $\boldsymbol{\mathcal{P}_i}$, i.e., $\boldsymbol{W(\mathcal{P}_i, \overline{\mathcal{P}_i})}$. ■

Definition 3.11. (**Association**) For a given partition set $\boldsymbol{\mathcal{P}} = \boldsymbol{\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}}$ the *association* of a partition $\boldsymbol{\mathcal{P}_i}$ is defined as the summation of weights associated with all the superlinks having supernodes at both ends in $\boldsymbol{\mathcal{P}_i}$, i.e. $\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_i)}$. ■

The *cut* value of a partition $\boldsymbol{\mathcal{P}_i}$ gives a measure of connectivity strength between $\boldsymbol{\mathcal{P}_i}$ and the rest of the partitions, and thus quantifies the loss incurred in cutting those superlink connections while partitioning the graph. When this value is divided by the number of supernodes in $\boldsymbol{\mathcal{P}_i}$, it gives the average contribution of each supernode in the overall cut of $\boldsymbol{\mathcal{P}_i}$. It represents the inter-partition similarity. Similarly, the *association* value of a partition $\boldsymbol{\mathcal{P}_i}$ gives a measure of connectivity strength within $\boldsymbol{\mathcal{P}_i}$ that binds it as a unit, and thus quantifies the retained association of $\boldsymbol{\mathcal{P}_i}$ after partitioning the graph. When this value is divided by the number of supernodes in $\boldsymbol{\mathcal{P}_i}$, it gives the average contribution of each supernode in the overall association of $\boldsymbol{\mathcal{P}_i}$. It represents the intra-partition similarity. A good partitioning is achieved by minimizing the summation of average cut values and simultaneously maximizing the summation of average association values of each partition [16]. However, optimizing any one of these objectives does not guarantee the other. One possible approach is that of normalized cut [16]. It minimizes inter-partition similarity and maximizes intra-partition similarity simultaneously. But that optimization is based on normalized values of cut and association, where the normalization considers the link connectivities between nodes, instead of the nodes directly, and it does not guarantee the optimization of their average cut and association.

To achieve a well balanced optimization of both average cut and average association, in this chapter we design a novel $\boldsymbol{k}$-way graph cut called $\boldsymbol{\alpha}$-Cut. It aims to achieve a partitioning configuration optimized by the objective function $\mathbf{min}_{\boldsymbol{\mathcal{P}}}\ \boldsymbol{\alpha\text{-}Cut(\mathcal{P})}$, where $\boldsymbol{\alpha\text{-}Cut(\mathcal{P})}$ is shown in Equation 3.7.

$$\boldsymbol{\alpha\text{-}Cut(\mathcal{P}) = \sum_{i=1}^{k} \left(\alpha \times \frac{W(\mathcal{P}_i, \overline{\mathcal{P}_i})}{|\mathcal{P}_i|} - (1-\alpha) \times \frac{W(\mathcal{P}_i, \mathcal{P}_i)}{|\mathcal{P}_i|} \right)} \tag{3.7}$$

It minimizes a combination of two components, which separately are the minimization of average *cut* representing the inter-partition similarity, and the maximization of average *association* representing the intra-partition similarity. The $\boldsymbol{\alpha \in [0,1]}$ acts as a balance between the two components. Its value is crucial to obtain the best possible optimized partitions. An advantage of $\boldsymbol{\alpha}$-Cut over normalized cut is that $\boldsymbol{\alpha}$-Cut normalizes the cut and association by the partition size, whereas normalized cut normalizes the cut by the association.

3.5.3 Determining $\boldsymbol{\alpha}$ in $\boldsymbol{\alpha}$-Cut

Instead of considering $\boldsymbol{\alpha}$ as a single constant value for all the partitions, we consider it as a vector $\boldsymbol{\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_k \rangle}$, where each $\boldsymbol{\alpha_i}$ corresponds to the partition $\boldsymbol{\mathcal{P}_i}$. The advantage in considering it as a vector over a single scalar value is its non-uniformly defined value depending on the nature of the respective partition. We consider this factor $\boldsymbol{\alpha_i}$ as the portion of connectivity weight contributed by $\boldsymbol{\mathcal{P}_i}$ in the whole supergraph (including intra-connections as well as inter-connections), and define it as the ratio of the summation of its superlink connection weights to the summation of all superlink connection weights in the supergraph, i.e., $\boldsymbol{\alpha_i = \frac{W(\mathcal{P}_i, \mathcal{V}_s)}{W(\mathcal{V}_s, \mathcal{V}_s)}}$. Its value ranges from 0 to 1. On the other hand, $\boldsymbol{(1-\alpha_i)}$ gives the portion of the connectivity weight contributed by all partitions other than $\boldsymbol{\mathcal{P}_i}$. Putting this value of $\boldsymbol{\alpha_i}$ in Equation 3.7, $\boldsymbol{\alpha}$-Cut simplifies as shown in Equation 3.8.

$$\begin{aligned}
\alpha\text{-}Cut(\mathcal{P}) =& \sum_{i=1}^{k}\left(\frac{W(\mathcal{P}_i,\mathcal{V}_s)}{W(\mathcal{V}_s,\mathcal{V}_s)} \times \frac{W(\mathcal{P}_i,\overline{\mathcal{P}_i})}{|\mathcal{P}_i|} - \frac{W(\mathcal{P}_i,\mathcal{P}_i)}{|\mathcal{P}_i|}\right. \\
&\left.+\frac{W(\mathcal{P}_i,\mathcal{V}_s)}{W(\mathcal{V}_s,\mathcal{V}_s)} \times \frac{W(\mathcal{P}_i,\mathcal{P}_i)}{|\mathcal{P}_i|}\right) \\
=& \sum_{i=1}^{k}\left(\frac{W(\mathcal{P}_i,\mathcal{V}_s)}{W(\mathcal{V}_s,\mathcal{V}_s)} \times \left(\frac{W(\mathcal{P}_i,\overline{\mathcal{P}_i})}{|\mathcal{P}_i|} + \frac{W(\mathcal{P}_i,\mathcal{P}_i)}{|\mathcal{P}_i|}\right)\right. \\
&\left.-\frac{W(\mathcal{P}_i,\mathcal{P}_i)}{|\mathcal{P}_i|}\right) \\
=& \sum_{i=1}^{k}\left(\frac{W(\mathcal{P}_i,\mathcal{V}_s)}{W(\mathcal{V}_s,\mathcal{V}_s)} \times \frac{W(\mathcal{P}_i,\mathcal{V}_s)}{|\mathcal{P}_i|} - \frac{W(\mathcal{P}_i,\mathcal{P}_i)}{|\mathcal{P}_i|}\right)
\end{aligned} \tag{3.8}$$

Like normalized cut [16], the problem to achieve a partitioning configuration which minimizes this cost is an NP-complete problem. To solve it in a time-bound and computationally efficient manner, we follow a spectral clustering approach described in the following subsection.

3.5.4 Spectral Clustering Approach to α-Cut

If $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k\}$ is the set of k disjoint partitions of $\mathcal{G}_s$, let $\mathbf{1} \in \mathbb{R}^{n_\varsigma}$ be a vector with each of its values as 1, and $c_i \in \mathbb{R}^{n_\varsigma}$ be the cluster indicator vector of $\mathcal{P}_i$ such that its jth value $c_i(j) = 1$, if $\varsigma_j \in \mathcal{P}_i$, and $c_i(j) = 0$ otherwise, as shown in Equation 3.9..

$$c_i(j) = \begin{cases} 1, & \text{if } (\varsigma_j \in \mathcal{P}_i) \\ 0, & \text{if } (\varsigma_j \notin \mathcal{P}_i) \end{cases} \tag{3.9}$$

The spectral clustering approach to minimize the cost of α-Cut partitioning follows a relaxed approach based on eigenvectors and eigenvalues. The relaxation lies in the cluster indicator vectors, which are allowed to take on any real value, instead of restricting them only to discrete values. Using the cluster indicator vectors, the α-Cut formulation can be simplified by substituting $W(\mathcal{P}_i, \mathcal{V}_s)$ by $\mathbf{1}^T D c_i$, $W(\mathcal{P}_i, \mathcal{P}_i)$ by $c_i^T A c_i$, $W(\mathcal{V}_s, \mathcal{V}_s)$ by

Algorithm 3: $\boldsymbol{\alpha}$-Cut Partitioning(Supergraph $\boldsymbol{\mathcal{G}_s}$, number of desired partitions $\boldsymbol{k}$)

```
1  $A \leftarrow$ adjacency matrix of $\mathcal{G}_s$;
2  $D \leftarrow$ degree matrix of $\mathcal{G}_s$;
3  $M \leftarrow \left( \frac{(1^T D)^T (1^T D)}{1^T D 1} - A \right)$;          // get the $\alpha$-Cut matrix
4  $\bigcup_{i=1}^{n_\varsigma} \{(y_i, \lambda_i)\} \leftarrow$ get eigenvector and eigenvalue pairs of $M$;
5  sort eigenvalues $\lambda_i$ to have $\lambda_{n_\varsigma} \leq \lambda_{n_\varsigma - 1} \leq \cdots \leq \lambda_1$;
6  select $\{\lambda_{n_\varsigma}, \lambda_{n_\varsigma-1}, \ldots, \lambda_{n_\varsigma-k+1}\}$ eigenvalues and corresponding eigenvectors
   $\{y_{n_\varsigma}, y_{n_\varsigma-1}, \ldots, y_{n_\varsigma-k+1}\}$;
7  generate matrix $Y_{n_\varsigma \times k} = \begin{pmatrix} y_1 & y_2 & \ldots & y_k \end{pmatrix}$;
8  $Z \leftarrow$ row normalize $Y$;
9  $\{z_1, z_2, \ldots, z_{n_\varsigma}\} \leftarrow$ get row vectors of $Z$;
10 $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\} \leftarrow$ k-means $(\{z_1, z_2, \ldots, z_{n_\varsigma}\}, k)$;
11 $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{k'}\} \leftarrow$ get disjoint partitions from $\mathcal{C}$;
   // global recursive bipartitioning to obtain $k$ partitions
12 $A'_{k' \times k'} \leftarrow$ compute partition connectivity matrix of $\mathcal{P}$;
13 $queue \leftarrow$ initialize a queue;
14 partition set $\mathcal{P} \leftarrow$ initialize with a single partition of $A'$;
15 enqueue $A'$ into $queue$;
16 repeat
17     $A' \leftarrow$ dequeue from $queue$;
18     $(\mathcal{P}_1, \mathcal{P}_2) \leftarrow$ bipartition $A'$ using $\alpha$-Cut;
19     $A'_1 \leftarrow$ create adjacency matrix for $\mathcal{P}_1$;
20     $A'_2 \leftarrow$ create adjacency matrix for $\mathcal{P}_2$;
21     enqueue $A'_1$ into $queue$, enqueue $A'_2$ into $queue$;
22     $\mathcal{P} \leftarrow \mathcal{P} \setminus$ partition of $A'$;
23     $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_1 \cup \mathcal{P}_2$;
24 until number of partitions in $\mathcal{P}$ equals to $k$;
25 return $\mathcal{P}$;
```

$\boldsymbol{1^T D 1}$, and $|\boldsymbol{\mathcal{P}_i}|$ by $\boldsymbol{c_i^T c_i}$ in Equation 3.8. The simplification steps are shown in Equation 3.10.

$$\begin{aligned}
\alpha\text{-}Cut(\mathcal{P}) &= \sum_{i=1}^{k} \left(\frac{1^T D c_i}{1^T D 1} \times \frac{1^T D c_i}{c_i^T c_i} - \frac{c_i^T A c_i}{c_i^T c_i} \right) \\
&= \sum_{i=1}^{k} \frac{1}{c_i^T c_i} \times \left(\frac{\left(1^T D c_i\right)^2}{1^T D 1} - c_i^T A c_i \right) \\
&= \sum_{i=1}^{k} \frac{1}{c_i^T c_i} \times \left(\frac{c_i^T \left(1^T D\right)^T \left(1^T D\right) c_i}{1^T D 1} - c_i^T A c_i \right) \\
&= \sum_{i=1}^{k} \frac{1}{c_i^T c_i} \times c_i^T \left(\frac{\left(1^T D\right)^T \left(1^T D\right)}{1^T D 1} - A \right) c_i \\
&= \sum_{i=1}^{k} \frac{c_i^T M c_i}{c_i^T c_i} \\
&\text{where } M = \left(\frac{\left(1^T D\right)^T \left(1^T D\right)}{1^T D 1} - A \right)
\end{aligned} \tag{3.10}$$

The derived matrix M is called the α-Cut matrix for $\alpha_i = \dfrac{W(\mathcal{P}_i, \mathcal{V}_s)}{W(\mathcal{V}_s, \mathcal{V}_s)}$, and the spectral clustering algorithm works on this matrix. Equation 3.10 is further simplified as follows.

$$\sum_{i=1}^{k} \frac{c_i^T M c_i}{\|c_i\|^2} = \sum_{i=1}^{k} \left(\frac{c_i}{|c_i|} \right)^T M \left(\frac{c_i}{|c_i|} \right) = \sum_{i=1}^{k} y_i^T M y_i$$

where y_i is a unit vector in the direction of c_i, such that $y_i^T y_i = 1$. Hence the optimization function becomes

$$\min_{\mathcal{P}} \sum_{i=1}^{k} y_i^T M y_i \text{ subject to } y_i^T y_i = 1 \tag{3.11}$$

This is solved by setting its derivative with respect to y_i to zero and introducing a Lagrange multiplier λ_i for each $\mathcal{P}_i$ to incorporate the associated constraint [27], as shown in Equation 3.12.

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{y_i}}\left(\sum_{i=1}^{k}\boldsymbol{y_i^T M y_i}+\sum_{i=1}^{n}\boldsymbol{\lambda_i}\left(\boldsymbol{1-y_i^T y_i}\right)\right)&=\boldsymbol{0}\\ \boldsymbol{My_i-\lambda_i y_i}&=\boldsymbol{0}\\ \boldsymbol{My_i}&=\boldsymbol{\lambda_i y_i}\end{aligned} \tag{3.12}$$

It implies that $\boldsymbol{y_i}$ is one of the eigenvectors of $\boldsymbol{M}$ corresponding to the eigenvalue $\boldsymbol{\lambda_i}$, and $\boldsymbol{y_i^T M y_i = y_i^T \lambda_i y_i = \lambda_i}$. As the objective is minimization, we select $\boldsymbol{k}$ smallest eigenvalues from the total of $\boldsymbol{n_\varsigma}$ eigenvalues as $\boldsymbol{\lambda_{n_\varsigma} \le \lambda_{n_\varsigma-1} \le \cdots \le \lambda_{n_\varsigma-k+1}}$ and corresponding eigenvectors $\boldsymbol{y_{n_\varsigma}, y_{n_\varsigma-1}, \ldots, y_{n_\varsigma-k+1}}$ which represent the relaxed cluster indicator vectors. Thus, it leads to Equation 3.13.

$$\begin{aligned}\min_{\boldsymbol{\mathcal{P}}} \boldsymbol{\alpha\text{-}Cut(\mathcal{P})} &= \boldsymbol{y_{n_\varsigma}^T M y_{n_\varsigma} + \cdots + y_{n_\varsigma-k+1}^T M y_{n_\varsigma-k+1}}\\ &= \boldsymbol{\lambda_{n_\varsigma} + \cdots + \lambda_{n_\varsigma-k+1}}\end{aligned} \tag{3.13}$$

Algorithm 3 presents the complete partitioning method, where the $\boldsymbol{\alpha}$-Cut matrix is computed in line 3 and eigen-decomposed in line 4. Lines 5–6 select the $\boldsymbol{k}$ smallest eigenvalues and corresponding eigenvectors. Ideally the indicator vectors should have only binary values, but the actually obtained indicator vectors are in fact the relaxed vectors and do not follow the binary pattern. Due to the lack of concrete information about clusters, it becomes another problem to separate the $\boldsymbol{k}$ clusters. Here we assume that the clusters are well-separated in the $\boldsymbol{k}$-dimensional eigenspace, which is a general assumption in spectral clustering [27], and use the eigenvectors (or indicator vectors) to generate a matrix $\boldsymbol{Y}$ of $\boldsymbol{n_\varsigma \times k}$ dimensions (line 7). It is then row-normalized using Equation 3.14 to have row-vectors $\boldsymbol{z_i}$ of unit length giving the final matrix $\boldsymbol{Z}$ (line 8). Each row-vector $\boldsymbol{z_i}$ represents a supernode $\boldsymbol{\varsigma_i}$. The set of row-vectors are used to cluster the supernodes by applying $\boldsymbol{k}$-means to find a set of $\boldsymbol{k}$ clusters $\boldsymbol{\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}}$ (lines 9–10), where each cluster $\boldsymbol{\mathcal{C}_i}$ comprises one or more row vectors (supernodes) in $\boldsymbol{Z}$. The supernodes inside each cluster are linked together as they exist in the supergraph. Upon linking them, sometimes even more than one connected components may be found inside a single cluster. These connected components are extracted from each cluster to form disjoint partitions (line 11).

$$Y = \begin{pmatrix} y_{11} & y_{21} & \cdots & y_{k1} \\ y_{12} & y_{22} & \cdots & y_{k2} \\ | & | & & | \\ y_{1n} & y_{2n} & \cdots & y_{mn} \end{pmatrix} = \begin{pmatrix} — & z_1^T & — \\ — & z_2^T & — \\ & \vdots & \\ — & z_n^T & — \end{pmatrix} = Z \tag{3.14}$$

$$\text{where, } z_i = \frac{1}{\sqrt{\sum_{j=1}^{k} y_{ji}^2}} (y_{1i}, y_{2i}, \ldots, y_{mi})^T$$

Depending on data, the number of disjoint partitions may sometimes be large, which would yield the partition set from $\mathcal{C}$ as $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{k'}\}$, where $k' \geq k$. These k' partitions may be accepted as the final result. However, if the requirement to have exactly k partitions is strict, there are two approaches as described in [16]– $i)$ greedy pruning, and $ii)$ global recursive bipartitioning. The greedy pruning approach iteratively merges the two nearest partitions optimizing the defined graph cut, until it results in a total of k partitions. In contrast, the global recursive bipartitioning approach generates a condensed graph where each partition forms a node and adjacent partitions are connected by weighted links with $W(\mathcal{P}_i, \mathcal{P}_j)$ as the weight, and is recursively bipartitioned until it results in a total of k partitions. For large k' values, the greedy pruning approach is computationally intensive. Therefore we follow the global recursive bipartitioning approach.

It begins with computing a partition connectivity matrix A' of dimension $k' \times k'$ (line 12). Its values are computed as connectivity strengths between the partitions $A'(i,j) = \sqrt{\frac{1}{numadj(\mathcal{P}_i,\mathcal{P}_j)} \times \sum_{\varsigma_p \in \mathcal{P}_i, \varsigma_q \in \mathcal{P}_j} (A(p,q))^2}$, where $numadj(\mathcal{P}_i, \mathcal{P}_j)$ gives the number of supernode adjacency relationships between the supernodes of $\mathcal{P}_i$ and $\mathcal{P}_j$. This value automatically becomes zero for the pair of partitions that do not share any adjacency relationship. We use a queue to recursively apply the bipartitioning (lines 13–24). The α-Cut algorithm is applied on A' to have two partitions (line 18). Nodes of A' (old partitions) belonging to each partition (new partition) are separated, and two new matrices are created by separating the corresponding rows and columns of $\boldsymbol{A'}$, such that the sum of dimensions of the two new matrices equals to the dimension of $\boldsymbol{A'}$ (lines 19–20). Each matrix now represents one partition. The bipartitioning is again applied on each matrix individually to yield more than two partitions. These interleaved steps of bipartitioning and matrix creation are repeated until the total number of final partitions equals $\boldsymbol{k}$ (lines 16–24).

The computational complexity of eigen-decomposition is $\mathbf{O}(\boldsymbol{n^3})$ in general and $\mathbf{O}(\boldsymbol{n^2})$ for sparse matrices. The application of $\boldsymbol{k}$-means on row-vectors to find the clusters costs $\mathbf{O}(\boldsymbol{tnk^2})$. In these costs, $\boldsymbol{n = n_\varsigma}$ when the spectral clustering is applied on the super-graph, and $\boldsymbol{n = n_r}$ when it is applied directly on the road graph.

3.6 Experimental Evaluation

In this section, we evaluate the proposed framework in terms of different evaluation metrics. Although there exist many works on general graph partitioning, we compare our results to a recent work [61], which is on the same problem, for a close and specific comparison.

3.6.1 Datasets

We perform experiments on two kinds of datasets, small ($\mathtt{D_1}$) and large ($\mathtt{M_1}$, $\mathtt{M_2}$, and $\mathtt{M_3}$) road networks. Table 3.1 shows the statistics of all these datasets. The traffic on the small network, shared by the authors of [61], is based on a micro-simulation performed for 4 hours at 120 time intervals of 2 minutes. At each time point $\boldsymbol{t}$, the traffic density on each road segment is computed in terms of vehicle/metre. In this work we perform experiments at $\boldsymbol{t = 71}$ to compare our results with [61] which used the same dataset.

The traffic data for the large networks is generated by a web-based[3] random road traffic generator MNTG [133]. We populate $\mathtt{M_1}$, $\mathtt{M_2}$, and $\mathtt{M_3}$ by 25,246, 62,300, and 84,999 vehicles respectively, and obtain their trajectories for 100 continuous timestamps. A self-designed

[3]It can be accessed through `http://mntg.cs.umn.edu/tg/`

Table 3.1: Dataset statistics

	$\mathtt{D_1}$	$\mathtt{M_1}$	$\mathtt{M_2}$	$\mathtt{M_3}$
Place	Downtown San Francisco	CBD Melbourne	CBD(+) Melbourne	Melbourne
Area (sq. ml.)	2.5	6.6	31.5	42.03
Road seg	420	17,206	53,494	79,487
Intersection pt	237	10,096	28,465	42,321

program is used to map their positions to corresponding road segments, and compute the traffic density of road segments (in terms of vehicles/metre) at each point of time.

3.6.2 Evaluation Metrics

The partitioning framework is evaluated using metrics that quantify the quality of results from different perspectives. The problem defined in Section 3.2.2 intends to achieve four different conditions. As we obtain results in the form of disjoint and connected road network partitions, **C.1** and **C.2** are automatically fulfilled. **C.3** which enforces inter-partition heterogeneity is evaluated by the *inter* metric. It is the average of inter-partition distances between each pair of spatially adjacent partitions, where the inter-partition distance is the average absolute distance between nodes from the respective pair of adjacent partitions. **C.4** which enforces intra-partition homogeneity is evaluated by the *intra* metric[4]. For each partition, it computes the intra-partition distance as the average absolute distance between the pair of nodes, and then takes the average of that computed for all the partitions.

Additionally, we also evaluate the overall partitioning. The standard metrics of cluster evaluation do not take the associated spatial adjacencies into account. For its proper evaluation, we use two metrics derived from the standard cluster evaluation metrics to make them suitable for the graph partitioning problem. They are the graph Davies-Bouldin index (GDBI). based on Davis-Bouldin index (DBI), and the average NcutSilhouette (ANS)

[4] $\boldsymbol{Intra(\mathcal{P})} = \frac{1}{|\mathcal{P}|} \times \sum_{\mathcal{P}_i \in \mathcal{P}} \frac{\sum_{\substack{v_p, v_q \in \mathcal{P}_i \\ p \neq q}} \mathbf{abs}(v_p.f - v_q.f)}{|\mathcal{P}_i| \cdot (|\mathcal{P}_i| - 1)}$

measure defined in [61] especially for partition evaluation. In both these measures, smaller values indicate better partitioning.

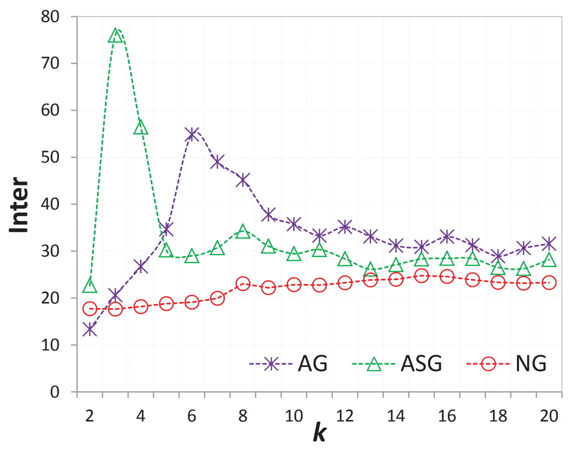
3.6.3 Experimental Results on Small Networks

We perform experiments on the small road network $\mathtt{D_1}$ to compare the partitioning quality of our $\boldsymbol{\alpha}$-Cut based partitioning framework with other state-of-the-art techniques using performance evaluation metrics listed in Section 3.6.2, and demonstrate its effectiveness. For an exhaustive analysis from different perspectives we present the results obtained on several different schemes. Here we introduce the notations used for those schemes. AG and NG are the schemes when $\boldsymbol{\alpha}$-Cut and normalized cut are applied directly on the road graph respectively, and ASG and NSG are the schemes when $\boldsymbol{\alpha}$-Cut and normalized cut are applied on the road supergraph with no stability check respectively.

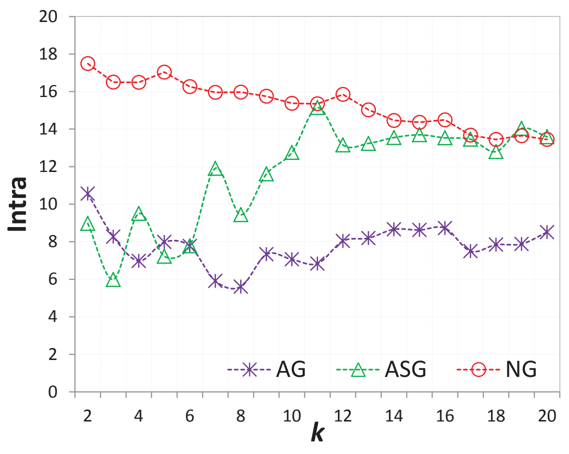
Results in this section are the median values of evaluation metrics obtained from 100 execution of the algorithm. The reason is that $\boldsymbol{k}$-means (used to cluster eigenvectors) may sometimes produce slightly different results in different executions because of randomized cluster initialization.

We consider NG as the baseline, and comparatively show our results. Figure 3.5 shows the complete results of AG and ASG in comparison to NG in terms of evaluation metrics for the number of partitions $\boldsymbol{k}$ ranging from 2 to 20. GDBI and ANS measures quantify the overall partitioning quality. In terms of both these measures, both AG and ASG schemes of our framework outperform NG at all values of $\boldsymbol{k}$. Also in terms of *intra*, that quantifies intra-partition homogeneity, we outperform NG. In terms of *inter*, that quantifies inter-partition heterogeneity, AG outperforms NG at all values except $\boldsymbol{k = 2}$, whereas ASG outperforms at all values.

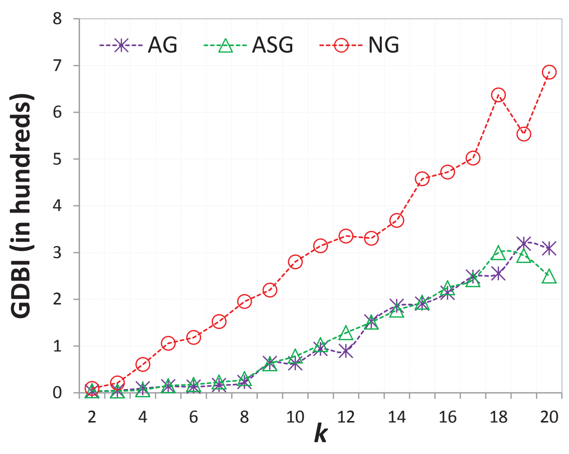
The overall partitioning quality is evaluated by GDBI and ANS, which consider both the inter-partition heterogeneity and intra-partition homogeneity simultaneously. As stated earlier, lower values indicate better partitioning for both these measures. In both the Figures 3.5(c) and 3.5(d), AG is much lower than NG at all values of $\boldsymbol{k}$. The GDBI measure increases with increasing $\boldsymbol{k}$, but this is not the case with ANS. In [61], the authors used



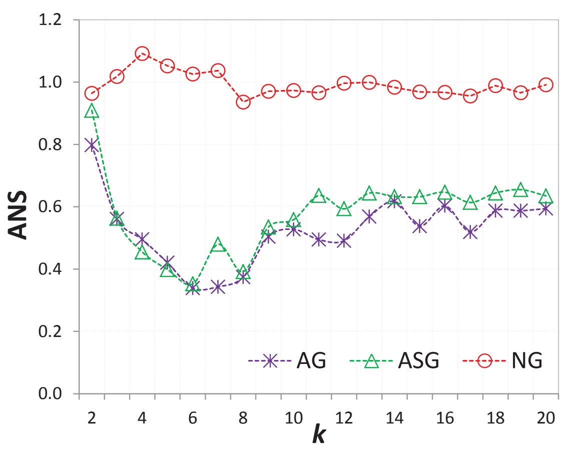
(a) Inter



(b) Intra



(c) GDBI



(d) ANS

FIGURE 3.5: Road graph and supergraph partitioning results in small networks

the ANS measure to learn the number of optimal partitions. They accept the value of $\boldsymbol{k}$ that leads to the ANS minima as the optimal number of partitions, which in this case is 6 for `AG` and 8 for `NG`. The minima of `AG` being much lower than that of `NG`, is found as the better performer. As the schemes `AG` and `NG` applies $\boldsymbol{\alpha}$-Cut and normalized cut respectively directly on the graph, the obtained results shows the superiority of $\boldsymbol{\alpha}$-Cut.

The supergraph technique, as said earlier, is to make the framework applicable for large road networks, in which we may need to compromise the quality to some extent. Here we show the effects of the supergraph on the partitioning quality. Figure 3.7(a) shows the stability measures $\boldsymbol{\eta(\varsigma)}$ of the 105 supernodes. When the stability threshold $\boldsymbol{\epsilon_\eta = 0}$, the partitioning scheme behaves as `ASG`, whereas $\boldsymbol{\epsilon_\eta = 1}$ makes it behave as `AG`. The figure, which also presents the results obtained by `ASG`, shows how the inclusion of the supergraph technique affects the partitioning. The GDBI and ANS plots show that partitioning a network by `ASG` is qualitatively almost the same as that by `AG`. However, `AG` generally lies below `ASG` in the ANS plot indicating its superiority. At few points ($\boldsymbol{k = 4, 5}$) in Figure 3.5(d), ASG is better than AG. The reason is that those values do not suit the dataset for partitioning and therefore sometimes it takes place arbitrarily. Applying the stability check on the supergraph with any value between 0 to 1 as its threshold results in a partitioning that is qualitatively between `AG` and `ASG`.

To summarize the overall results, like Ji and Geroliminis [61], we consider the ANS measure as the deciding factor for the optimal number of partitions. We now compare the best (lowest) ANS measures (which gives the optimal partitioning) of all the schemes along with [61]. Table 3.2 shows that both of our schemes `AG` and `ASG` are much lower (better) than `NG` and [61]. As Ji and Geroliminis perform additional adjustments after partitioning by normalized cut, their partitions are somewhat improved in quality than `NG`, but even then our method outperforms theirs.

Table 3.2: Overall quality of partitioning

Scheme	**ANS**	**k**	**Scheme**	**ANS**	**k**
`AG`	**0.3392**	6	`NG`	0.9362	8
`ASG`	0.3526	6	Ji and Geroliminis [61]	0.6210	3

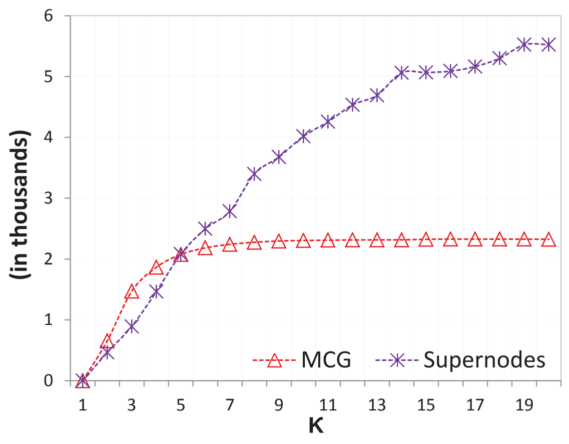
We can also look into the partitioning quality more closely in Figures 3.5(a) and 3.5(b), which show the inter-partition and intra-partition distances separately. As we want to obtain a partition set having the highest possible inter-partition distances, higher values of *inter* indicate better partitioning. Except $\boldsymbol{k = 2}$, at all values of $\boldsymbol{k}$ in the range, `AG` has higher values than `NG`. Thus if the optimal number of partitions for this data comes out to be 2, which is not true (found as 6 in previous paragraphs), `NG` outperforms `AG` in terms of this measure. The value of `AG` increases rapidly until $\boldsymbol{k = 6}$, which is the maxima. After that point it decreases rapidly again, and gradually comes to relatively stable values. The maxima of `AG` for *inter* lies at $\boldsymbol{k = 6}$ which coincides with the minima of ANS. Another perspective to evaluate the partitioning is to look into the intra-partition distances using *intra*. As our objective is to minimize intra-partition distances, lower values are an indicator of better partitioning. In the figure we can see that at all values of $\boldsymbol{k}$ in the range, `AG` has lower values than `NG`.

In the curve of `ASG` of *inter*, there is a sudden rise at $\boldsymbol{k = 2}$, but then it comes down in between `AG` and `NG`. Similarly *intra* fluctuates over the initial values of $\boldsymbol{k}$, after which it comes in between the other two, whereas at higher values its trend becomes similar to `NG`. The reason for the abnormal behavior at the initial values is that they do not suit the dataset for partitioning by the $\boldsymbol{\alpha}$-Cut. When the partitioning is applied at those values, it takes place arbitrarily for some instances, which makes it behave abnormally. As is evident from the ANS plot, the initial values of $\boldsymbol{k}$ are not so good for partitioning.

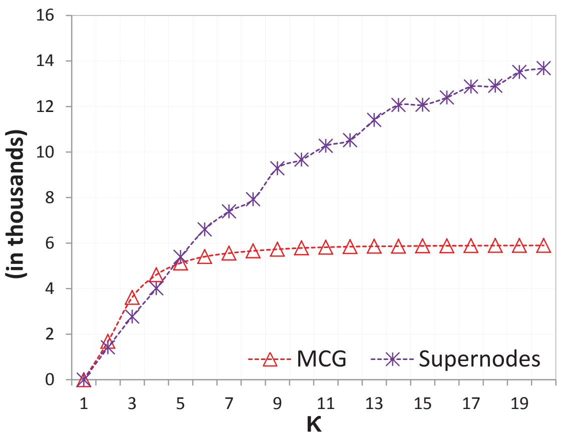
3.6.4 Experimental Results on Large Networks

We perform experiments on large road networks $\mathtt{M_1}$, $\mathtt{M_2}$ and $\mathtt{M_3}$ to validate the scalability of our framework. Additionally, we also show that the quality of partitioning large networks is comparable to that of partitioning small networks.

Figure 3.6 shows the MCG measures and the number of supernodes obtained from the cluster sets produced by $\boldsymbol{k}$-means at different values of $\boldsymbol{\kappa}$ on $\mathtt{M_1}$ and $\mathtt{M_2}$. At the initial values, the MCG measure rises steeply up to some point, beyond which there is little change. In case of $\mathtt{M_1}$, the maxima 2326.88 is attained at $\boldsymbol{\kappa = 18}$, after which it starts declining gradually, but the major rise is only up to $\boldsymbol{\kappa = 5}$. As higher MCG measures indicate better clustering, the best quality cluster set of $\mathtt{M_1}$ is obtained at $\boldsymbol{\kappa = 18}$, but those obtained at lower values, up to $\boldsymbol{\kappa = 5}$ with an MCG measure of 2075.16, do not differ much in quality. If we look into the number of obtained supernodes, it increases monotonically with the increasing value of $\boldsymbol{\kappa}$. As having larger number of supernodes adds on complexity to the remaining partitioning task, it is worth choosing the value of $\boldsymbol{\kappa}$ after which there is little increase in MCG. We get this value by fixing the optimality threshold $\boldsymbol{\epsilon_\theta}$ to 2000 for $\mathtt{M_1}$ and 5000 for $\mathtt{M_2}$. It leads to an optimal $\boldsymbol{\kappa}$ of 5 for both datasets, and the obtained number of supernodes are 2,081 and 5,391 respectively. Thus our supergraph technique reduces the adjacency matrix dimension from 17,206 and 53,494 to 2,081 and 5,391 for $\mathtt{M_1}$ and $\mathtt{M_2}$ respectively. Similarly, the optimal $\boldsymbol{\kappa}$ for $\mathtt{M_3}$ is found as 5, which produces 9179 supernodes. This significantly reduces both the space and time complexity, and if required the complexity can further be reduced by selecting a lower $\boldsymbol{\kappa}$, in which the partitioning quality may degrade to some extent. Figure 3.7(b) shows the supernode stability measures of 5391 supernodes of the $\mathtt{M_2}$ dataset. We can see that most supernodes are highly stable. Therefore we proceed with these supernodes for supergraph construction and its partitioning by $\boldsymbol{\alpha}$-Cut.

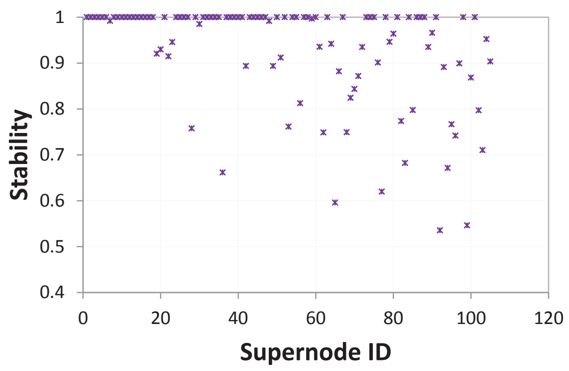


(a) $\mathtt{M_1}$

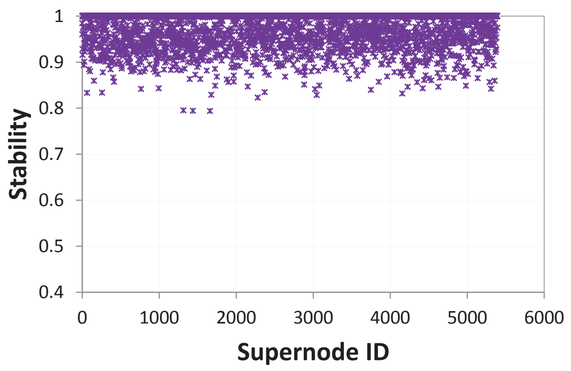


(b) $\mathtt{M_2}$

FIGURE 3.6: MCG measure and number of supernodes in large networks



(a) $\mathtt{D_1}$



(b) $\mathtt{M_2}$

FIGURE 3.7: Stability measure of supernodes

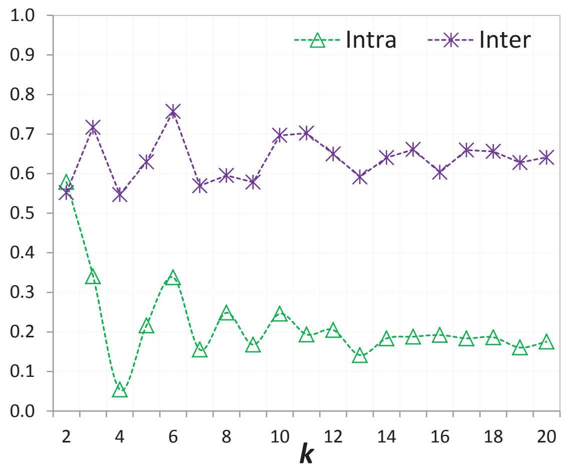
Figure 3.8 shows the final partitioning results obtained for all the three large datasets. The plots show the measures of respective metrics in Y-axis at different values of $\boldsymbol{k}$. As shown in Figures 3.8(b), 3.8(d), and 3.8(f), we get the best (lowest) ANS measures of 0.423 at $\boldsymbol{k = 4}$, 0.511 at $\boldsymbol{k = 5}$, and 0.512 at $\boldsymbol{k = 5}$ on $\mathtt{M_1}$, $\mathtt{M_2}$, and $\mathtt{M_3}$, respectively. These values are not as good as we found on small networks (`AG`- 0.3392 and `ASG`- 0.3526) in Section

3.6.3, but still they are much better than the small network baseline results (NG- 0.9362, [61]- 0.6210). Moreover, results indicate that the partitioning of $\mathtt{M_1}$ is qualitatively better than that of $\mathtt{M_2}$ and $\mathtt{M_3}$, but worse than that of $\mathtt{D_1}$. It shows that as the size of the road network increases, the partitioning quality decreases.

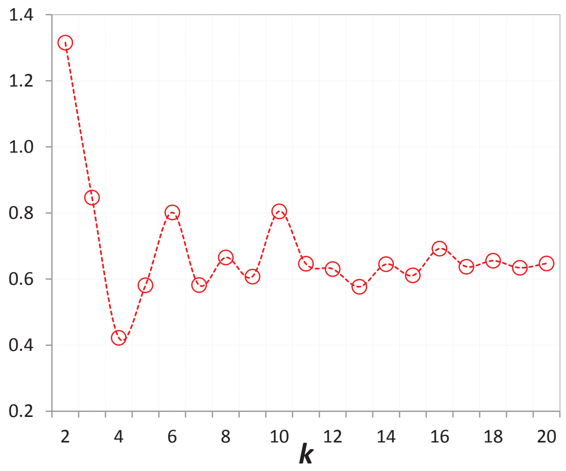
As we get the lowest ANS measure for $\mathtt{M_1}$ at $\boldsymbol{k = 4}$, the best possible way to partition this network is to divide it into 4 segments as produced by our framework, each of which exhibit distinctive traffic congestion inside. However, if the congestion pattern has to be analyzed more closely, we can also have more partitions, and $\boldsymbol{k = 7, 9, 13, \ldots}$ being the local minima serve as good candidates for the number of partitions. Similarly, some other suitable candidates for having a good congestion-based partitioning are $\boldsymbol{k = 7, 9, 12, 14, \ldots}$ for $\mathtt{M_2}$, and $\boldsymbol{k = 9, 11, 14, 17, \ldots}$ for $\mathtt{M_3}$.

At lower values of $\boldsymbol{k}$, a small change makes a big effect in the partitioning quality, as can be seen from the fluctuations in Figures 3.8(b), 3.8(d) and 3.8(f). However, as $\boldsymbol{k}$ becomes large, the fluctuations diminish. The reason behind this is that at smaller values of $\boldsymbol{k}$, say 2, when it is increased to 3, a large re-arrangement takes place inside the partitions. It would be much larger than the re-arrangement that takes place at higher values of $\boldsymbol{k}$, say when it increases from 22 to 23. Unlike the results of the small network, the *intra* and *inter* measures here are very small. The reason is that the road segment densities in $\mathtt{M_1}$ and $\mathtt{M_2}$ are much lower than those in $\mathtt{D_1}$, and those in $\mathtt{M_3}$ is even lower than all.

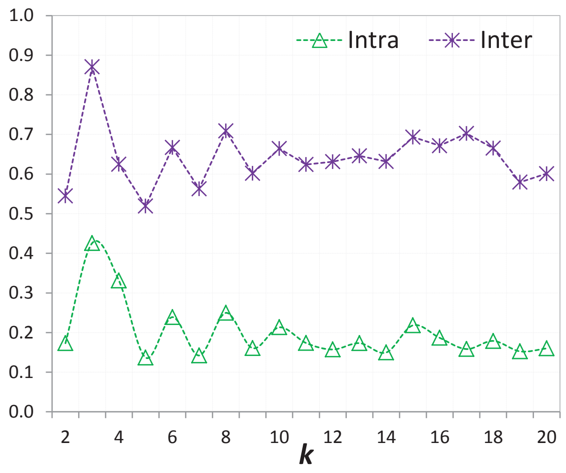
For large road networks, the most time-taking task in the framework is the eigen-decomposition (Algorithm 3, line 4). This becomes a major overhead when dimension of the matrix $\boldsymbol{M}$ becomes large. We overcome this issue (up to some extent) in our study by applying a high performance algorithm developed and used in Matlab [134]. It reduces of the original matrix to a condensed form by orthogonal transformations, decomposes the matrix, and then transforms it back. Table 3.3 shows the running time of our framework in number of seconds consumed in its complete execution. The total time has been broken down to show the individual times consumed int the different modules, where the different modules are those described in Section 3.3. For the small dataset $\mathtt{D_1}$, it takes just fractions of a second to complete, whereas for large datasets $\mathtt{M_1}$, $\mathtt{M_2}$, and $\mathtt{M_3}$ it takes 2.15 minutes, 31.75 minutes, and 1.64 hours, respectively. Module 1 takes the lowest amount of time,



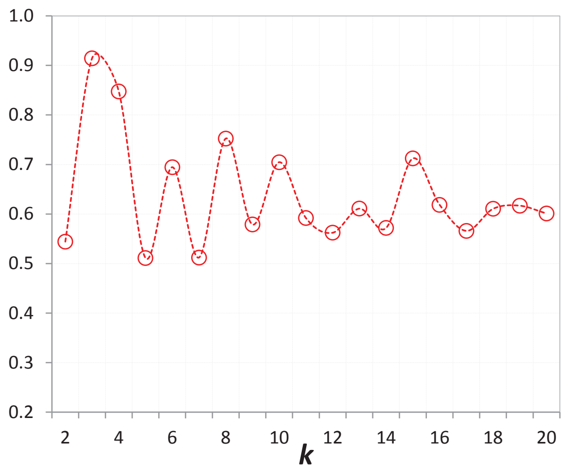
(a) Inter and Intra in $\mathtt{M_1}$



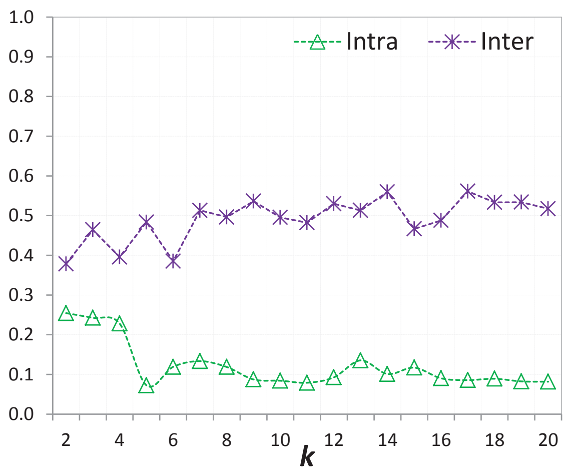
(b) ANS in $\mathtt{M_1}$



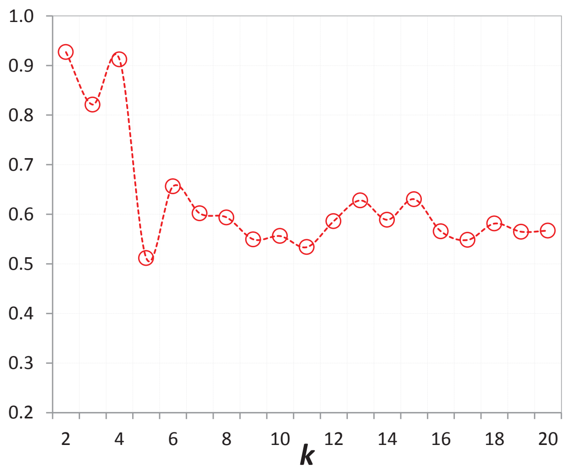
(c) Inter and Intra in $\mathtt{M_2}$



(d) ANS in $\mathtt{M_2}$



(e) Inter and Intra in $\mathtt{M_3}$



(f) ANS in $\mathtt{M_3}$

FIGURE 3.8: Road supergraph partitioning results in large networks

whereas module 3, which includes the eigen-decomposition task, takes the highest amount of time. The total time taken for $\mathtt{M_3}$ is certainly very high, and therefore while applying repeated partitioning on an urban road network, at the beginning it can be started by partitioning the whole network. But after having its relatively small partitions, they can be repeatedly subjected to partitioning distributively with the changing congestion measures with respect to time. In this way it helps in reducing the running time, and can even be applied in real-time if the network becomes as small as $\mathtt{M_1}$.

Table 3.3: Running Time (in seconds)

Module	$\mathtt{D_1}$	$\mathtt{M_1}$	$\mathtt{M_2}$	$\mathtt{M_3}$
1	less than 1	9	24	137
2	less than 1	54	848	2044
3	less than 1	66	1033	3726
Total	less than 1	129	1905	5907

3.7 Summary

In this chapter, we presented a spectral clustering based framework for traffic congestion-based spatial partitioning of large urban road networks. We first formally gave a mathematical representation of actual road networks and transformed the road network into a road graph, and then to a road supergraph by clustering the node feature values. The novel $\boldsymbol{k}$-way $\boldsymbol{\alpha}$-Cut partitioning algorithm is applied on the supergraph to obtain $\boldsymbol{k}$ partitions. The mining of the supergraph leads to a preliminary grouping of road segments in the form of supernodes, which significantly reduces the partitioning load. This technique makes the framework scalable and suitable to handle the rapidly growing urban road networks. The $\boldsymbol{\alpha}$-Cut algorithm, proposed in this chapter, aims to achieve a good balance of average cut and average association through spectral clustering. This algorithm also approximately maximizes the network modularity. In our experiments, we found that it produces partitions qualitatively better than normalized cut. We performed experiments on a small road network of Downtown San Francisco to demonstrate the framework effectiveness, and on three large road networks of Melbourne of different sizes to demonstrate its scalability along with effectiveness. In all the four networks, we outperform the existing techniques in terms of different performance evaluation metrics.

Chapter 4

Fast Partitioning of Road Traffic Networks Using Density Peak Graphs

The task of spatial partitioning of road networks is inherently complex. The application of our spectral clustering based $\boldsymbol{\alpha}$-Cut directly on the large road networks takes long execution time. In this chapter, we propose a robust framework for spatial partitioning of large urban road networks based on traffic measures, with an emphasis on minimizing the execution time for large networks. For a given urban road network, we aim to identify the different sub-networks or partitions that exhibit homogeneous traffic patterns internally, but heterogeneous patterns to others externally. To this end, we develop a two-stage algorithm (referred as `FaDSPa`) within our framework. It first transforms the large road graph into a well-structured and condensed density peak graph (DPG) via density based clustering and link aggregation using traffic density and adjacency connectivity, respectively. Thereafter we apply our spectral theory based $\boldsymbol{\alpha}$-Cut to partition the DPG and obtain the different sub-networks. Thus the framework applies the locally distributed computations of density based clustering to improve efficiency and the centralized global computations of spectral clustering to improve accuracy. We perform extensive experiments on real as well as synthetic datasets, and compare its performance with that of an existing road network partitioning method. Our results show that the proposed method outperforms the existing normalized cut based method for small road networks and provides impressive results for much larger networks, where other methods may face serious problems of time and space complexities.

4.1 Introduction

These days there is an increase in the frequency of traffic congestion on urban road networks, especially during the peak hours and in the city centers. This increasing congestion requires an improvement in its management by learning from its behavior to help balance the traffic flow. Usually the roads of each locality, say inside a suburb, experience a specific traffic flow pattern regardless of the global flow. For example, roads inside the city centre or any area having popular venues like a stadium or hospital, usually remain more congested than others without any such significance. Additionally, the congestion on roads connecting important places of public gatherings like airports, train stations, hospitals, and bus stops, remains comparatively higher than other locations. Thus different subnetworks of the urban road network exhibit congestion at different times. To analyze the behavior of congestion it is important for traffic management authorities to be able to partition an urban road network into different sub-networks based on the road connectivities and their congestion level, which is determined by the real traffic measures [3].

Moreover, as we move towards smart urban infrastructure, there is a growing demand for traffic-aware smart travel services, including route guidance and trip planning. These services are usually based on complex graph processing methods dealing with the road network. One way to efficiently process the execution of these services is to exploit computation in a distributed computing environment, in which the large road network is partitioned into several small sub-networks, so that queries can be focused on the relevant sub-networks [88]. Thus there exist different applications where instead of using the complete urban network, problem solving can be simplified by separately processing the smaller sub-networks that exhibit homogeneous traffic patterns inside. This leads to the important problem of traffic-based spatial partitioning of urban road networks. The application of graph partitioning on general information networks has been studied in the past [20, 28]. However, the geospatial properties of a road network associated with traffic flow patterns makes a unique kind of network [61]. The problem was recently raised in the intelligent transportation systems (ITS) community [61], where the authors proposed a normalized cut based method for partitioning road networks. While this works well for small networks, it faces serious limitations in its time and space complexity for large networks.

In Chapter 3 and our recent work [3], we proposed a method for traffic-based spatial partitioning of large road networks that outperformed existing techniques. The method comprises three different modules– road graph construction, road supergraph mining, and supergraph partitioning. The first module deals with transforming the real road network into a road graph to give it a mathematical representation. To address the problem of large number of road segments in large urban road networks, we followed a 2-level partitioning. The second module is the first level partitioning, which mines a road supergraph from the road graph with a much reduced order following a bottom-up approach. It goes through the steps of clustering feature values using $\boldsymbol{k}$-means and constructing the road supergraph. The last module of supergraph partitioning is the second level partitioning, which follows a top-down approach to split up the supergraph into multiple heterogeneous partitions that are homogeneous within. It is achieved by approximately optimizing $\boldsymbol{\alpha}$-Cut by following a spectral clustering based solution (proposed in Chapter 3). It produces supernode partitions, from which the road segment partitions are extracted. Despite obtaining good results the following issues are still outstanding, $\boldsymbol{i}$) the problem of learning the right number of clusters, while applying $\boldsymbol{k}$-means to create supernodes, is a computationally expensive task; $\boldsymbol{ii}$) when the value of $\boldsymbol{k}$ in $\boldsymbol{k}$-means is set very low, the number of supernodes is sometimes still very large, implying the relation between $\boldsymbol{k}$ and the supernodes is weak; $\boldsymbol{iii}$) the connectivity among the nodes is not considered together with their feature values when applying clustering ($\boldsymbol{k}$-means) to mine the supergraph. In this chapter, we address the above issues and present a robust framework employing both density and spectral based clustering. It is known that spectral clustering based solutions provide good results but exhibit high computational complexity [3, 135]. On the other hand, density-based methods are able to discover clusters of arbitrary shapes and are very fast. Our framework combines the advantages of spectral and density based approaches simultaneously, and also overcomes the issues that existed in our previous work [3].

Our partitioning framework aims to identify the different heterogeneous regions of an urban network that internally exhibit homogeneous traffic patterns. We propose three algorithms, `FaDPa`, `FaDPa+`, and `FaDSPa`. The main scalable algorithm `FaDSPa`, which is based on the other two, mines a *density peak graph* by identifying the density peaks from the *road graph*. Then the density peak graph is subjected to our spectral theory based $\boldsymbol{\alpha}$-Cut to obtain the set *road network partitions*. In summary, we make the following contributions in this chapter.

- We develop a fast density-based road network partitioning method `FaDPa` (extended to `FaDPa+`). It identifies the density peaks locally in the graph, and gradually grows them to form clusters. Unlike spectral clustering methods, it works very fast, and is highly suitable to large networks.
- Using `FaDPa`, we develop an efficient and effective method `FaDSPa` for partitioning small as well as large road networks. It provides an option to input a factor to control the trade-off between efficiency and partitioning quality.
- We perform extensive experiments on real as well as synthetic datasets including road networks of different sizes to establish its efficacy.

The rest of the chapter is organized as follows. Section 4.2 presents some preliminary theories followed by the problem definition and framework overview. Section 4.3 presents our density-based partitioning algorithm `FaDPa` and its extension `FaDPa+`, followed by the main algorithm `FaDSPa` in Section 4.4. Experimental results are shown in Section 4.5, followed by the chapter summary in Section 4.6.

4.2 Problem Definition and Framework Overview

This section defines the problem, and presents the framework overview.

4.2.1 Problem Definition

The problem of *partitioning road networks* addressed in this chapter is defined as splitting a given urban road network based on traffic measures into several disjoint partitions, keeping intact the associated spatial properties. The different partitions exhibit the property of intra-partition traffic *homogeneity* and inter-partition traffic *heterogeneity.* Let us suppose we have a real urban directed road network $\mathcal{N} = (\mathcal{I}, \mathcal{R})$, which is transformed into a *road graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by following the method described in Section 3.2.1 of Chapter 3. Before formally stating the problem, we present four definitions.

Definition 4.1. (**Cost of Partitioning**) While partitioning the set of nodes $\mathcal{V}$ in a road graph $\mathcal{G}$ into different partitions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, the *cost of partitioning* is defined as the aggregation of *affinity values* of all possible node pairs (v_i, v_j) for which v_i and v_j lie in different partitions in the final result, where the affinity values are measures of traffic similarity between nodes in the pairs. ■

Definition 4.2. (**Partition Volume**) Given a set of road graph partitions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, *partition volume* is defined as the aggregation of *affinity values* of all possible linked pairs (v_i, v_j) for which v_i and v_j lie in the same partition. ■

Definition 4.3. (**Partition Connectivity**) A partition $\mathcal{P}_l = (\mathcal{V}_l, \mathcal{E}_l)$ is said to be *connected* if for any node pair $(v_i, v_j) \in \mathcal{P}_l$ there exists a path from v_i to v_j (or vice versa), such that each node v_k in the path belongs to $\mathcal{V}_l$ (i.e., $v_k \in \mathcal{V}_l$). ■

The problem of traffic-based spatial partitioning of a road graph $\mathcal{G}$ is to split its node set $\mathcal{V}$ into k partitions (or subsets) $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ such that the following conditions hold.

C.1 $\bigcup_{i=1}^{k} \mathcal{P}_i = \mathcal{V}$ and $\mathcal{P}_i \bigcap \mathcal{P}_j = \emptyset$ for all $i \neq j$;

C.2 each $\mathcal{P}_i$ is *connected*, and all adjacency relations, except the cross-partition relations (inter-partition links), are maintained as in $\mathcal{G}$;

C.3 the *partition volume* of $\mathcal{G}$ is the maximum; and

C.4 the *cost of partitioning* $\mathcal{G}$ is the minimum;

In the above conditions, **C.1** is a general condition of grouping the set of nodes (or road segments) into $\boldsymbol{k}$ non-overlapping subsets, **C.2** introduces the spatial connectivity (or linkage) of nodes, **C.3** enforce the condition of intra-partition traffic homogeneity, and **C.4** enforces inter-partition traffic heterogeneity. A partitioning may not satisfy **C.3** and **C.4** together simultaneously. Optimizing one of them may lead to sacrificing the other condition. Therefore, our goal is to make an optimized trade-off between **C.3** and **C.4**.

4.2.2 Framework Overview

The task of road network partitioning is to cluster the road segments of a given road network based on their traffic measures and the associated spatial connectivities (connectivity of road segments). However, the traditional clustering algorithms, like $\boldsymbol{k}$-means, do not take care of the connectivities directly. It requires to develop ways to incorporate the connectivities during clustering in an efficient and effective manner. In the proposed framework shown in Figure 4.1, our partitioning algorithm called `FaDSPa` uses a combination of an efficient density-based clustering approach and an effective spectral clustering approach. It starts with constructing a road graph from the given road network. The graph is passed to the partitioning algorithm `FaDSPa` to obtain the set of partitions. Lastly the real road network partitions are extracted from the resulting road graph partitions.

The transformation of the real road network $\boldsymbol{\mathcal{N}}$ into a road graph $\boldsymbol{\mathcal{G}}$ is done in the beginning to give it a mathematical representation, explained as a preliminary step in Chapter 3 Section 3.2.1. Due to the large and rapidly expanding nature of urban areas, the size of an urban road network $|\boldsymbol{\mathcal{R}}|$ and the order of the corresponding road graph $|\boldsymbol{\mathcal{V}}|$ may become very large, which heavily affects the time and space complexity for partitioning $\boldsymbol{\mathcal{G}}$. To address this problem, the framework follows a two-level partitioning (`FaDSPa`), where the first level is fast and the second level produces quality partitions. The first level follows a bottom-up approach and applies a density based algorithm called `FaDPa+` to compress the large graph $\boldsymbol{\mathcal{G}}$ into a small density peak graph $\boldsymbol{\mathcal{G}^d}$ (defined later) by identifying the locally dense components. The second level partitioning follows a top-down approach to split up the density peak graph $\boldsymbol{\mathcal{G}^d}$ into multiple heterogeneous partitions that are internally homogeneous. It is achieved by approximately optimizing $\boldsymbol{\alpha}$-Cut, by following a spectral

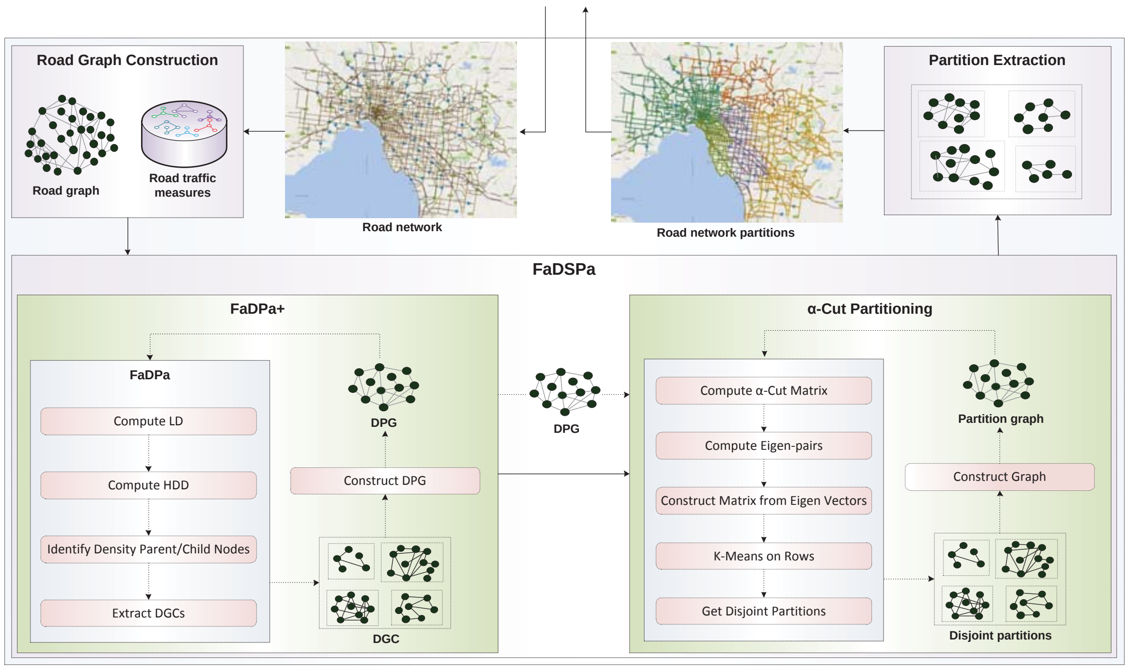


Figure 4.1: Architecture of the proposed framework

clustering based solution. It produces partitions of the density peak graph, from which the road segment partitions are extracted.

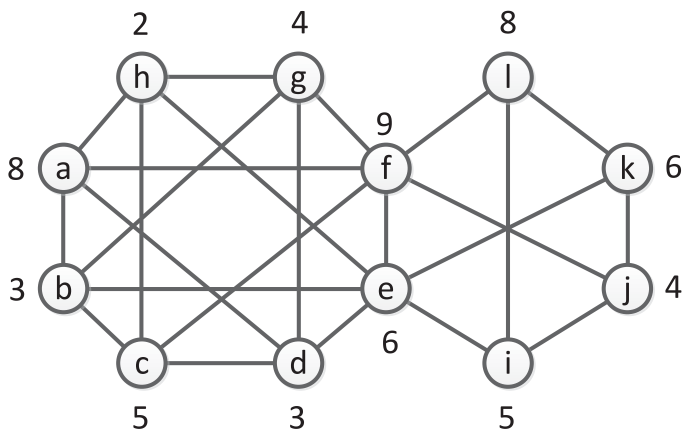
The density based `FaDPa+` is fast and thus suitable for large networks. On the other hand, the spectral based $\boldsymbol{\alpha}$-Cut produces quality partitions, but comes with high time and space complexity, and thus is suitable for small networks. Depending on the available computing resources and processing time, `FaDSPa` maintains a balance between the efficiency and accuracy of the partitioning task, by using an input parameter. If the urban network is small in size (manageable by the available resources), the task is done more by the $\boldsymbol{\alpha}$-Cut, and if it is large (beyond manageable by the available resources), the task is transferred more to `FaDPa+`. This makes `FaDSPa` effective as well as efficient in dealing with graphs of all sizes.

We propose `FaDPa` (in Section 4.3) as a fast density-based partitioning algorithm, which is further extended to `FaDPa+` (in Section 4.3.4) to partition into any desired small number of clusters, and `FaDSPa` (in Section 4.4) as a combined density and spectral based partitioning algorithm. `FaDSPa` is the main partitioning algorithm that is able to handle all small to large urban road networks, by following an appropriate balance between the density based (`FaDPa+`) and spectral based ($\boldsymbol{\alpha}$-Cut [3]) algorithms.

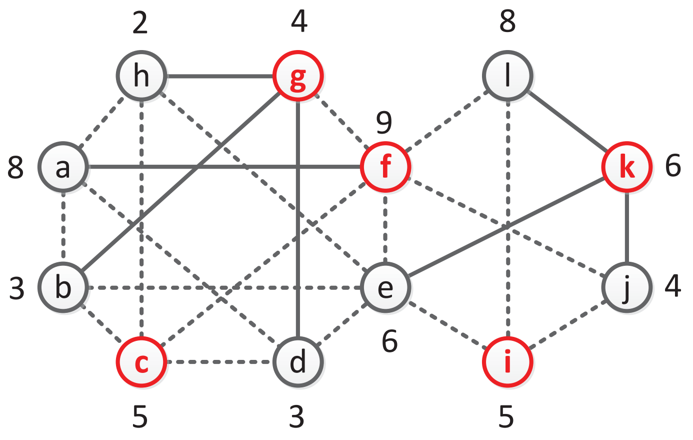
4.3 FaDPa: Fast Density-based Partitioning

The road segments inside a road sub-network or partition are linked together. Any vehicle entering into a partition through a road segment needs to go through the following segments to cross the partition or reach the destination. It makes the traffic pattern of a road segment more likely to be similar to (or dependent on) other (following or preceding) segments inside the partition. Also in each partition, locally there exist some important road segments that are spatially more closely connected to others and play a special role in the traffic movement. The road network segments including these important roads and the surrounding roads form dense components with high similarity in the traffic density, where the most important and dominating road occupies the density peak. The traffic on the surrounding roads, other than the density peak, is heavily dependent on the peak, which again have following roads that depend on these nearby roads. In this section we use this natural phenomenon of road traffic networks to propose a fast density-based network partitioning method called `FaDPa` (pronounced as *fad-paa*). It first identifies the density peaks in a network and then grows them to identify the density-based clusters.

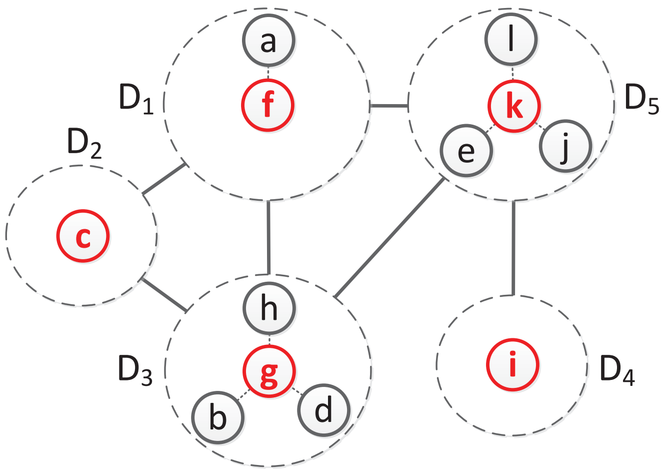
There exist density based clustering algorithms like DBSCAN [12], which are efficient, able to detect clusters of arbitrary shapes, and able to find the suitable number of clusters automatically. They identify a cluster by looking into the neighborhood of each object within a radius of a predefined threshold $\boldsymbol{\epsilon}$ distance. With each $\boldsymbol{minpts}$ (predefined) objects in the $\boldsymbol{\epsilon}$-neighborhood, a new cluster is formed. The process is carried out to find all density-connected clusters, where a density-connected cluster is defined as the maximal set of density-connected objects. The main drawback of this method is the predetermination of $\boldsymbol{\epsilon}$ and $\boldsymbol{minpts}$ thresholds, and their high sensitivity to cluster formation [13]. The method we propose here is free from these requirements. We start with presenting the main concepts and terminology, which is followed by the algorithm.



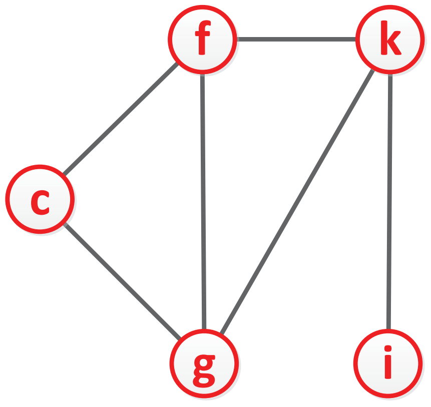
(a) Sample road graph



(b) Identification of DPNs



(c) DGCs



(d) DPG

FIGURE 4.2: Illustration of DPG construction from a graph

4.3.1 Concepts and Terminology

We use some of the ideas of [13] in Definitions 4.4 and 4.5 to find the density peaks in unlinked data, and then extend them to graph data.

Definition 4.4. (**Local Density (LD)**) Given a set of data objects $\mathcal{D} = \{d_1, d_2, ..., d_{(n_d)}\}$, the local density $\rho(d_i)$ of an object d_i is defined as the number of objects closer than a predefined distance threshold ϵ^d to d_i. It is formulated in Equation 4.1, where $dist(d_i, d_j)$ gives the distance[1] between d_i and d_j in terms of their feature values, and $\chi(.)$ is a binary function defined in Equation 4.2. ■

$$\rho(d_i) = \sum_j \chi(dist(d_i, d_j) - \epsilon^d) \tag{4.1}$$

$$\chi(x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Definition 4.5. (**Higher Density Distance (HDD)**) Given a set of data objects $\mathcal{D} = \{d_1, d_2, ..., d_{(n_d)}\}$, the higher density distance $\delta(d_i)$ of an object d_i is defined as the distance from d_i to the closest object d_j of higher *local density*. It is formulated in Equation 4.3 as the minimum distance between d_i and any other object d_j with higher density. ■

$$\delta(d_i) = \min_{\forall d_j : \rho(d_j) > \rho(d_i)} dist(d_i, d_j) \tag{4.3}$$

Definition 4.6. (**LD in Graph**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the LD $\rho^g(v_i)$ of a node v_i is defined as the number of nodes that are directly linked to v_i and closer than a predefined distance threshold ϵ^d. It is formulated in Equation 4.4, where $neigh(v_i)$ returns all the neighboring or linked nodes to v_i, $dist(v_i, v_j)$ returns the distance between v_i and v_j in terms of their feature values, and $\chi(.)$ is the same binary function defined in Equation 4.2. ■

[1]We use Gaussian based distance measure defined later in Section 4.3.2

$$\rho^g(v_i) = \sum_{\forall v_j \in neigh(v_i)} \chi(dist(v_i, v_j) - \epsilon^d) \tag{4.4}$$

Table 4.1: Distance measures in the sample road graph

	Distance measures												**LD**
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	
a	-	0.93	-	0.93	-	**0.10**	-	0.98	-	-	-	-	1
b	0.93	-	**0.35**	-	0.62	-	**0.10**	-	-	-	-	-	2
c	-	**0.35**	-	**0.35**	-	0.82	-	0.62	-	-	-	-	2
d	0.93	-	**0.35**	-	0.62	-	**0.10**	-	-	-	-	-	2
e	-	0.62	-	0.62	-	0.62	-	0.82	**0.10**	-	**0.00**	-	2
f	**0.10**	-	0.82	-	0.62	-	0.93	-	-	0.93	-	**0.10**	2
g	-	**0.10**	-	**0.10**	-	0.93	-	**0.35**	-	-	-	-	3
h	0.98	-	0.62	-	0.82	-	**0.35**	-	-	-	-	-	1
i	-	-	-	-	**0.10**	-	-	-	-	**0.10**	-	0.62	2
j	-	-	-	-	-	0.93	-	-	**0.10**	-	**0.35**	-	2
k	-	-	-	-	**0.00**	-	-	-	-	**0.35**	-	**0.35**	3
l	-	-	-	-	-	**0.10**	-	-	0.62	-	**0.35**	-	2

Example 4.1. *Figure 4.2(a) shows an example of a road graph constructed from a small road network, in which exemplary node feature values are shown beside the nodes, and Table 4.1 shows the distance measures computed for each pair of nodes. Setting the distance threshold* ϵ^d *to 0.5, the distances lower than this threshold are highlighted (bold) in the table, and the rightmost column shows the node local density as the count of these highlighted entries in each row.* ■

Definition 4.7. (**Density Parent**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *density parent* of a node v_i is defined as the linked node v_j having the closest higher *local density*, such that $\chi(dist(v_i, v_j) - \epsilon^d) = 1$. If there are multiple nodes equally close to v_i in terms of LD, then the one with the lowest $dist(v_i, v_j)$ is chosen as the parent. ■

Definition 4.8. (**Density Child**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *density children* of a node v_i is defined as the set of linked nodes $\{v_j\}$ that have v_i as their *density parent.* A node can have multiple density children. ■

Definition 4.9. (**HDD in Graph**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the HDD $\delta^g(v_i)$ of a node v_i is defined as the distance from v_i to its *density parent* v_j if v_j exists (Equation 4.5), otherwise it is the maximum of all distances between any two linked nodes (Equation 4.6). In the equations, $neigh(v_i)$ returns all the neighboring or linked nodes to v_i, $dist(v_i, v_j)$ returns the distance between v_i and v_j in terms of their feature values, and $v_k \Leftrightarrow v_l$ denotes that v_k is linked to v_l. ■

$$\delta^g(v_i) = \min_{\forall v_j} \{dist(v_i, v_j)\} \tag{4.5}$$

$$\delta^g(v_i) = \max_{\forall v_k, v_l, v_k \Leftrightarrow v_l} \{dist(v_k, v_l)\} \tag{4.6}$$

Definition 4.10. (**Density Peak Node (DPN)**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node v_j is called a *density peak node* ς_i, if v_j does not have any *density parent*. Like nodes, the DPNs also associate a feature value $\varsigma_i.f (= v_j.f)$ with them. ■

Definition 4.11. (**Density Similar**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two nodes v_i and v_j are said to be *density similar*, if they have a *density parent* and *density child* relationship, or if there is another node v_k such that v_i is *density similar* to v_k and v_k is *density similar* to v_j. Hence this relationship is both *reflexive* and *transitive*. ■

Definition 4.12. (**Graph Component (GC)**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *graph component* is defined as a subgraph in which there exists a path between any two nodes $v_i, v_j \in \mathcal{V}$ in such a way that each node v_k in the path belongs to $\mathcal{V}$. ■

Definition 4.13. (**Dense Graph Component (DGC)**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *dense graph component* D_i is defined as a *graph component* in which each pair of nodes (v_i, v_j) are *density-similar*. Every DGC must have exactly 1 DPN, which will not have any density parent, whereas all other nodes in the DGC must have. ■

Definition 4.14. (**Density Peak Graph (DPG)**) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *density peak graph* $\mathcal{G}^d$ is defined as a 3-tuple $(\mathcal{V}^d, \mathcal{E}^d, \mathcal{W}^d)$, where $\mathcal{V}^d = \{\varsigma_1, \varsigma_2, \ldots, \varsigma_{n_\varsigma}\}$ is the set of DPNs, $\mathcal{E}^d = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{n_\varepsilon}\}$ is the set of links connecting the DPNs, and $\mathcal{W}^d = \{\omega_1, \omega_2, \ldots, \omega_{n_\varepsilon}\}$ is the set of weights associated with each of the corresponding links. The links between the DPNs are established by looking into the neighborhood relationships between the corresponding DGCs. For each pair of DPNs $(\varsigma_i, \varsigma_j)$ in the DPG, if there exists a link e_k between a pair of nodes (v_p, v_q), such that ($v_p \in D_i$ and $v_q \in D_j$) or ($v_q \in D_i$ and $v_p \in D_j$), then a link ε_l is established between them. The weight of this link is set as a measure of similarity between ς_i and ς_j, i.e., $\omega_l = sim(\varsigma_i, \varsigma_j)$ (defined later in Equation 4.10). ■

Example 4.2. *In Table 4.1, the LD of node a is 1. To find its density parent, we look into the LD of all the linked nodes (i.e., b, d, f, h) that have their distance less than $\epsilon^d (= 0.5)$ (i.e., f), and select the node having the closest higher LD, which is f. Thus a becomes density child of f, and f becomes the density parent of a. After establishing this relationship, a and f are called to be density similar. In the set $\{b, d, g, h\}$, $b, d,$ and h are children of g, which makes all the nodes density similar to each other. Therefore the set forms a dense graph component. For a node, if there does not exist any linked node with higher LD, then it forms the density peak. As shown in the table, g has $b, d,$ and h as the linked nodes satisfying the ϵ^d condition, but none of them have their LD higher than g. Therefore, g becomes a density peak node. Figure 4.2(b) shows the DPNs (colored) found in the sample graph. The solid lines represent a parent-child relationship and the dotted lines represents a link in the road graph. Figure 4.2(c) shows the identified dense graph components enclosed in the circles with solid lines, where the colored nodes are the DPNs, and the links with solid lines represent the neighborhood relationship between the DGCs. Figure 4.2(d) shows the density peak graph, where the nodes are the identified DPNs linked by the neighborhood relationship.* ■

For a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the density *parent-child* relationships among the nodes can be easily established after computing their LD and HDD. According to Definitions 4.7 and 4.10, all nodes must have a density parent, unless they are DPNs. It means that except the DPNs, all other nodes in $\mathcal{V}$ can be accessed by traversing through the children of DPNs, followed by their children, and so on, until the nodes do not have any children. This traversal from a single DPN results into accessing a complete DGC, and doing this for all the DPNs, gives the complete set of DGCs, which include all the nodes in $\mathcal{V}$. It leads to the conclusion that any given $\mathcal{G}$ can be decomposed into a set of DGCs, where each of them have one DPN. These DPNs are the density peaks, which form the center of attention in a surrounding. They become nodes in the DPG $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d, \mathcal{W}^d)$, while the remaining surrounding nodes in $\boldsymbol{\mathcal{V}}$ disappear, as shown in Figure 4.2. Each DPN represents its corresponding DGC, and thus constructing a DPG from a road graph condenses the graph using the density peaks.

4.3.2 Algorithm

The algorithm (shown in Algorithm 4) starts after transforming the given road network $\boldsymbol{\mathcal{N} = (\mathcal{I}, \mathcal{R})}$ into the road graph $\boldsymbol{\mathcal{G} = (\mathcal{V}, \mathcal{E})}$ as explained in Section 3.2.1 of Chapter 3. For all the nodes $\boldsymbol{\mathcal{V}}$ in $\boldsymbol{\mathcal{G}}$, the LD (lines 3–4), and the HDD with density parent/child nodes (lines 5–19) are computed. We assume that the feature values are in Gaussian distribution[2] and define the distance measure for computing LD and HDD based on the Gaussian similarity. Equation 4.7 formulates the Gaussian similarity between two linked nodes $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$, where $\boldsymbol{\sigma^2(v) = \frac{1}{n_v} \times \sum_{i=1}^{n_v} (v_i.f - \mu^v)^2}$ is the variance of node feature values with respect to the node mean $\boldsymbol{\mu^v}$. It is a direct similarity with path length[3] 1.

$$\boldsymbol{gsim^1(v_i, v_j) = \exp\left(\frac{-(v_i.f - v_j.f)^2}{2 \times \sigma^2(v)}\right)} \tag{4.7}$$

Equation 4.8 shows the similarity with path length 2 where we multiply the $\boldsymbol{gsim^1(.)}$ of intermediate links together for each different path between $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ and get the average of all such paths. As the value of $\boldsymbol{gsim^1(.)}$ ranges from 0 to 1, its product of intermediate links also lies in the same range, and thus it also follows to $\boldsymbol{gsim^2(.)}$. This equation is generalized for path length $\boldsymbol{n}$ in Equation 4.9.

$$\boldsymbol{gsim^2(v_i, v_j) = \frac{1}{|\mathcal{V}^2_{(v_i,v_j)}|} \times \left(\sum_{v_k \in \mathcal{V}_{(v_i,v_j)}} \left(gsim^1(v_i, v_k) \times gsim^1(v_k, v_j)\right)\right)} \tag{4.8}$$

[2]In [61], the authors have used the Gaussian function in road networks, and we follow them.

[3]It refers to the number of links in the path connecting the two nodes.

$$gsim^n(v_i, v_j) = \frac{1}{|\mathcal{V}^n_{(v_i,v_j)}|} \times \Bigg(\sum_{\langle v_{k1}, v_{k2}, \ldots, v_{k(n-1)} \rangle \in \mathcal{V}^n_{(v_i,v_j)}} (gsim^1(v_i, v_{k1}) \\ \times gsim^1(v_{k1}, v_{k2}) \times \ldots \times gsim^1(v_{k(n-1),v_j})\Big)\Bigg) \quad (4.9)$$

The final similarity measure is defined in Equation 4.10, by considering all the possible path lengths up to n, where we weight the similarity terms with the harmonic series members and divide their summation by the harmonic series. Generally the shorter paths between two nodes define their associativity (or relationship) strength more accurately than the longer paths. The rationality behind using the harmonic series to define the aggregated similarity is to make the effect of shorter paths more than longer paths, proportional to the path length. All the measures $gsim^1(v_i, v_j)$, $gsim^2(v_i, v_j)$, ..., $gsim^n(v_i, v_j)$, range between 0 and 1, and so does the $sim(v_i, v_j)$.

$$sim(v_i, v_j) = \frac{gsim^1(v_i, v_j) + \frac{gsim^2(v_i,v_j)}{2} + \ldots + \frac{gsim^n(v_i,v_j)}{n}}{1 + \frac{1}{2} + \ldots + \frac{1}{n}} \quad (4.10)$$

Based on this, the distance between a pair of nodes (v_i, v_j) is defined in Equation 4.11, which again makes it range between 0 and 1.

$$dist(v_i, v_j) = 1 - sim(v_i, v_j) \quad (4.11)$$

All those nodes having their HDD value $\delta^g(v_i)$ as the maximum of all distances between a pair of linked nodes $\mathbf{max}_{\forall v_k, v_l, v_k \Leftrightarrow v_l} \{dist(v_k, v_l)\}$ are designated as a DPN (line 17). For each DPN, a search is then started for the density children, which are combined with the DPNs to form a DGC (lines 20–29). This component is grown further by looking into the density children of the children of each DPN, and so on, until they return null. Thus a DGC is the largest component that could be grown from a DPN by exploring the density children. The number of DGCs in $\mathcal{G}$ is equal to the number of DPNs, $|D| = |\mathcal{V}^d|$, and the union of all the DGCs equals to the whole graph, $\{\cup_{\forall i} D_i\} = \mathcal{G}$. These DGCs are finally accepted as the different partitions of the road graph $\mathcal{G}$ (lines 30–31).

4.3.3 Determining Distance Threshold

As mentioned earlier, the main drawback of DBSCAN is the requirement of predetermined constants, $\boldsymbol{\epsilon}$ and $\boldsymbol{minpts}$. The cluster formation is highly sensitive to these parameters; a bad value for these parameters will lead to poor results. In `FaDPa`, one of our objectives is to make the algorithm more robust against these pre-determined parameters. We have only one constant, which is the distance threshold $\boldsymbol{\epsilon^d}$ used in Equation 4.4. We consider this as a vector $\langle \boldsymbol{\epsilon_1^d, \epsilon_2^d, \ldots, \epsilon_{n_v}^d} \rangle$ of dimension $\boldsymbol{n_v}$, instead of a single constant value, where each $\boldsymbol{\epsilon_i^d}$ corresponds to the distance threshold for node $\boldsymbol{v_i}$. The value of $\boldsymbol{\epsilon_i^d}$ is computed by looking into the neighborhood of $\boldsymbol{v_i}$ locally using Equation 4.12, where $\boldsymbol{\mathcal{V}^1_{(v_i)}}$ denotes the set of nodes directly linked to $\boldsymbol{v_i}$ (with path length 1).

$$\boldsymbol{\epsilon_i^d = 1 - \frac{1}{|\mathcal{V}^1_{(v_i)}|} \times \sum_{v_j \in \mathcal{V}^1_{(v_i)}} sim(v_i, v_j)} \tag{4.12}$$

4.3.4 FaDPa+: Reducing the Number of Partitions Further

In a graph where nodes are linked among themselves, each node is exposed only to its neighboring nodes. While computing the DPNs in $\boldsymbol{\mathcal{G}}$ in Section 4.3.2, the *density parents* and *density children* relationships are established by looking into only the neighboring nodes. The DPNs obtained in this manner are based on the local connections (not on the complete node set globally). This leads to a large number of DPNs locally, and in turn a large number of DGCs. But in real situations, we may sometimes need to cluster the graph into a small number of partitions to know the global partitioning pattern. For example, in our experimental dataset $\mathtt{M_2}$ that has a graph of 53,494 nodes, the number of partitions produced by `FaDPa` is 22670; it generally depends on the number of nodes, links, and their distance weights. This number is still large. A manual analysis of these partitions would be very difficult, and the user may want to have far fewer partitions numbering less than 100 or even 10.

To further reduce the number of partitions generated by `FaDPa` as per the user requirements, we propose an extended algorithm named `FaDPa+` (shown in Algorithm 5). In

Algorithm 4: FaDPa (Road graph $\mathcal{G}$, Distance threshold ϵ^d)

```
1  $\mathcal{V}^d \leftarrow$ instantiate an empty set of DPNs;
2  $stack \leftarrow$ initialize a stack;
   // compute LD
3  for $i \leftarrow 1$ to $(n_r)$ do
4      $\rho^g(v_i) = \sum_{\forall v_j \in neigh(v_i)} \chi(dist(v_i, v_j) - \epsilon^d)$;
   // compute HDD, and density parent/child nodes
5  for $i \leftarrow 1$ to $(n_r)$ do
6      $\delta^g(v_i) \leftarrow -1$ ;                                // initialize with null
7      forall $v_j \in neigh(v_i)$ do
8          if $\rho^g(v_j) > \rho^g(v_i)$ then
9              if $\delta^g(v_i) = -1$ then
                   // assign distance
10                 $\delta^g(v_i) \leftarrow dist(v_i, v_j)$;
11                 $parentnode \leftarrow v_j$;
12             else if $\delta^g(v_i) > dist(v_i, v_j)$ then
                   // overwrite HDD with the minimum distance
13                 $\delta^g(v_i) \leftarrow dist(v_i, v_j)$;
14                 update, $parentnode \leftarrow v_j$;
15     if $\delta^g(v_i) = -1$ then
           // Equation 4.6
16         $\delta^g(v_i) = \max_{\forall v_k, v_l, v_k \Leftrightarrow v_l} \{dist(v_k, v_l)\}$;
17         $\mathcal{V}^d \leftarrow \mathcal{V}^d \cup \{v_i\}$ ;              // $v_i$ found as DPN $\varsigma_i$
18     else
19         Set $v_i$ as child of $parentnode$, and $parentnode$ as parent of $v_i$;
   // extract DGCs
20 $D \leftarrow$ instantiate an empty set of DGCs;
21 forall $\varsigma_i \in \mathcal{V}^d$ do
22     push $\varsigma_i$ into $stack$;
23     $d \leftarrow \{\phi\}$ ;                            // instantiate an empty DGC
24     while $stack$ is not empty do
25         $node \leftarrow$ pop out from $stack$;
26         $d \leftarrow d \cup \{node\}$ ;          // add density-similar nodes to the DGC
27         forall $childnode \in child(node)$ do
28             push $childnode$ into $stack$;
29     $D \leftarrow D \cup \{d\}$;
   // extract partitions from DGCs
30 $P \leftarrow$ extract partitions from $D$;
31 return $P$;
```

this extension, the DPNs obtained from $\mathcal{G}$ by FaDPa are used to construct a DPG

$\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d, \mathcal{W}^d)$ (defined in Definition 4.14). The new graph $\mathcal{G}^d$ becomes a condensed form of the original graph $\mathcal{G}$ where some local information is merged together. Considering $\mathcal{G}^d$ as the main graph now, the DPNs are further obtained using `FaDPa`. This time the number of DPNs would reduce further, and so would the number of partitions. These steps of constructing the DPG and identifying the DPNs are repeated alternatively until the number of DPNs becomes lower than the predefined number of partitions ϵ_p (lines 2–4). Thereafter the DGCs are obtained in the same way as explained earlier and accepted as the different partitions of the road graph (line 5).

Algorithm 5: `FaDPa+` (Road graph $\mathcal{G}$, Distance threshold ϵ^d, Number of partitions threshold ϵ_p)

```
1 $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d, \mathcal{W}^d) \leftarrow \mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$;
2 while $|\mathcal{V}^d| > \epsilon_p$ do
3     partition set $P \leftarrow$ FaDPa($\mathcal{G}^d, \epsilon^d$);
4     $\mathcal{G}^d \leftarrow$ construct DPG from $P$;
5 return $P$;
```

4.4 FaDSPa: Fast Density and Spectral based Partitioning

`FaDPa+`, proposed in the previous section, is a complete road network partitioning algorithm in itself. It grows the clusters in arbitrary shapes by first identifying the dense components, and is able to work efficiently. In contrast, spectral clustering methods have been a major focus in the literature due to their ability to produce high quality results. Due to its high computational complexity, spectral clustering is often not used directly in large-scale data mining problems. However, attempts are being made to improve the efficiency of spectral clustering [135]. In this section, we propose `FaDSPa` (pronounced as *fad-spaa* and shown in Algorithm 6) as an efficient as well as effective road network partitioning algorithm that employs both density-based (`FaDPa`) (lines 2–4) and spectral-based (α-Cut) (line 5) theories.

Algorithm 6: `FaDSPa` (Road graph $\mathcal{G}$, Distance threshold ϵ^d, Compression threshold ϵ_c, Number of desired partitions k)

```
1 $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d, \mathcal{W}^d) \leftarrow \mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$;
  // density based clustering
2 while $|\mathcal{V}^d| > \epsilon_c$ do
3     partition set $P \leftarrow$ FaDPa($\mathcal{G}^d, \epsilon^d$) ;                    // Algorithm 4
4     $\mathcal{G}^d \leftarrow$ construct DPG from $P$;
  // spectral based clustering
5 partition set $P \leftarrow \alpha$-Cut Partitioning($\mathcal{G}^d, k$) ;                // Algorithm 7
6 return $P$;
```

4.4.1 Mining DPG

`FaDSPa` starts by mining a road DPG $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d, \mathcal{W}^d)$ from the road graph $\mathcal{G}$. It uses `FaDPa+` to mine this DPG, in which ϵ_c is a compression threshold that determines the number of DPNs ($|\mathcal{V}^d|$). The value of ϵ_c is pre-defined depending on the available computing resources and the time that we can afford to spend in order to obtain good partitioning results. `FaDPa+` compresses the graph until $|\mathcal{V}^d|$ becomes lower than or equal to ϵ_c. $\mathcal{G}$ is normally a sparse graph in nature. $\mathcal{G}^d$ is mined by identifying the dense components in $\mathcal{G}$ in arbitrary shapes, which reduces the sparsity of the graph as well as the overhead in dealing with that sparsity. The resulting graph $\mathcal{G}^d$ becomes a condensed form of the road graph $\mathcal{G}$, which is much smaller in order. As the level of compression of $\mathcal{G}$ is controlled by ϵ_c, there exist two extremes. At one end, ϵ_c could be set to $|\mathcal{V}|$, and on the other end, it could be the number of required partitions k. The first case makes it `FaDPa+`, whereas the second case makes it the α-Cut spectral clustering algorithm. Thus `FaDSPa` provides a good balance of `FaDPa+` and α-Cut, and is a generalization of these two algorithms. After mining the DPG, a preliminary level of grouping of road segments has already happened in the form of DGCs in the DPG ($\mathcal{G}^d$) in a bottom-up manner. Thereafter the spectral based partitioning algorithm α-Cut is applied on the compressed graph $\mathcal{G}^d$, instead of the large graph $\mathcal{G}$, in a top-down manner.

Algorithm 7: α-Cut Partitioning (DPG $\mathcal{G}^d$, number of desired partitions k)

```
1  $A \leftarrow$ adjacency matrix of $\mathcal{G}^d$;
2  $D \leftarrow$ degree matrix of $\mathcal{G}^d$;
   // repeated partitioning to obtain $k$ partitions
3  repeat
4      $M \leftarrow \left(\frac{(1^T D)^T (1^T D)}{1^T D 1} - A\right)$ ;          // get the $\alpha$-Cut matrix
5      $\bigcup_{i=1}^{n_\varsigma} \{(y_i, \lambda_i)\} \leftarrow$ get eigenvector and eigenvalue pairs of $M$;
6      sort eigenvalues $\lambda_i$ to have $\lambda_{n_\varsigma} \leq \lambda_{n_\varsigma - 1} \leq \ldots \leq \lambda_1$;
7      select $\{\lambda_{n_\varsigma}, \lambda_{n_\varsigma - 1}, \ldots, \lambda_{n_\varsigma - k+1}\}$ eigenvalues and corresponding eigenvectors
         $\{y_{n_\varsigma}, y_{n_\varsigma - 1}, \ldots, y_{n_\varsigma - k+1}\}$;
8      generate matrix $Y_{n_\varsigma \times k} = \begin{pmatrix} y_1 & y_2 & \ldots & y_k \end{pmatrix}$;
9      $Z \leftarrow$ row normalize $Y$;
10     $\{z_1, z_2, \ldots, z_{n_\varsigma}\} \leftarrow$ get row vectors of $Z$;
11     $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\} \leftarrow$ k-means $(\{z_1, z_2, \ldots, z_{n_\varsigma}\}, k)$;
12     $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{k'}\} \leftarrow$ get disjoint partitions from $\mathcal{C}$ ;   // resulting set of
         partitions
13     if $k'$ is not equal to $k$ then
           // construct a graph from the partitions and consider this as the
              new graph for partitioning
14         $n_\varsigma \leftarrow k'$;
15         $\mathcal{G}^p \leftarrow$ construct partition graph from $\mathcal{P}$;
16         $A'_{n_\varsigma \times n_\varsigma} \leftarrow$ adjacency matrix of $\mathcal{G}^p$;
17         $D'_{n_\varsigma \times n_\varsigma} \leftarrow$ degree matrix of $\mathcal{G}^p$;
18         $A \leftarrow A'$;
19         $D \leftarrow D'$;
20 until number of partitions in $\mathcal{P}$ equals to $k$;
21 return $\mathcal{P}$ ;          // return the partitions when their number equals to $k$
```

4.4.2 The Spectral based α-Cut

Algorithm 7 presents the complete partitioning method using α-Cut. The detailed theoretical derivations of α-Cut are shown in Chapter 3. It starts with getting the adjacency and degree matrices in lines 1 and 2. The steps 4–19 are repeatedly performed until the resulting number of partitions equals to k. The α-Cut matrix is computed from the adjacency and degree matrices in line 4 and eigen-decomposed in line 5. Lines 6–7 select the k smallest eigenvalues and corresponding eigenvectors. Ideally the indicator vectors should have only binary values, but the actually obtained indicator vectors are in fact the relaxed vectors and do not follow a binary pattern. Due to the lack of concrete information about clusters, it becomes another problem to separate the k clusters. We assume that the clusters are well-separated in the $\boldsymbol{k}$-dimensional eigenspace, which is a general assumption in spectral clustering [27], and use the eigenvectors (or indicator vectors) to generate a matrix $\boldsymbol{Y}$ of $\boldsymbol{n_\varsigma} \times \boldsymbol{k}$ dimensions (line 8). It is then row-normalized using Equation 4.13 to have row-vectors $\boldsymbol{z_i}$ of unit length giving the final matrix $\boldsymbol{Z}$ (line 9). Each row-vector $\boldsymbol{z_i}$ represents a DPN $\boldsymbol{\varsigma_i}$. The set of row-vectors are used to cluster the DPNs by applying $\boldsymbol{k}$-means to find a set of $\boldsymbol{k}$ clusters $\boldsymbol{\mathcal{C}} = \{\boldsymbol{\mathcal{C}_1}, \boldsymbol{\mathcal{C}_2}, \dots, \boldsymbol{\mathcal{C}_k}\}$ (lines 10–11), where each cluster $\boldsymbol{\mathcal{C}_i}$ comprises one or more row vectors (DPNs) in $\boldsymbol{Z}$. The DPNs inside each cluster are linked together as they exist in the DPG. Upon linking them, sometimes more than one connected component may be found inside a single cluster. As these multiple connected components within a single cluster are disjoint, they can not become part of the same partition. These connected components are extracted from each cluster to form disjoint partitions (line 12).

$$\boldsymbol{Y} = \begin{pmatrix} y_{11} & y_{21} & \cdots & y_{k1} \\ y_{12} & y_{22} & \cdots & y_{k2} \\ | & | & & | \\ y_{1n} & y_{2n} & \cdots & y_{mn} \end{pmatrix} = \begin{pmatrix} — & z_1^T & — \\ — & z_2^T & — \\ & \vdots & \\ — & z_n^T & — \end{pmatrix} = \boldsymbol{Z} \tag{4.13}$$

$$\text{where,} \quad \boldsymbol{z_i} = \frac{1}{\sqrt{\sum_{j=1}^{k} y_{ji}^2}} (y_{1i}, y_{2i}, \dots, y_{mi})^T$$

Depending on the data, the number of disjoint partitions may sometimes be large, which would yield the partition set from $\boldsymbol{\mathcal{C}}$ as $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \dots, \boldsymbol{\mathcal{P}_{k'}}\}$, where $\boldsymbol{k'} \geq \boldsymbol{k}$. These $\boldsymbol{k'}$ partitions may be accepted as the final result. However, if the requirement to have exactly $\boldsymbol{k}$ partitions is strict, Shi and Malik [16] described two approaches to achieve this, which are greedy pruning and global recursive bipartitioning. The greedy pruning approach iteratively merges the two nearest partitions optimizing the defined graph cut, until it results in a total of $\boldsymbol{k}$ partitions. In contrast, the global recursive bipartitioning approach generates a condensed graph where each partition forms a node and adjacent partitions are connected by weighted links with $\boldsymbol{W(\mathcal{P}_i, \mathcal{P}_j)}$ as the weight, and is recursively bipartitioned until it results in a total of $\boldsymbol{k}$ partitions. For large $\boldsymbol{k'}$ values, the greedy pruning approach is computationally intensive. On the other hand, as the global recursive approach bipartitions each time, it would yield a balanced set of partitions only when $\boldsymbol{k}$ follows the pattern $\boldsymbol{2^i}$ for any value of $\boldsymbol{i}$. For example, if the value of $\boldsymbol{k}$ is 3, firstly the graph is bipartitioned to get 2 partitions, and then only one of them has to be bipartitioned to get a total of 3 partitions, whereas the other partition remains as it is. In this way it generates one large partition and two small partitions, where the large partition is approximately double in size than the small ones. Moreover, the selection of the large partition that is to be bipartitioned first, is either arbitrary or some additional condition has to be applied to decide this.

To avoid these complexities and make the method efficient, we follow a repeated partitioning approach where the interleaved steps of partitioning and constructing a new graph each time from the obtained partitions are repeated until we obtain exactly $\boldsymbol{k}$ partitions (lines 4–12). In each repetition, we construct a new graph from the set of partitions, by considering each partition as a node and their connectivity via the nodes belonging to them as links (lines 14–15). The feature value of the nodes (formed from the partitions obtained in the last iteration) is assigned as the average of feature values of all nodes belonging to the respective partitions (old partitions), based on which the link weights are assigned. In lines 16–17, we get its adjacency and degree matrices. These matrices are considered to compute the $\boldsymbol{\alpha}$-Cut matrix for the next iteration of partitioning. Finally the $\boldsymbol{k}$ partitions are returned at the end (line 21).

4.4.3 Computational Complexity

The algorithm `FaDSPa` comprises successive applications of `FaDPa` to mine the DPG of desired order, followed by $\boldsymbol{\alpha}$-Cut. The computational complexity of `FaDPa` is O($\boldsymbol{n^2}$) for computation of $\boldsymbol{\rho^g(v_i)}$ and $\boldsymbol{\delta^g(v_i)}$, after which the partitions are extracted using a stack. Thus the overall computational complexity of `FaDPa` becomes O($\boldsymbol{n^2}$), which is applied multiple times but still much less than $\boldsymbol{n}$, thereby making it $\approx$ O($\boldsymbol{n^2}$). In $\boldsymbol{\alpha}$-Cut, the eigen-decomposition task is done in $\mathbf{O}(\boldsymbol{n^3})$ time in general and $\mathbf{O}(\boldsymbol{n^2})$ time for sparse matrices. The application of $\boldsymbol{k}$-means on row-vectors to find the clusters costs $\mathbf{O}(\boldsymbol{tnk^2})$, where $\boldsymbol{t}$ is the number of iterations required to reach the convergence. In these costs,

$\boldsymbol{n} = \boldsymbol{n}_\varsigma$ when the spectral clustering is applied on the DPG, and $\boldsymbol{n} = \boldsymbol{n}_r$ when it is applied directly on the road graph.

4.4.4 Relation with Modularity

Definition 4.15. (**Modularity**) The modularity of a set of graph partitions $Q(\mathcal{P})$ [17] is defined as the difference between the observed and expected fraction of links within a partition, and is formulated as Equation 4.14. ■

$$Q(\mathcal{P}) = \sum_{i=1}^{k} \left[\frac{W(\mathcal{P}_i, \mathcal{P}_i)}{W(\mathcal{V}, \mathcal{V})} - \left(\frac{W(\mathcal{P}_i, \mathcal{V})}{W(\mathcal{V}, \mathcal{V})} \right)^2 \right] \tag{4.14}$$

Larger modularity values are correlated with better graph partitioning. To maximize modularity while partitioning a graph, in [17] the authors presented a spectral clustering solution. They showed that the partitioning can be obtained using the k largest eigenvalues and corresponding eigenvectors obtained after eigen-decomposition of a derived matrix called Q-Laplacian [17]. This matrix actually equals to the negative of our α-Cut matrix derived in Chapter 3. As we obtain the partitioning by selecting the k smallest eigenvalues and corresponding eigenvectors, both the techniques result into the same set of eigenvalues and eigenvectors, and thus the same partitioning. It means that the minimization of α-Cut approximately maximizes the modularity.

4.5 Experimental Evaluation

Although there exist many works on general graph partitioning, we compare our results to a recent work [61] on the same problem, for a specific comparison. In addition we also compare with the results obtained by replacing α-Cut by normalized cut in `FaDSPa` to show the effectiveness of α-Cut. Section 4.5.3 presents our experiments on small road networks, where we compare the results obtained by α-Cut, normalized cut, `FaDSPa`, [61], and `FaDPa+`. Section 4.5.4 presents our experiments on the real SCATS dataset, where we compare the results obtained by the proposed `FaDSPa` and a modified version of `FaDSPa` (by replacing α-Cut by normalized cut in `FaDSPa`). Section 4.5.5 presents our experiments on large networks, where we show the performance of `FaDPa+` and `FaDSPa` for varying values of the compression threshold ϵ_c to understand the trade-off between efficiency and accuracy.

4.5.1 Datasets

We perform experiments on five datasets of different sizes including both real data and synthetic data generated on real road networks. Table 4.2 shows the statistics of all these datasets. The real data ($\mathtt{M}_s$) is recorded by the Sydney Coordinated Adaptive Traffic System (SCATS)[4] from the Melbourne road networks provided to us by VicRoads[5]. This dataset is an accumulation of the traffic records of individual road segments for each signal cycle from 1st Jan 2011 to 1st Jan 2013. The considered Melbourne network consists of 7245 road segments and 2928 intersection points, where the traffic measures are logged by the installed sensors, respective to each lane of road segments at the SCATS sites. The traffic measures include traffic volume (number of vehicles crossing a road segment during the green time) and degree of saturation (the ratio of the effectively used green time to the total available green time). In this dataset we consider the degree of saturation as feature value of the road segments, as it gives an indication of the traffic density. The degree of saturation measure for each road segment is computed by taking the average of this measure of all the different lanes that are part of the referred road segment.

The other datasets include synthetic data generated on real small and large road networks. The traffic on the small network ($\mathtt{D}_1$), shared by the authors of [61], is based on a micro-simulation performed for 4 hours at 120 time intervals of 2 minutes. At each time point t, the traffic density on each road segment is computed in terms of number of vehicles per meter. For large road network, we consider the city of Melbourne with three sets of data, $\mathtt{M}_1$, $\mathtt{M}_2$, and $\mathtt{M}_3$. $\mathtt{M}_1$ is the road network of the 6.6 sq. miles CBD area consisting of 10,096 intersection points and 17,206 directed road segments. $\mathtt{M}_2$ and $\mathtt{M}_3$, larger than $\mathtt{M}_1$, is the road network of the CBD and adjoining areas in a total of 31.5 and 42.03

[4]SCATS is a fully adaptive urban traffic control system developed in Australia in 1970. It manages the signal phases (cycle times, phase splits and offsets) of the traffic signals dynamically in real-time, based on the traffic data collected by the vehicle sensors (inductive loops) installed within road pavements of each traffic signal.

[5]https://www.vicroads.vic.gov.au

sq. miles, consisting of 28,465 and 42,321 intersection points, and 53,494 and 79,487 directed road segments respectively. These road segments are obtained by considering all the two-way road segments as two different one-way road segments. The traffic data for the large networks is generated by a web-based[6] random road traffic generator MNTG [133, 136]. It populates vehicles on a selected real road network, which keep on moving for a time duration on the roads. We populate $\mathtt{M_1}$, $\mathtt{M_2}$, and $\mathtt{M_3}$ by 25,246, 62,300, and 84,999 vehicles respectively, and obtain their trajectories for 100 continuous timestamps. The trajectories are sequences of 100 or less ⟨latitude,longitude⟩ pairs corresponding to vehicle positions at each timestamp. A self-designed program is used to map their positions to corresponding road segments, and compute the traffic density of road segments (in terms of vehicles/meter) at each point of time. While doing this, after each interval of 10 timestamps, each vehicle is considered as a different one and its updated position is recounted to compute the density. Thus it makes $\boldsymbol{t}$ range from 1 to 10, and in this work we experiment with $\boldsymbol{t = 1}$. It is done to make the network more dense, and reflect the flow speed on corresponding road segments. The count is then divided by the road segment length to get the average traffic density in terms of vehicles per meter.

Table 4.2: Dataset statistics

Dataset	**Place**	**Area (sq ml)**	**# Road segments**	**# Intersection points**
		Real SCATS data on real road network		
$\mathtt{M_s}$	Melbourne	627.5	7245	2928
		Synthetic data generated on real road network		
$\mathtt{D_1}$	Downtown San Francisco	2.5	420	237
$\mathtt{M_1}$	CBD Melbourne	6.6	17,206	10,096
$\mathtt{M_2}$	CBD(+) Melbourne	31.5	53,494	28,465
$\mathtt{M_3}$	Melbourne	42.03	79,487	42,321

[6]It can be accessed through `http://mntg.cs.umn.edu/tg/`

4.5.2 Evaluation Metrics

The partitioning framework is evaluated using metrics that quantify the quality of the results from different perspectives. The problem defined in Section 4.2.1 intends to achieve four different conditions. As we obtain results in the form of disjoint and connected road network partitions (connected components), **C.1** and **C.2** are automatically fulfilled. **C.3** which enforces intra-partition homogeneity is evaluated by the *intra* metric defined in Equation 4.15. For each partition, it computes the intra-partition distance as the average absolute distance between the pair of nodes, and then takes the average of that computed for all the partitions. Lower values of ***intra*** indicate better partitioning.

$$\boldsymbol{Intra(\mathcal{P})} = \frac{\mathbf{1}}{|\boldsymbol{\mathcal{P}}|} \times \sum_{\boldsymbol{\mathcal{P}_i \in \mathcal{P}}} \frac{\sum\limits_{\substack{\boldsymbol{v_p, v_q \in \mathcal{P}_i} \\ \boldsymbol{p \neq q}}} \mathbf{abs}(\boldsymbol{v_p.f - v_q.f})}{|\boldsymbol{\mathcal{P}_i}| \cdot (|\boldsymbol{\mathcal{P}_i}| - \mathbf{1})} \tag{4.15}$$

C.4 which enforces inter-partition heterogeneity is evaluated by the *inter* metric defined in Equation 4.16, where $\boldsymbol{\mathcal{P}_i} \overset{\text{adj}}{\longleftrightarrow} \boldsymbol{\mathcal{P}_j}$ denotes the set of adjacency relationships[7]. It is the average of inter-partition distances between each pair of spatially adjacent partitions, where the inter-partition distance is the average absolute distance between nodes from the respective pair of adjacent partitions. Higher values of ***inter*** indicate better partitioning.

$$\boldsymbol{Inter(\mathcal{P})} = \frac{\mathbf{1}}{\left|\boldsymbol{\mathcal{P}_i} \overset{\text{adj}}{\longleftrightarrow} \boldsymbol{\mathcal{P}_j}\right|} \times \sum_{\substack{\boldsymbol{\mathcal{P}_i, \mathcal{P}_j \in \mathcal{P}} \\ \boldsymbol{\mathcal{P}_i} \overset{\text{adj}}{\longleftrightarrow} \boldsymbol{\mathcal{P}_j}}} \frac{\sum\limits_{\boldsymbol{v_p \in \mathcal{P}_i}} \sum\limits_{\boldsymbol{v_q \in \mathcal{P}_j}} \mathbf{abs}(\boldsymbol{v_p.f - v_q.f})}{|\boldsymbol{\mathcal{P}_i}| \cdot |\boldsymbol{\mathcal{P}_j}|} \tag{4.16}$$

We also evaluate the overall partitioning using average NcutSilhouette (ANS) measure defined in [61] especially for partition evaluation. It is derived from the standard Silhouette measure used for cluster evaluation. NS between a pair of partitions

[7]A pair of partitions $\boldsymbol{\mathcal{P}_i}$ and $\boldsymbol{\mathcal{P}_j}$ are said to be *adjacent*, if there exists at least one link connecting nodes $\boldsymbol{v_p}$ and $\boldsymbol{v_q}$ such that $\boldsymbol{v_p} \in \boldsymbol{\mathcal{P}_i}$ and $\boldsymbol{v_q} \in \boldsymbol{\mathcal{P}_j}$

$\boldsymbol{(\mathcal{P}_i, \mathcal{P}_j)}$ is calculated using Equation 4.17, and the quality of each individual partition is evaluated using $\boldsymbol{NS(\mathcal{P}_i)}$ defined in Equation 4.18, where $\boldsymbol{NSN(\mathcal{P}_i, \mathcal{P}_j) = \min\{NS(\mathcal{P}_i, \mathcal{P}_x) | \mathcal{P}_x \in neighbor(\mathcal{P}_j)\}}$.

$$\boldsymbol{NS(\mathcal{P}_i, \mathcal{P}_j) = \frac{\sum_{v_p \in \mathcal{P}_i} \sum_{v_q \in \mathcal{P}_j} (v_p.f - v_q.f)^2}{|\mathcal{P}_i| \times |\mathcal{P}_j|}} \tag{4.17}$$

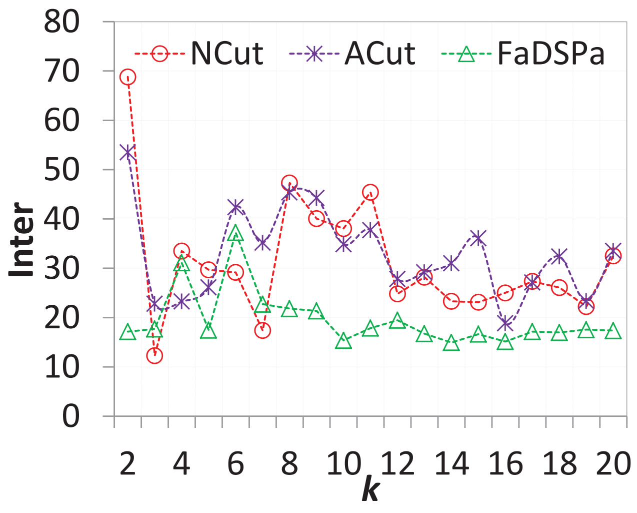
$$\boldsymbol{NS(\mathcal{P}_i) = \frac{NS(\mathcal{P}_i, \mathcal{P}_i)}{NSN(\mathcal{P}_i, \mathcal{P}_j)}} \tag{4.18}$$

Average NS (ANS) is computed as the average of $\boldsymbol{NS(\mathcal{P}_i)}$ for all $\boldsymbol{\mathcal{P}_i \in \mathcal{P}}$. A value less than 1 indicates a good partitioning, and lower values indicate better partitioning.

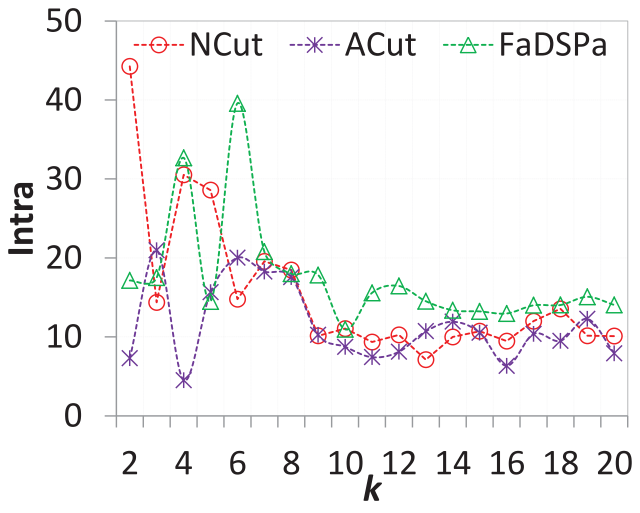
4.5.3 Experimental Results on Small Networks

We perform experiments on the small road network $\mathtt{D_1}$ to compare the partitioning quality of our $\boldsymbol{\alpha}$-Cut, `FaDPa+` and `FaDSPa`, with other state-of-the-art techniques (normalized cut and [61]) using performance evaluation metrics listed in Section 4.5.2, and demonstrate their effectiveness.

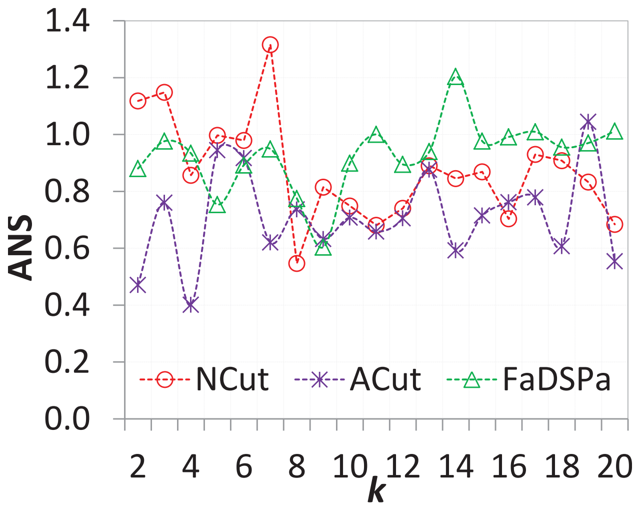
Quality comparison of `NCut`, $\boldsymbol{\alpha}$-`Cut`, and `FaDSPa`: We consider normalized cut as the baseline, and show our comparative results. Figure 4.3 shows the complete results obtained by $\boldsymbol{\alpha}$-Cut (`ACut`) and `FaDSPa` in comparison to normalized cut (`NCut`). We present the results in terms of ***inter***, ***intra*** and ANS in Figures 4.3(a), 4.3(b) and 4.3(c) respectively, for the number of partitions $\boldsymbol{k}$ ranging from 2 to 20. The ANS measure quantifies the overall partitioning quality. In terms of ANS, `ACut` of our framework outperforms `NCut` for most of the values of $\boldsymbol{k}$, whereas `NCut` outperforms `FaDSPa` for most of the values. This is expected because `FaDSPa` is of lower complexity and suitable for large networks. In terms of *inter*, which quantifies inter-partition heterogeneity, we observe that `ACut` performs similar to `NCut`. However, both of them outperform `FaDSPa` for all $\boldsymbol{k \geq 8}$, and for $\boldsymbol{k \leq 7}$ sometimes `FaDSPa` outperforms `NCut` and `ACut`. In terms of *intra*, which quantifies intra-partition homogeneity, `ACut` outperforms `NCut` for most of the values except $\boldsymbol{k} =$ 3, 6, 13, 14, and 19, whereas both of them outperform `FaDSPa` for all $\boldsymbol{k \geq 6}$. These results



(a) Inter



(b) Intra



(c) ANS

FIGURE 4.3: Road graph and DPG partitioning results in small networks

of FaDSPa are obtained by setting the compression threshold $\boldsymbol{\epsilon_c}$ to the number of DPNs obtained after the first round of FaDPa+, which is 109.

Experiments with FaDPa+: FaDPa+, as proposed in this chapter, does not provide the option to input the value of $\boldsymbol{k}$. It has a parameter called the number of partitions threshold $\boldsymbol{\epsilon_p}$. The algorithm merges the partitions on the basis of their local densities, until $\boldsymbol{k}$ is lower than or equal to $\boldsymbol{\epsilon_p}$ for the first time. Thus its $\boldsymbol{k}$ can be any value closest to and lower than or equal to $\boldsymbol{\epsilon_p}$. In our experiment on this dataset it started with 420 nodes, and after the first round of FaDPa it gave 109 partitions with 14.61, 7.85, and 0.74, as their *inter*, *intra* and ANS measures respectively. After the second round, it gave 13 partitions with the values as 20.69, 16.11, and 1.00, respectively, and after the next round all the partitions were merged to a single partition. Looking into the ANS values, it shows that the quality of 109 partitions are better than the 13 partitions obtained after the second round. However, Figure 4.3 shows that the best clustering is obtained at lower values (e.g., $\boldsymbol{k}$ = 5, 6 and 8, by different methods). To know how FaDPa+ behaves for these lower $\boldsymbol{k}$, we merged the density-closest partitions one by one, and found the best partitioning at $\boldsymbol{k = 5}$. The performance metric values are found as 42.16, 16.53, and 0.68, respectively.

Summary of comparisons: The overall partitioning quality is evaluated by ANS, which considers both the inter-partition heterogeneity and intra-partition homogeneity simultaneously. Lower values indicate a better partitioning. In Figure 4.3(c), ACut is lower than NCut at most values of $\boldsymbol{k}$, whereas FaDSPa is mostly above both of them. In [61], the authors used the ANS measure to learn the number of optimal partitions. They accept the value of $\boldsymbol{k}$ that leads to the ANS minimum as the optimal number of partitions, which in this case is 8 for NCut, 4 for ACut, and 9 for FaDSPa. We observe in the figure that the minimum of ACut is the lowest followed by NCut, and that of FaDSPa is the highest. It shows the accuracy of the partitioning task by these three methods. ACut performs the best, followed by NCut, and both of them are better than FaDSPa. Figure 4.4 and Table 4.3 show the obtained ANS measures by ACut, FaDSPa, FaDPa+, NCut, in comparison to the existing work [61]. Our methods ACut and FaDSPa are lower (and thus perform better) than [61]. In the methods, we determine the optimal value of $\boldsymbol{k}$ by repeatedly obtaining the results for a range of $\boldsymbol{k}$ and comparing their ANS measure. It can also be an application dependent issue, as a small $\boldsymbol{k}$ produces partitions of coarse granularity and this granularity becomes finer with an increasing $\boldsymbol{k}$.

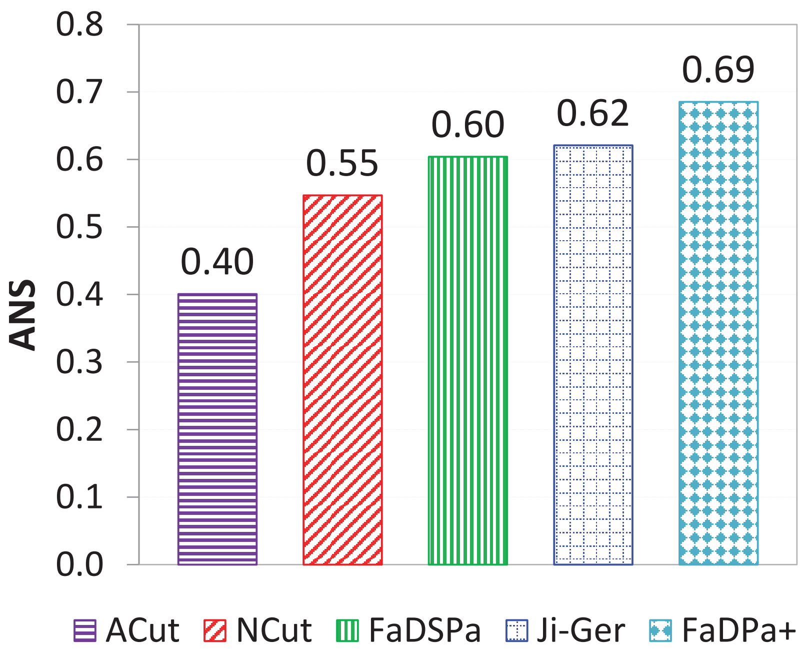


FIGURE 4.4: Overall comparison of partitioning results in small networks

TABLE 4.3: Overall quality of partitioning

	`ACut`	`NCut`	`FaDSPa`	`Ji-Ger`	`FaDPa+`
ANS	0.4009	0.5470	0.6041	0.6210	0.6853
k	4	8	9	3	5

Even though the results of `FaDSPa` do not look very impressive (in comparison to $\boldsymbol{\alpha}$-Cut or `NCut`), its advantage is that it can efficiently handle large networks while simultaneously maintaining the quality. The spectral based algorithms face time and space complexity issues, whereas the density based algorithms compromise the partitioning accuracy.

4.5.4 Experimental Results on Real Data

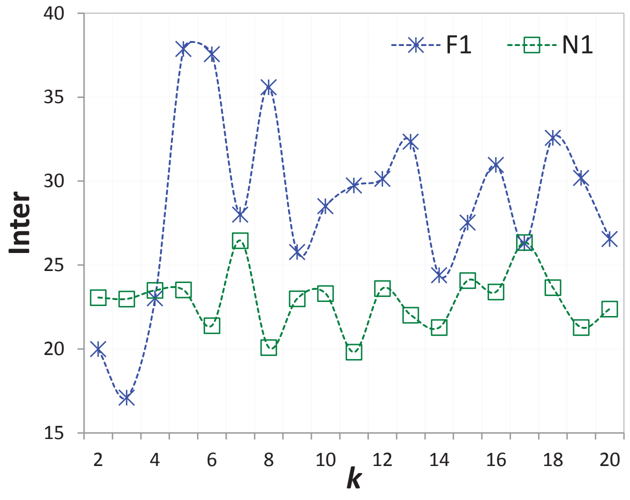
We perform experiments on real data to see the applicability of the proposed method in real environments. For this we consider the SCATS data $\mathtt{M}_{\boldsymbol{s}}$, described in Section 4.5.1. Through our experiments on this dataset, we show the results obtained by the proposed `FaDSPa` algorithm and also compare them with those obtained by replacing $\boldsymbol{\alpha}$-Cut by normalized cut in `FaDSPa`.

Different schemes of `FaDSPa`: Before going further, we explain the different schemes we have used to present our result insights. We applied the proposed method `FaDSPa` in different ways, which are denoted by F$\langle \boldsymbol{number} \rangle$. This stands for the method when `FaDSPa` is applied by repeatedly running `FaDPa+` $\langle \boldsymbol{number} \rangle$ number of times forming hierarchical groups, before passing the control to $\boldsymbol{\alpha}$-Cut. Thus F1 applies one *round*[8] of `FaDPa+`, and the generated DPG after that is treated by $\boldsymbol{\alpha}$-Cut to obtain the $\boldsymbol{k}$ partitions. F2, F3, F4, and F5 work similarly with two, three, four and five rounds of `FaDPa+` followed by $\boldsymbol{\alpha}$-Cut. One alternative to $\boldsymbol{\alpha}$-Cut in the proposed `FaDSPa` is to replace $\boldsymbol{\alpha}$-Cut by normalized cut and keep the remaining method same. We denote these schemes by N$\langle \boldsymbol{number} \rangle$. This stands for the method when `FaDSPa` is applied by repeatedly running `FaDPa+` $\langle \boldsymbol{number} \rangle$ number of times before passing the control to normalized cut. Thus N1 applies one *round* of `FaDPa+`, and the generated DPG after that is treated by normalized cut to obtain the $\boldsymbol{k}$ partitions.

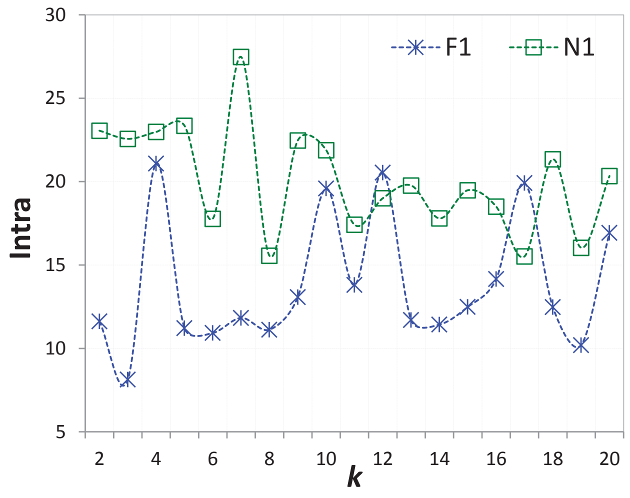
Quality comparison of F1 and N1: Figure 4.5 shows the quality of partitioning obtained by the F1 and N1 schemes of `FaDSPa`. It presents a clear comparison of $\boldsymbol{\alpha}$-Cut and normalized cut when they are embedded in `FaDSPa`. Figures 4.5(a), 4.5(b), and 4.5(c) show the quality in terms of $\boldsymbol{inter}$, $\boldsymbol{intra}$, and ANS respectively (shown in Y-axis) for the number of partitions $\boldsymbol{k}$ varying from 2 to 20 (shown in X-axis) using two curves. The overall partitioning quality is shown in terms of ANS in Figure 4.5(c). We observe that at all the values of $\boldsymbol{k}$, F1 is lower (better in quality) than N1. As this measure considers both the inter-partition and intra-partition distances, it very clearly shows that our proposed `FaDSPa` algorithm (using $\boldsymbol{\alpha}$-Cut) outperforms the other method.

We also look into the inter-partition heterogeneity and intra-partition homogeneity individually. In Figure 4.5(a), we observe that except at $\boldsymbol{k} = 2$, 3 and 4, the $\boldsymbol{inter}$ measure of F1 is always greater than or equal to that of N1. As a higher $\boldsymbol{inter}$ indicates a better partitioning in terms of inter partition distances, it means that most of the times F1 performs better than N1 in terms of this measure. In Figure 4.5(b), we observe that except at $\boldsymbol{k} = 12$ and 17, the $\boldsymbol{intra}$ measure of F1 is always smaller than that of N1. As a lower $\boldsymbol{intra}$ indicates a better partitioning in terms of intra partition distances, it means that most of the times F1 performs better than N1 in terms of this measure. Thus we see

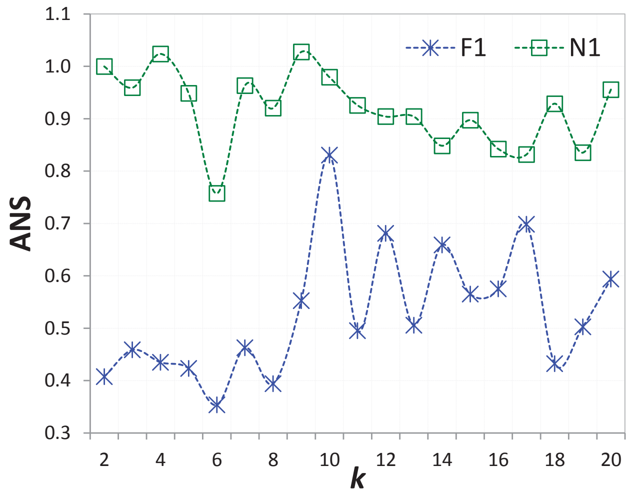
[8]All subsequent usage of this term refer to the schemes F$\langle \boldsymbol{number} \rangle$



(a) Inter



(b) Intra



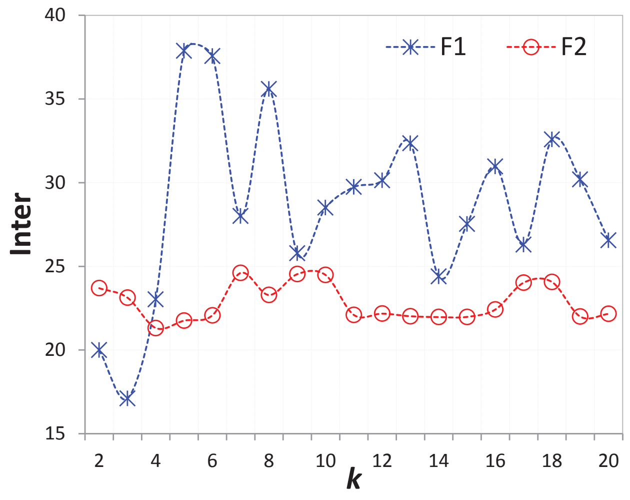
(c) ANS

FIGURE 4.5: Comparison of proposed FaDSPa and normalized cut based FaDSPa on real data

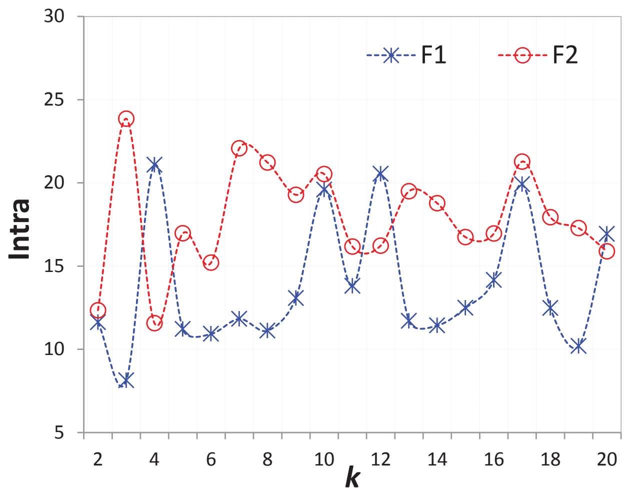
that sometimes F1 performs inferior to N1 in terms of either ***inter*** or ***intra*** individually, but as the overall partition quality considers both the factors simultaneously, F1 always outperforms N1.

Quality comparison of F1 and F2: After we are known about the good performance of our proposed `FaDSPa`, it is important to look into the variation in quality as we increase the number of *rounds*. Figure 4.6 shows the quality of partitioning obtained by the F1 and F2 schemes of `FaDSPa`. Both of them use $\boldsymbol{\alpha}$-Cut. It presents a comparison of one and two rounds of the density-based `FaDPa+` (both followed by $\boldsymbol{\alpha}$-Cut). The overall partitioning quality is shown in terms of ANS in Figure 4.6(c). We observe that at all the values of $\boldsymbol{k}$ except 10, F1 is lower (better in quality) than F2. It shows that generally fewer rounds of `FaDPa+` followed by $\boldsymbol{\alpha}$-Cut produces results in better quality. Also in terms of inter-partition and intra-partition distances individually, we found that most of the times F1 performs better than F2. In Figure 4.6(a), we observe that except at $\boldsymbol{k} = 2$ and 3, the ***inter*** measure of F1 is always greater than that of N1. It means that most of the times F1 performs better than F2 in terms of inter-partition distances. In Figure 4.6(b), we observe that except at $\boldsymbol{k} = 4$, 12 and 20, the ***intra*** measure of F1 is always smaller than that of F2. It means that most of the times F1 performs better than F2 in terms of intra-partition distances.

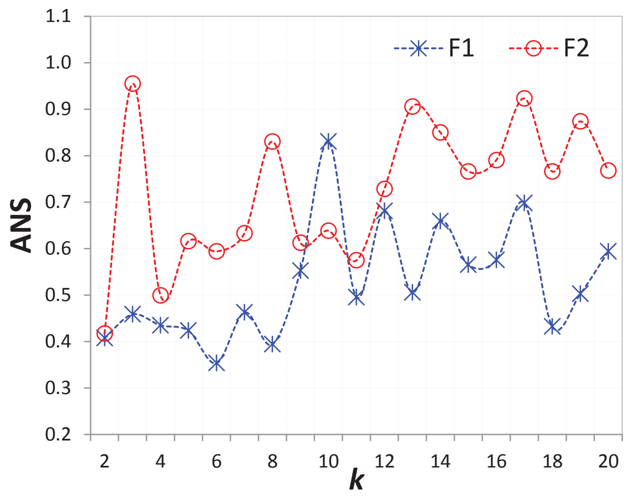
Summary of comparisons: Figure 4.7 presents the overall results summary obtained on the real $\mathtt{M}_{\boldsymbol{s}}$ dataset. It shows the final comparison of the F1, F2, and N1 schemes (in the X-axis) in terms of ANS (in the Y-sxis). As mentioned in Section 4.5.3, the ANS measure also gives the information to identify the optimal number of partitions by selecting the $\boldsymbol{k}$ where the its minimum is found. We see in Figures 4.5(c) and 4.6(c) that the minima of F1, F2 and N1 occur at $\boldsymbol{k} = 6$, 2, and 6 respectively. These values of $\boldsymbol{k}$ become the optimal number of partitions for the respective schemes. We compare the quality of partitioning in terms of ANS obtained at these optimal values of $\boldsymbol{k}$ in the figure. The lowest value of F1 shows itself as the best performer, followed by F2 and N1. Looking into the running times of F1 and F2, we found that they take 173.56 and 17.41 seconds respectively to complete the execution. Thus F1 produces better results than F2 in terms of effectiveness, but takes longer execution time, thus sacrificing the efficiency. The running time of `FaDSPa` on all the datasets is discussed in detail in Section 4.5.5.



(a) Inter



(b) Intra



(c) ANS

FIGURE 4.6: Partitioning results of `FaDSPa` on real data

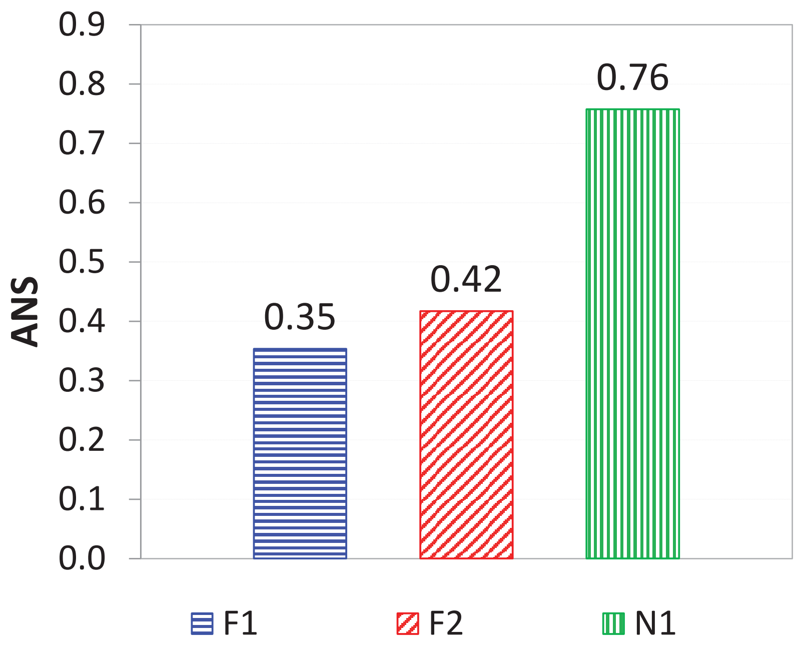
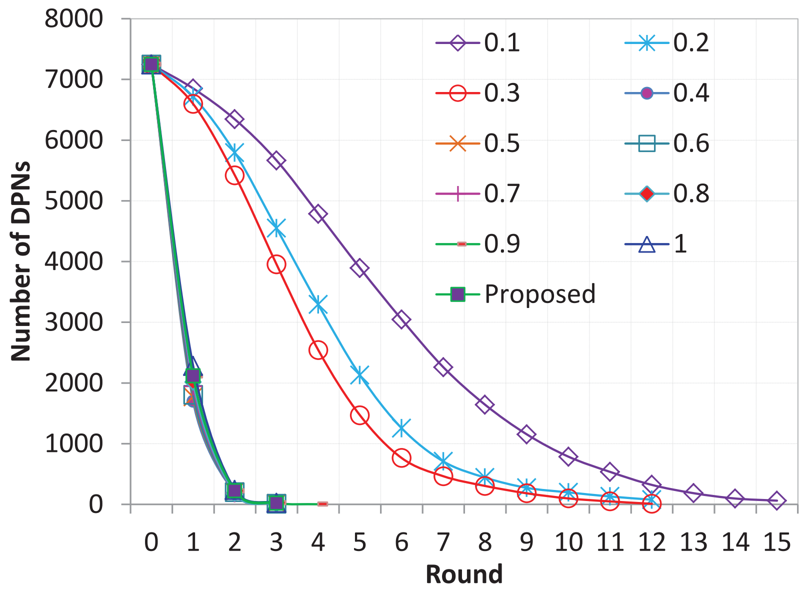
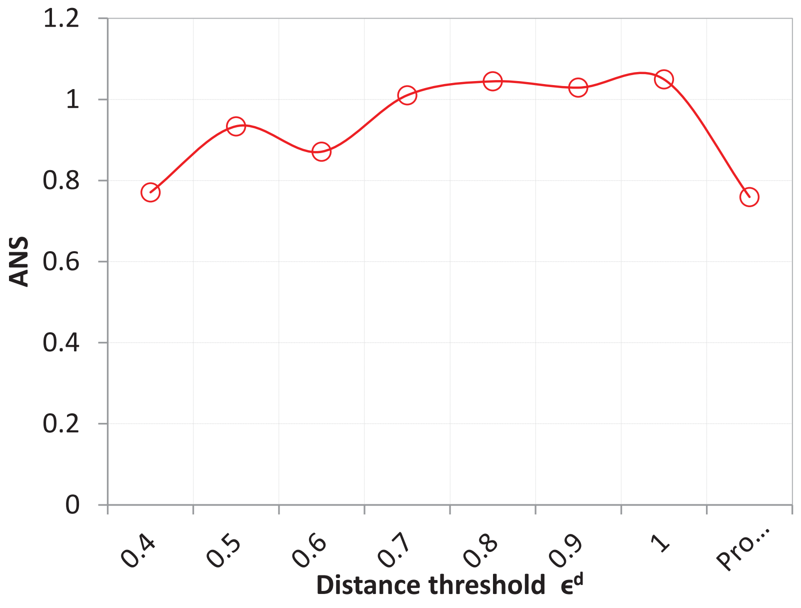


Figure 4.7: Partitioning results summary on SCATS data

Impact of ϵ^d in `FaDPa+`: The distance threshold ϵ^d is a fundamental parameter of `FaDPa+` and has an impact on the quality of partitioning. As mentioned in Section 4.3.3, we consider ϵ^d as a vector of values, where each of those values individually refer to a specific node in the graph. It means that the distance threshold for each node is customized according to the kind of links of the referred node, which helps in clustering the nodes relatively. As it is an external parameter, it can also be set to some fixed value ($\in [\mathbf{0}, \mathbf{1}]$) that is same for all the nodes. When it is set to low values (less than 0.4), due to the strict condition the number of established *density parent-child* relationships is found to be low. As shown in Figure 4.8(a), the DPG is compressed at a slow rate in each round of `FaDPa+` for $\epsilon^d = \mathbf{0.1}, \mathbf{0.2}, \& \ \mathbf{0.3}$, and therefore requires more *rounds* to produce the final partitions. During this gradual compression, most of the nodes accumulate as children of one (or very few) density peak. It results into an imbalanced set constituting one (or very few) big partition and several small partitions, which is not good. For example, setting $\epsilon^d = \mathbf{0.1}$ results into one big partition of 7179 nodes and the remaining 66 nodes are distributed into 61 other partitions. The results gradually improve as this threshold is increased, and become satisfactory only after 0.4. Therefore, ignoring $\epsilon^d = \mathbf{0.1}, \mathbf{0.2}, \& \ \mathbf{0.3}$, Figure 4.8(b) shows the quality of partitions obtained by varying ϵ^d from 0.4 to 1, and compares it with the proposed method of determining its value. Observe that the ANS measure for the proposed method is the lowest of all, and thus leads to the best quality results.



(a) DPG compression



(b) Quality of Partitions

FIGURE 4.8: Impact of the distance threshold ϵ^d

Visualization of obtained partitions: Figure 4.9 presents a snapshot of the different partitions obtained by F1 setting $\boldsymbol{k = 6}$ (optimal number of partitions) at 08:00 AM on

03-12-2012 (Monday). The road segments in each different color represent a partition. The displayed road network includes only those road segments that are operated by traffic signals and have the traffic sensors installed. Therefore even though the area is large, the number of road segments is relatively small. In the Section 4.5.5, we consider all the road segments that exist on digital maps, which results into large number of road segments in small areas.

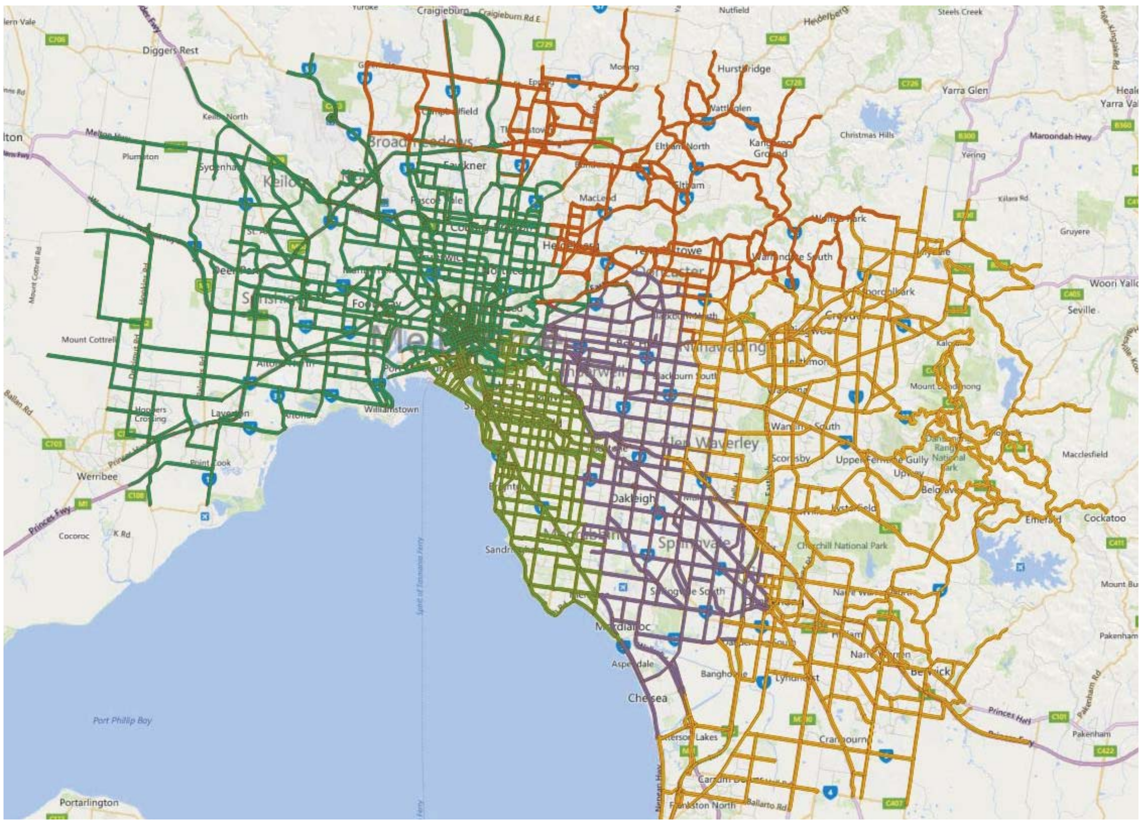


FIGURE 4.9: Partitions obtained from the Melbourne network at 08:00 AM on 03-12-2012 (Monday)

4.5.5 Experimental Results on Large Networks

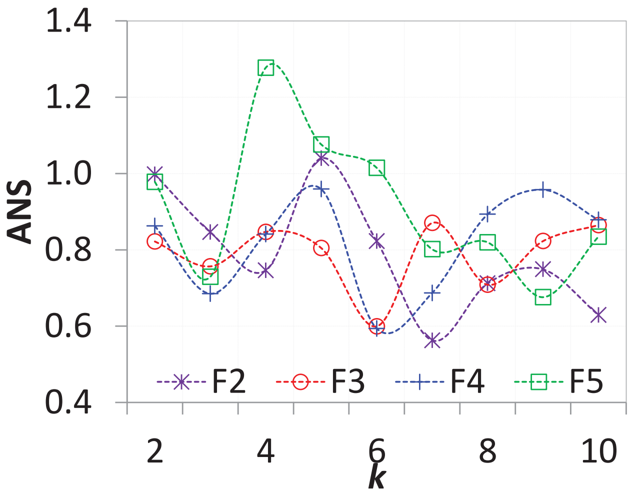
Through our experiments on large urban road networks, we show the performance of `FaDPa+` and `FaDSPa` at different values of the thresholds ϵ_p and ϵ_c respectively. As `FaDSPa` provides the flexibility to handle the complexity of large networks, we also show the trade-off between efficiency and accuracy by varying the external parameter ϵ_c.

Impact of ϵ_p in FaDPa+: Table 4.4 shows the results obtained using FaDPa+ on the M_1 and M_2 datasets. In this algorithm the number of desired partitions is controlled by ϵ_p, shown in the left column, and the actual number of partitions denoted by k could be any value less than or equal to that number. The results in terms of *intra*, *inter*, and ANS are shown for $\epsilon_p = 25$, 50, 75, and 100. The optimal ϵ_p, optimal k, and optimal partitioning are determined by looking into the minimum ANS. The minimum values of 0.75 in M_1 and 0.78 in M_2 at $\epsilon_p = \mathbf{75}$ ($k = 69$ and 63 respectively) indicate that 75 is the optimal ϵ_p for both the datasets. We also observe that the same value of ϵ_p leads to different values of k in different datasets.

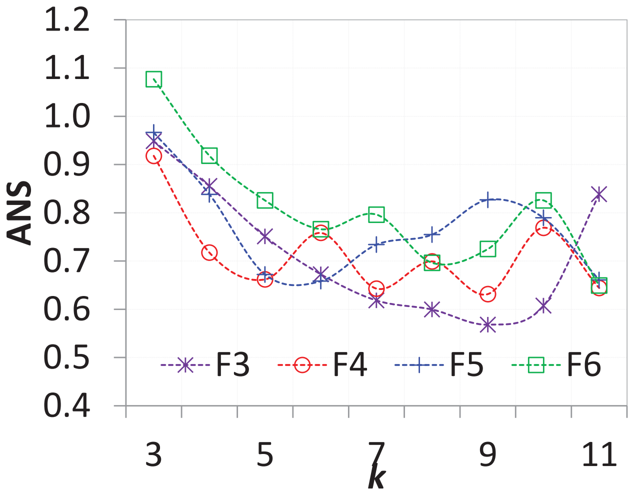
TABLE 4.4: FaDPa+: quality of partitioning

	M_1				M_2			
ϵ_p	k	**Intra**	**Inter**	**ANS**	k	**Intra**	**Inter**	**ANS**
25	23	0.4554	1.2805	1.1375	24	0.1691	0.2904	1.1857
50	47	0.4384	1.2886	0.8071	46	0.2274	0.3735	0.8632
75	69	0.3878	1.2934	**0.7463**	63	0.2947	0.5810	**0.7826**
100	93	0.4849	1.1628	0.8322	89	0.3326	0.6113	0.8146

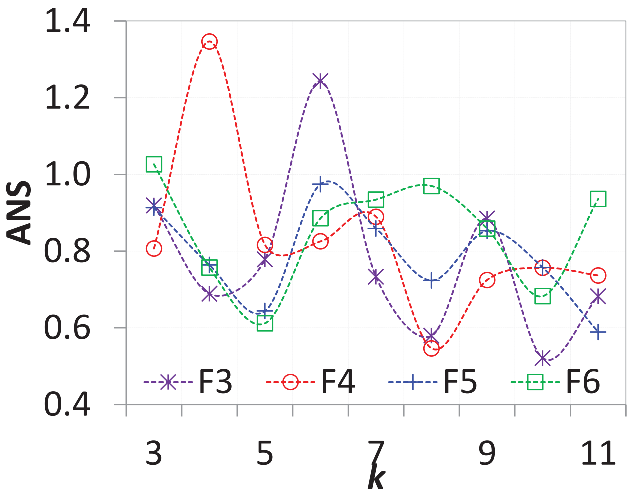
Accuracy in FaDSPa: Figure 4.10 shows the partitioning quality of FaDSPa in large road network datasets M_1, M_2 and M_3. Figure 4.10(a) shows the ANS measures obtained for $k = 2$ to 10 for M_1. There are four curves for F2, F3, F4 and F5. As explained earlier in Section 4.5.4, F$\langle \boldsymbol{number} \rangle$ denotes the method when FaDSPa is applied by repeatedly running FaDPa+ $\langle \boldsymbol{number} \rangle$ number of times before passing the control to α-Cut. Thus F2, F3, F4, and F5 apply two, three, four, and five *rounds* of FaDPa+, and the generated DPG after that is treated by α-Cut to obtain the k partitions. Fewer rounds of FaDPa+ produces a large DPG and puts more work on α-Cut, and vice versa. Thus the figure shows the quality of results obtained by varying the combination of our density and spectral based methods. We observe that there is no such clear trend that for all the values of k, one setting outperforms the other. The reason is that the partitioning quality is also highly dependent on the selected value of k, which could vary for the different schemes. To know their relative performance, we find the most suitable k for each of them. The scheme F2 has its minimum at $k = \mathbf{7}$, which shows 7 as its most suitable k. Similarly F3, F4, and F5, have their most suitable k as 6, 6 and 9 respectively. Their ANS values at the minimum are 0.56, 0.60, 0.59 and 0.68 respectively. Comparing the depth of their minimum we



(a) ANS in $\mathtt{M_1}$



(b) ANS in $\mathtt{M_2}$



(c) ANS in $\mathtt{M_3}$

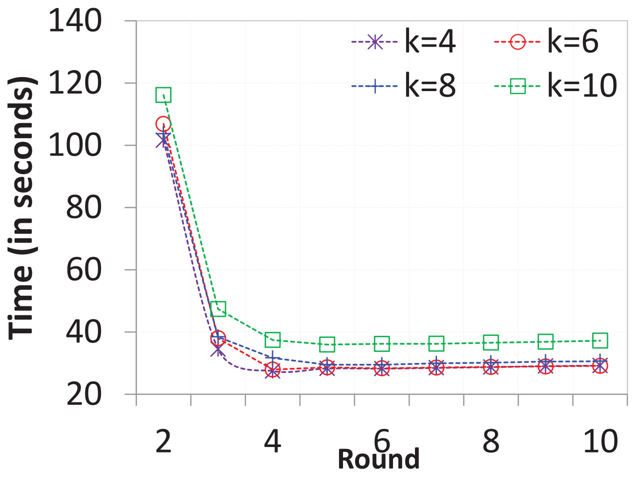
FIGURE 4.10: Partitioning results in large networks

observe that the order of their performance is $\boldsymbol{F2} > \boldsymbol{F3} \simeq \boldsymbol{F4} > \boldsymbol{F5}$. Figures 4.10(b) and 4.10(c) show the ANS measures for the larger datasets $\mathtt{M_2}$ and $\mathtt{M_3}$ starting[9] from F3 to F6. A similar performance trend is found also for these larger datasets, which are $\boldsymbol{F3} > \boldsymbol{F4} > \boldsymbol{F5} \simeq \boldsymbol{F6}$ and $\boldsymbol{F3} > \boldsymbol{F4} > \boldsymbol{F5} > \boldsymbol{F6}$ respectively. We observe that the partitioning quality is generally better with fewer rounds of `FaDPa+` (or lower values of $\langle \boldsymbol{number} \rangle$) and doing more of the task by $\boldsymbol{\alpha}$-Cut. The reason behind this is that a larger value of $\langle \boldsymbol{number} \rangle$ compromise more with the partitioning accuracy, and improve the efficiency[10]. We saw in previous experiments that $\boldsymbol{\alpha}$-Cut performs better than the density-based `FaDPa+`, because of its global perspective in partitioning. When `FaDPa+` is repeatedly applied for F$\langle \boldsymbol{number} \rangle$, it starts with forming small partitions locally which combine with others in the subsequent rounds and form larger partitions in each repetition. In this way, they form hierarchical partitions locally. Once a grouping (in the form of a partition) is formed based on the local density-based information during these steps, it is done permanently for the final result. These small partitions are not broken or re-adjusted later, which leads to increase in lose of accuracy in each round. After $\langle \boldsymbol{number} \rangle$ rounds, $\boldsymbol{\alpha}$-Cut (that looks into the graph globally) is applied on the DPG constructed from the obtained intermediate partitions to get the final partitions.

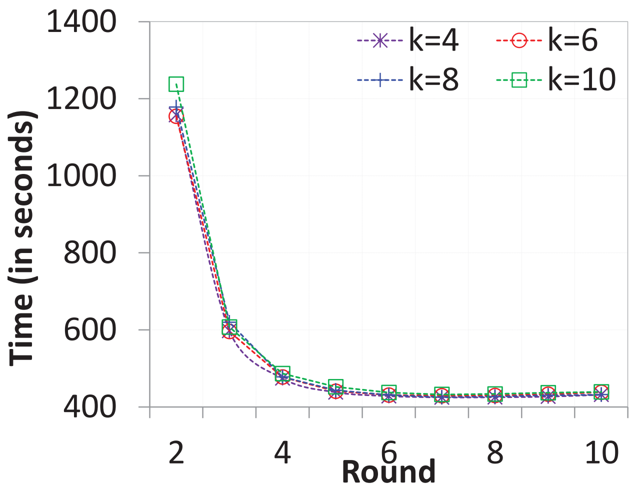
Efficiency in `FaDSPa`: Figure 4.11 shows the execution time (in seconds) when $\boldsymbol{k}$ is set to 4, 6, 8, and 10, for all the three large datasets. In the X-axis we vary the number of rounds of `FaDPa+` from 2 to 10 (F2 to F10). We observe that for any value of $\boldsymbol{k}$, as the rounds increase the execution time decreases, while at the same time the accuracy degrades, and vice versa. Another thing to note is that, the execution time decreases drastically in the initial rounds (decreases from more than 100 secs to around 40 secs from F2 to F3 for $\mathtt{M_1}$) and this amount of decrease reduces in the subsequent rounds (decreases from around 40 secs to around 30 secs from F3 to F4). Thus the efficiency increases drastically as we progress through the initial rounds but becomes almost stable later, whereas the accuracy decreases as the rounds increase and no such drastic decrease is seen. It suggests that if the efficiency of execution is also a concern in addition to accuracy, then it is worth sacrificing the accuracy in the first few rounds. Thus a good choice of rounds for $\mathtt{M_1}$ could be any one

[9]The reason we do not start from F2 is that their DPGs produced after two rounds of `FaDPa+` have a large number of DPNs (close to 10,000), and to apply $\boldsymbol{\alpha}$-Cut on such a large graph is beyond the scope of our computing environment because of the high computational complexity.

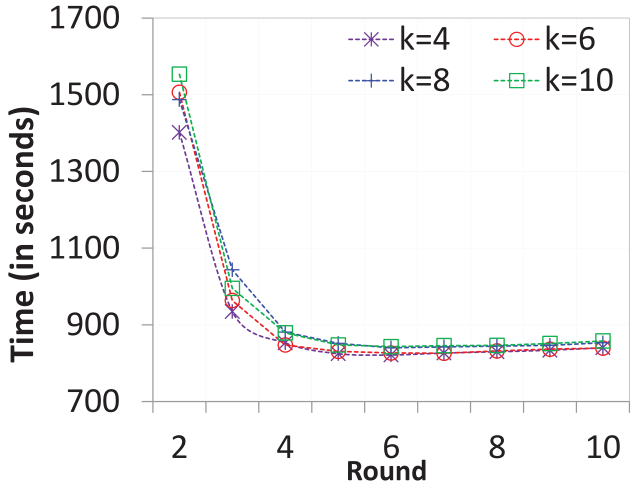
[10]Shown in Figure 4.11 and explained in the next paragraph.



(a) Running time in M₁



(b) Running time in M₂



(c) Running time in M₃

FIGURE 4.11: Running time in large networks

of F3, F4, and F5, as they can do the work efficiently with a satisfactory accuracy. The elbow point can be used as an indicator to locate the best choice. In the higher rounds, the execution time goes a little higher, though it is not very significant. This behavior and the reason behind it are explained later in detail. In the figures we see that the curves for the different values of $\boldsymbol{k}$ almost overlap. It shows that though a larger $\boldsymbol{k}$ requires a longer execution time, the difference is small.

Table 4.5: Running Time (in seconds)

Method	**DPN-$\mathtt{D_1}$**	**Time-$\mathtt{D_1}$**	**DPN-$\mathtt{M_s}$**	**Time-$\mathtt{M_s}$**	**DPN-$\mathtt{M_1}$**	**Time-$\mathtt{M_1}$**	**DPN-$\mathtt{M_2}$**	**Time-$\mathtt{M_2}$**	**DPN-$\mathtt{M_3}$**	**Time-$\mathtt{M_3}$**
FaDPa+	NA	less than 1	NA	16.01	NA	37.58	NA	538.16	NA	887.196
F0 ($\boldsymbol{\alpha}$-Cut)	420	less than 1	7245	10507.20	17206	-	53494	-	79487	-
F1	109	less than 1	2124	173.56	7402	10723.70	22670	-	30543	-
F2	13	less than 1	224	17.41	2990	116.15	9398	25018.30	9967	28477.60
F3	1	NA	17	15.77	1419	47.39	4715	1237.63	4495	1553.46
F4	NA	NA	NA	NA	814	37.39	2874	607.31	2654	995.10
F5	NA	NA	NA	NA	512	**35.96**	1938	486.61	1785	879.18
F6	NA	NA	NA	NA	355	36.18	1439	452.09	1327	847.59
F7	NA	NA	NA	NA	257	36.20	1125	437.43	1047	**843.25**
F8	NA	NA	NA	NA	190	36.55	906	**432.22**	849	845.61
F9	NA	NA	NA	NA	145	36.83	752	433.77	698	846.46
F10	NA	NA	NA	NA	115	37.19	626	436.90	591	851.39

Efficiency analysis in detail: Table 4.5 shows the running time of `FaDPa+` and `FaDSPa` at varying $\boldsymbol{\epsilon_c}$ (by varying the number of *rounds*) on all the datasets. The left most column stands for the method, which is either `FaDPa+` or F$\langle \boldsymbol{number} \rangle$, where F0 refers to the method of applying $\boldsymbol{\alpha}$-Cut directly on the road graph. The table has other columns for the number of DPNs obtained after $\langle \boldsymbol{number} \rangle$ *rounds* and the execution time in seconds for each dataset. For the small dataset $\mathtt{D_1}$ that has 420 nodes, the methods `F3` and higher are not applicable, as in the third round it `FaDPa+` merges all the partitions into a single large partition. It completes running within fractions of a second. Similar to $\mathtt{D_1}$, `F4` and higher are not applicable to the real dataset $\mathtt{M_s}$. In the large datasets, the running time decreases as $\boldsymbol{\epsilon_c}$ becomes lower (with increasing *rounds*). The reason for this decrease is that $\boldsymbol{\epsilon_c}$ decides how much the road graph is to be compressed using `FaDPa+` to form the DPG, and thus the size of the matrix that has to be eigen-decomposed for applying $\boldsymbol{\alpha}$-Cut. As eigen-decomposition is a computationally expensive task, the running time increases substantially as the size of this matrix goes beyond a limit, where the limit depends on the computing environment. We see that there is a large difference between the time taken by `F0` and `F1` in $\mathtt{M_s}$. It jumps to 10507.20 seconds in `F0` from 173.56 seconds in `F1`. Similarly there is a sudden rise of execution time from `F2` to `F1` in $\mathtt{M_1}$, and `F3` to `F2` in $\mathtt{M_2}$ and $\mathtt{M_3}$. On applying α-Cut directly (F0) on $\mathtt{M_1}$, it could not complete execution even in 10 hours, and its application on $\mathtt{M_2}$ and $\mathtt{M_3}$ could not be performed due to higher memory requirement. The reason for such behavior is the expensive eigen-decomposition task. In the initial rounds of `FaDPa+`, the number of DPNs reduce rapidly whereas in the successive rounds they reduce at a slower rate. In $\mathtt{M_1}$ the number of DPNs at `F1`, `F2`, `F3`, and `F4` are 7402, 2990, 1419, and 814, respectively. The difference between the first two cases is more than the difference between the last two. The same trend is also observed with $\mathtt{M_2}$ and $\mathtt{M_3}$.

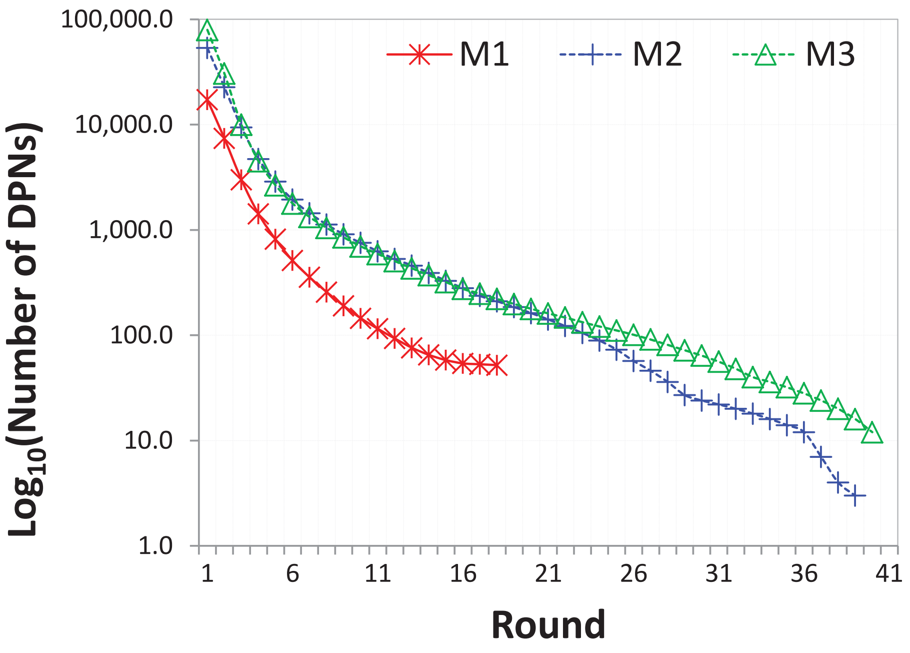


FIGURE 4.12: Compression of the DPG at different rounds

While normally it is expected that `FaDPa+` would have the lowest running time because no eigen-decomposition task is needed, we actually see that the minimum running time is taken by `F5` in case of $\mathtt{M_1}$. It reduces as the method goes close to `F5` and then increases gradually as it goes further. Similarly, the minimum execution time for $\mathtt{M_2}$ and $\mathtt{M_3}$ occur at `F8`, and `F7` respectively. It shows that for large datasets, purely density based clustering methods take more time than when it is combined with spectral clustering as in `FaDSPa`. The reason behind this phenomenon can be explained from Figure 4.12. For all three datasets, it shows the compression rate of the DPG by using `FaDPa+`. The vertical axis is the logarithm of the number of DPNs and the horizontal axis is the number of rounds of `FaDPa+`. We observe that at lower rounds, the number of DPNs reduces rapidly, and at later rounds, the DPNs remain almost constant. Thus achieving further compression requires many more iterations in the higher rounds, which consumes much longer time for `FaDPa+`. But if $\boldsymbol{\alpha}$-Cut is applied at that point instead of further compression using `FaDPa+`, the task can be done in a single iteration, thus requiring much less execution time.

4.6 Summary

In this chapter, we presented a framework called `FaDSPa` for spatial partitioning of large urban road networks, which employs both density and spectral based clustering. It is based on the data of traffic congestion on a road network defined by the vehicle density per unit distance on each road segment. It starts by transforming the actual road network into a *road graph*, followed by mining a *density peak graph* using our density based clustering algorithm called `FaDPa`. Thereafter we apply our spectral clustering based $\boldsymbol{\alpha}$-Cut on this graph to obtain the different *road network partitions*. The framework makes use of the locally distributed computations of `FaDPa` and the globally centralized computations of $\boldsymbol{\alpha}$-Cut together to make it efficient as well as effective. Our experiments on real SCATS data shows that it is very much applicable in real environments. Our method outperforms the existing road network partitioning method based on normalized cut. We found that in small networks our $\boldsymbol{\alpha}$-Cut performs the best whereas `FaDSPa` produces satisfactory results, but in large networks direct application of $\boldsymbol{\alpha}$-Cut brings huge time and space complexities that may go beyond the scope of the computing environment. `FaDSPa` handles such large networks with the help of a density peak graph, also providing the flexibility of setting the trade-off between efficiency and accuracy. We found that density based computations are faster, whereas the spectral based computations give better results in terms of quality.

Chapter 5

Tracking and Capturing the Spatio-temporal Evolution of Congestion

Nowadays the urban road networks undergo frequent traffic congestions, specially during the peak hours and around the city centre areas. Capturing the spatiotemporal evolution of the congestion scenario in real-time can aid in developing smart traffic management systems, and guiding commuters in understanding the real-time traffic. Some previous works have developed methods for traffic based spatial partitioning of road networks at a particular point of time, and represented the congestion scenario using the obtained differently congested partitions. These partitions have homogeneous level of congestion inside, but heterogeneous to others. As the traffic is dynamic and changes continuously in real-time, the partitions would evolve with time in terms of their structure and location. Partitioning the network at each point of time is a computationally expensive task. In this chapter, we propose a comprehensive framework to capture the evolution by incrementally updating the partitions in an efficient manner using a two-layer approach. The physical layer maintains a set of small-sized road network building blocks, and performs low-level computations to incrementally update them, whereas the logical layer performs high-level computations in order to serve as an interface to query the physical layer about the congested partitions. At each time point, the least stable road segments are identified based on a stability measure, and their processing is prioritized using a heap tree. We draw the concept of road network motifs and use the short cycles to heuristically identify the most suitable building block for an unstable road segment. We also propose an in-memory index called `Bin` that compactly stores the historical sets of building blocks with no information loss and facilitates their efficient retrieval. Extensive experiments are performed on real and synthetic datasets. Our results show that the proposed method is much efficient than the existing re-partitioning methods without significant sacrifice in accuracy. The proposed `Bin` consume a minimum space with least redundancy at different time points. The proposed framework can be used for a real-time continuous tracking and capturing of the evolution of congestion hotspots in an urban road network.

5.1 Introduction

All major cities these days are affected by the problems of frequent traffic congestions. It is very important to maintain suitable traffic management policies and strategies according to the nature of congestion occurring in the region. Traffic congestions generally occur frequently during the peak hours and around the city center areas. The analysis of the spatiotemporal evolution of the traffic leading to congestions is a problem of increasing importance, in order to understand and optimize the traffic flow.

Generally the roads of different localities or suburbs experience specific traffic flow patterns based on their *spatiotemporal* significance. In the spatial perspective, roads inside the city center or an area having popular venues like a stadium or hospital, usually remain more congested than others without such significance. It leads to the fact that the different small sub-networks (of small areas like suburbs) of a large urban road network experience distinctive traffic flow within them. Previous works have applied the partitioning of road networks based on their traffic level to identify such sub-networks called *spatial partitions* or simply *partitions* [3, 4]. The set of partitions obtained in this way include that of both high and low levels of congestion at a particular point of time. On the other hand, in the temporal perspective, the roads usually remain busier with higher congestion levels during the peak times than the off-peak times. It reflects the dynamic nature of congestion in the spatial partitions. These spatiotemporal behaviors altogether lead to the evolution of traffic congestion on urban road networks. For example, during the morning office-opening hours, the congestion generally starts developing in the outer suburbs and the roads connecting them to the city center, mostly occur inside the city during the day, and starts moving outwards again during the office-closing hours. The congestion can be simply understood and represented as a congested partition that keeps on changing its structure, location, and level of congestion. The non-congested partitions and their evolution further strengthens our understanding of the overall traffic dynamics. Thus the continuous maintenance, tracking, and capturing of the differently congested partitions in an incremental approach can potentially aid traffic management systems [61, 91].

The `Google Traffic` feature of `Google Maps`[1] visualizes real-time road traffic, based on anonymously collected data. It is simply a heatmap of the actual traffic on the corresponding roads. Figure 5.1 shows a snapshot of a typical Monday 11:00 AM visualization. Using this visual information, the commuters can plan their journey, including route and travel-time selections, instead of unexpectedly getting stuck into a congestion. As the congestion forms and dissolves via the linked road segments, the level of congestion on a road segment is dependent on the preceding and following segments. Instead of naively considering the congestion level of individual road segments, they can be effectively grouped in the form of differently congested partitions (including both congested and non-congested ones), and visualized to show the real-time congestion scenario. It would further enrich the visual information about the congestion spread and connectivity of the different individual congestions [6, 7]. Some other applications of the maintenance of road network partitions are partition-based route-guidance, trip planning, recovery of missing traffic data, and other complex graph processing methods for traffic-aware smart travel services [3, 4, 85].

The naive way to track the change in partitions is to perform spatial partitioning of the road network at each time point based on the corresponding traffic measures, and analyze the change. But a complete re-partitioning is a computationally expensive task and may even require more time than the time-interval of data collection. Generally in successive time points, the traffic does not change abruptly, rather it is a gradual process. A logically better and efficient way is to incrementally update the previously obtained set of partitions by processing only the sections of probable change. It significantly reduces the

[1]`https://maps.google.com.au`

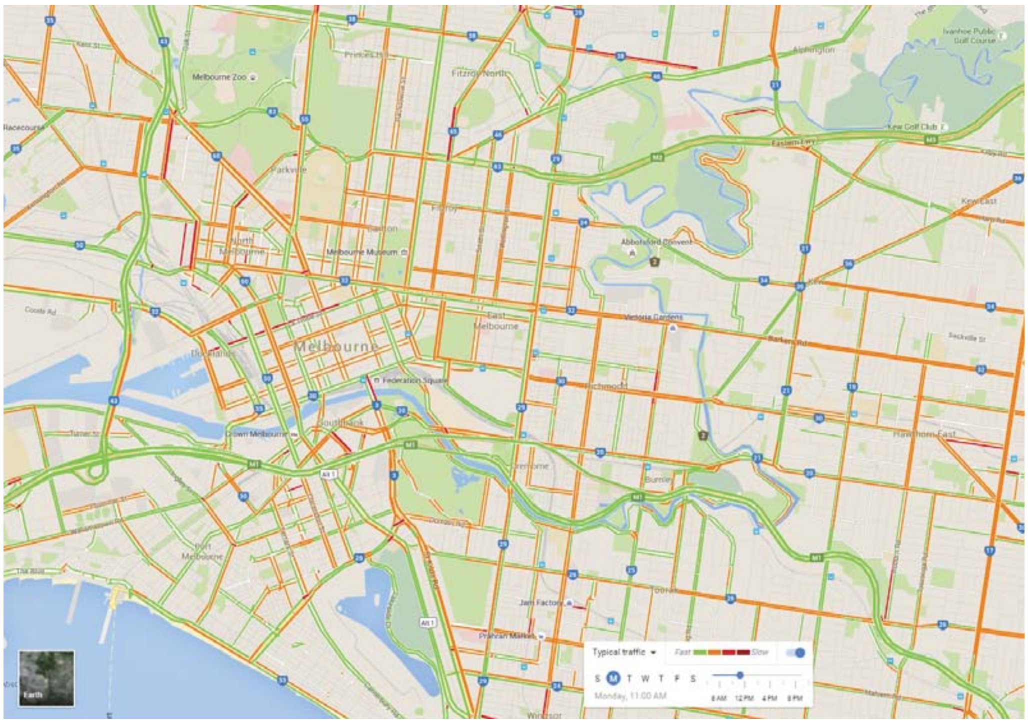


FIGURE 5.1: `Google Traffic` visualization at 11:00 AM (typical, Monday)

computations, while may sacrifice the quality of partitions marginally. There exist works on the complete partitioning of road networks [3, 4, 61], but the problem of incremental maintenance of their partitions has still remained unexplored. The main challenges in this problem are two folds. First, the computations need to be efficient enough to complete the incremental update before the arrival of data from the next time point. Second, there needs to be a mechanism to economically store the historical information in primary or secondary memory, which provides its efficient retrieval.

We extend this work in the current chapter, and develop a comprehensive framework to capture the spatiotemporal evolution of the traffic scenario. We models the building blocks in the form of a congestion evolution graph and propose an in-memory index called `Bin` to compactly store the historical sets of building blocks.

In this chapter, we present a comprehensive framework to capture the spatiotemporal evolution of the traffic congestion scenario by incrementally updating the road network partitions. This framework works in a two-layer approach, consisting of a *physical* layer and a *logical* layer. The physical layer performs low-level computations for the incremental update, whereas the logical layer presents those obtained results to the user after a light makeover. Our method in the physical layer starts with a set of *building blocks* (defined in Section 5.4.1) of the road network at the beginning time point. During the period of evolution, the unstable road segments are identified at each time point, indexed as a heap tree, and moved to their most suitable building blocks. We models the building blocks in the form of a congestion evolution graph and propose an in-memory index called `Bin` to compactly store the historical sets of building blocks. `Bin` is referenced and updated by the physical layer during the incremental updates. The logical layer accesses `Bin` to efficiently retrieve and present the congestion evolution graph, and support specific queries related to congested partitions. In summary, we make the following main contributions.

– We model the road network congestion in the form of a *congestion evolution graph* using a set of *building blocks*, to effectively capture its spatiotemporal evolution.

– We develop an in-memory index for the building blocks to compactly store the historical information in the main memory. It facilitates efficient retrieval, visualization, and understanding of the evolution using the *congestion evolution graph.*

– We adapt the two-layer method for incremental maintenance of the differently congested partitions (proposed in our previous preliminary work [5]) with respect to `Bin`, and develop a comprehensive framework. The method incrementally updates the *building blocks* at each new time point in `Bin`. The *stability measure*, used to identify the unstable road segments, and the concepts of *road network motifs* used to understand the grouping patterns, are the two main highlights, which set the foundation.

– We perform extensive experiments on both real and synthetic data to demonstrate the effectiveness of our method.

The rest of the chapter is organized as follows. We start with some preliminaries and problem definition in Section 5.2. Section 5.3 describes the proposed method in a high level, which is followed by the dynamics of road networks in Section 5.4, and capturing the spatiotemporal evolution in Section 5.5. The experimental results are presented in Section 5.6, followed by the chapter summary in Section 5.7.

5.2 Preliminaries

In this section, we firstly introduce some fundamental concepts related to road networks, and then define the problem.

5.2.1 Road Networks

Urban roads exist in the form of a physical network, defined in Definition 5.1, spatially spread over a large urban area. We give it a graphical representation[2] in Definition 5.2.

Definition 5.1. (**Road Network**) An urban road network is defined as $\mathcal{N} = (\mathcal{I}, \mathcal{R})$ comprising a set of intersection points $\mathcal{I} = \{\iota_1, \iota_2, \ldots, \iota_{n_\iota}\}$ as nodes that are connected among themselves by directed road segments $\mathcal{R} = \{r_1, r_2, \ldots, r_{n_r}\}$ as links, where each road segment r_i associates a measure of traffic density $r_i.d$ with itself. ■

Definition 5.2. (**Road Graph**) Given a road network $\mathcal{N}$, the corresponding road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed by representing each road segment $r_i \in \mathcal{N}$ as a node v_i, and establishing an undirected link e_i between each possible node pair (v_j, v_k) if there exists at least one intersection point ι_l which is a common intersection for the roads r_j and r_k, and the traffic can flow either from r_j to r_k or vice versa. Each node v_i $(node(r_i)) \in \mathcal{V}$ associates with it a feature value $v_i.f$, which is the road traffic density $r_i.d$. ■

Definition 5.3. (**Partition**) A given road network $\mathcal{N}$ can be partitioned into multiple segments, each of whom is called a partition $\mathcal{P}_i$ of $\mathcal{N}$. All the different segments form a set of partitions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, such that *i*) $\bigcup_{i=1}^{k} \mathcal{P}_i = \mathcal{R}$ and $\mathcal{P}_i \bigcap \mathcal{P}_j = \emptyset$ for all $i \neq j$, and *ii*) each $\mathcal{P}_i$ is connected inside and all adjacency relations, except the cross-partition relations (inter-partition links), are maintained as in $\mathcal{N}$. ■

A partition of a road network can be transformed to that of a road graph by following Definition 5.2, and vice versa. Most of the urban roads exist as two-way roads, which are divided into two parts from the middle for traffic of the two opposite directions. These two parts undergo different kinds of traffic flow patterns. For example, on a road that connects outskirts with the city center, the morning office hours would find more traffic

[2]Further details of this representation can be found in [4]

heading towards the city center, whereas the evening hours would find more traffic in the opposite direction. This feature is accommodated by considering the two directions as separate road segments that share common intersection points, and thus are adjacent.

Definition 5.4. (**Inter-Partition Associativity**) For a given set of partitions $\mathcal{P}$, the inter-partition associativity is defined as the aggregation of the level of congestion similarity between all possible pairs (r_i, r_j) for which r_i and r_j lie in different partitions. ■

Definition 5.5. (**Intra-Partition Associativity**) For a given set of partitions $\mathcal{P}$, intra-partition associativity is defined as the aggregation of the level of congestion similarity between all possible linked pairs (r_i, r_j) for which r_i and r_j lie in the same partition. ■

5.2.2 Problem Definition

Given an urban road network $\mathcal{N} = (\mathcal{I}, \mathcal{R})$ and its road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the problem addressed in this chapter is to incrementally update the road network partitions with the aim to capture the evolution of traffic congestion. Let us suppose, we are given a set of road network partitions $\mathcal{P}^{i-1} = \left\{\mathcal{P}_1^{i-1}, \mathcal{P}_2^{i-1}, \ldots, \mathcal{P}_k^{i-1}\right\}$ based on the traffic at time t_{i-1}, such that $\mathcal{P}^{i-1}$ has a minimum possible inter-partition associativity and a maximum possible intra-partition associativity.

$i)$ The first objective is to incrementally update $\mathcal{P}^{i-1}$ to $\mathcal{P}^i = \left\{\mathcal{P}_1^i, \mathcal{P}_2^i, \ldots, \mathcal{P}_k^i\right\}$ at each new time point t_i based on the respective traffic data, without re-partitioning the whole network, in such a way that the properties of inter-partition and intra-partition associativities are maintained.

$ii)$ The second objective is to develop an in-memory indexing scheme for a compact storage of the incrementally obtained historical sets of partitions $\mathcal{P}^0, \mathcal{P}^1, \ldots, \mathcal{P}^i$ and facilitate their efficient retrieval.

5.3 Proposed Method

The proposed method tracks the evolution of traffic congestion over a period of time. Instead of incrementally maintaining the partitions directly, we embed the functionalities in two different layers. The *logical layer* gets the query to identify the congested partitions at a time point from the user, passes it to the physical layer, lightly processes the returned data, and returns the results to the user, whereas the *physical layer* efficiently maintains a large number of evolving building blocks using an index structure.

5.3.1 Logical Layer

From the user end, the logical layer provides the service to get the congested/non-congested partitions at any point of time. It is based on a set of so-called building blocks that are maintained up-to-date by the physical layer. After getting a query from the user, this layer transforms the granularity of the query from partitions to the building blocks, and passes to to the physical layer. For example, a query to fetch $\boldsymbol{k}$ *differently congested partitions* of the network at the current time is transformed to fetch the building blocks of the current time. The physical layer returns the result as $\boldsymbol{\mathcal{B}} = \{\boldsymbol{\mathcal{B}_1}, \boldsymbol{\mathcal{B}_2}, \ldots, \boldsymbol{\mathcal{B}_{n_b}}\}$. The logical layer constructs a building block graph $\boldsymbol{\mathcal{G}^b} = \left(\boldsymbol{\mathcal{V}^b}, \boldsymbol{\mathcal{E}^b}\right)$, where each building block forms a node $\varsigma_i \in \boldsymbol{\mathcal{V}^b}$ and all pairs of *neighboring* (defined later) building blocks or nodes $\{\varsigma_i, \varsigma_j\} \in \boldsymbol{\mathcal{V}^b}$ are connected by links $\varepsilon_l \in \boldsymbol{\mathcal{E}^b}$. The number of nodes in this graph is much smaller than that in the road graph ($|\boldsymbol{\mathcal{V}^b}| = n_b << n_r$). The nodes are first assigned their feature values $\varsigma_i.\boldsymbol{f}$ as the mean of the corresponding building block. The links ε_l between nodes ς_i and ς_j are weighted by the similarity between $\varsigma_i.\boldsymbol{f}$ and $\varsigma_j.\boldsymbol{f}$ as in [3, 4]. Then a partitioning is performed on $\boldsymbol{\mathcal{G}^b}$ to obtain a set of $\boldsymbol{k}$ differently congested partitions, utilizing any existing method. For example, $\boldsymbol{\alpha}$-Cut [3, 4] is one such algorithm to do this task. Those with high means are considered as congested, and those with low means are considered as non-congested. Using this information the user query is responded accordingly.

The physical layer always keeps itself up-to-date with the building blocks that are to be served to the logical layer, and the logical layer partitions a small graph of building blocks, which takes fractions of a second. Thus the method is able to produce the results immediately for any query. Below we will focus on the development of the physical layer and the algorithms to maintain the building blocks.

5.3.2 Physical Layer

The traffic congestion has the property to form and gradually grow from small regions to spread into other parts via the linked road segments. It is also very natural to have multiple blocks of independent congestions at the same time, which sometimes even merge with others. Considering both the congested and non-congested blocks of road segments, we propose the concept of *building blocks* in road networks.

Definition 5.6. (**Building Block**) Given a road graph $\mathcal{G}$, a building block $\mathcal{B}_i = (\mathcal{V}_i^b, \mathcal{E}_i^b)$ is defined as a subgraph that forms one of the n_b fundamental constituents $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{n_b}\}$ in the physical layer at a time point, such that $\bigcup_{i=0}^{n_b} \mathcal{V}_i^b = \mathcal{V}$ and $\mathcal{V}_i^b \bigcap \mathcal{V}_j^b = \emptyset$ for all $i \neq j$. ■

The physical layer is the backbone of the proposed tracking method. It continuously maintains the evolving building blocks by incrementally updating them based on the most recent traffic data. To efficiently perform the incremental update, we start with an off-line preprocessing step to mine the road network building blocks. In the illustration examples of this chapter, we will partition the road network based on the historical traffic data using α-Cut [3], and consider the obtained partitions as the building blocks for the starting point. At each new time point, the most recent traffic data is fetched, based on which these blocks are incrementally updated by identifying and processing the *unstable* road segments. The building blocks for all the time points are stored and maintained in an index structure that facilitates their compact storage for later reference and efficient retrieval. For any query being passed from the logical layer, it retrieves the result from the index and returns back instantly.

5.4 Road Network Dynamics

The road networks are structurally static, but the continuously changing traffic induces the dynamics. It is this continuous change that leads to the formation and deformation of congestion.

5.4.1 Congestion Evolution

Definition 5.7. (**Boundary Node**) Given a set of building blocks $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{n_b}\}$ of a road graph $\mathcal{G}$, a node $v_i \in \mathcal{B}_p$ is called a *boundary node* if there exists another node v_j such that $\langle v_i, v_j \rangle \in \mathcal{E}$ and $v_j \notin \mathcal{B}_p$. ■

Definition 5.8. (**Neighbor**) Given a set of building blocks $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{n_b}\}$ of a road graph $\mathcal{G}$, $\mathcal{B}_p$ is called a *neighbor* of $\mathcal{B}_q$ if there exist nodes v_i and v_j such that $\langle v_i, v_j \rangle \in \mathcal{E}$ and $v_i \in \mathcal{B}_p \wedge v_j \in \mathcal{B}_q$. ■

A *quality* building block[3] $\mathcal{B}_i$ is characterized by two properties: high homogeneity (in terms of their feature values) amongst the nodes inside, and a stable structure that tends to change the least with time. We capture the evolution of network congestion by tracking the evolution of blocks in a low level. The blocks for the beginning time point are mined by partitioning the network based on historical traffic data, and the proposed method in this chapter starts tracking from this initial set.

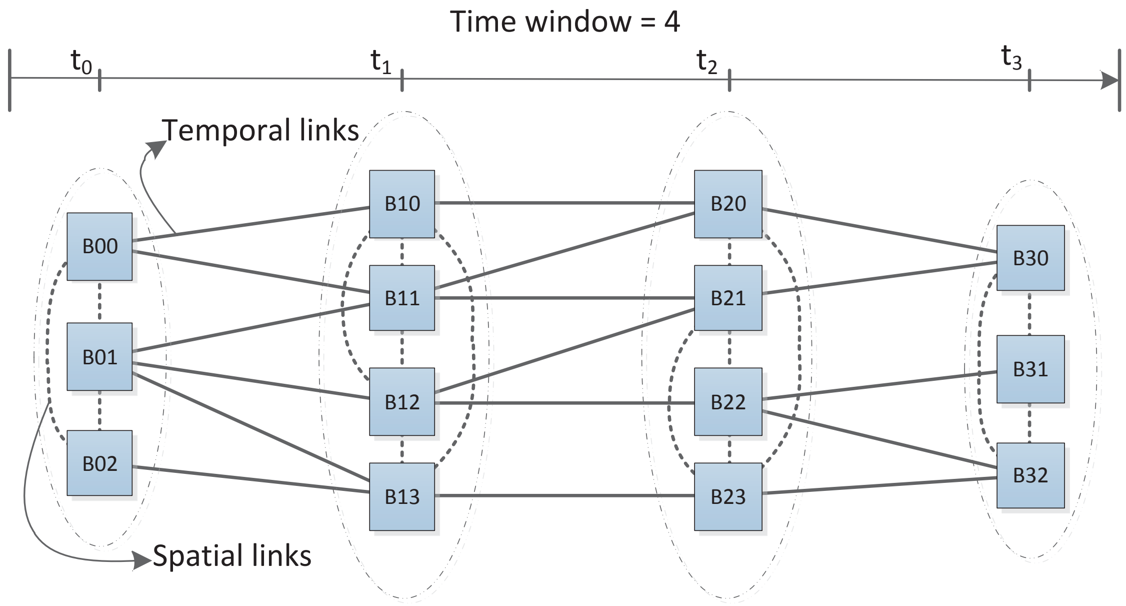


FIGURE 5.2: Congestion evolution graph

The building blocks are small constituents that represent the differently congested blocks of the spatial road network at a point of time in a fine granularity. Each of them have

[3] *Building block* and *block* are used synonymously hereafter.

a set of spatially linked building blocks as *neighbors*. The continuously changing traffic conditions affect their structure in terms of size, shape and location. For example, in the day time the traffic generally remains varied in the different regions, and thus require building blocks in fine granularity to effectively represent the traffic condition. At each subsequent time point their structure keeps changing, as much as the variation in network traffic, and the congestion hotspots keep evolving. The possible operations leading to the change are shifting nodes from one block to another, splitting one into multiple blocks, and merging of multiple blocks into one. We model this temporal evolution of spatially connected building blocks into a structure called *congestion evolution graph*, shown in Figure 5.2. At a time, it maintains a window of t time points in the form of t partites, with two exceptions, i) each partite forms a quasi clique of spatially connected blocks inside, and ii) blocks from each partite have links only to the preceding and following partites.

5.4.2 Stability

During the 24 hours of a day, the traffic load on an urban road network varies from time to time. For example, in early morning the roads are mostly free, and as peak hour draws near, they become busy quite rapidly. The period of time during which the traffic changes from free to congested (or vice versa) is very short for some roads, depending on their spatial importance, which makes the vicinity unstable. After sometime, the traffic gradually approaches towards being stable. Thus *stability* is an important feature of road networks that leads to a better understanding of the spatio-temporal aspects of traffic congestion.

Definition 5.9. (**Node Stability**) If a node v_i belongs to building block $\mathcal{B}_j$ at time t_{r-1}, then the stability $stab^r(v_i)$ is defined as the likelihood of v_i to remain in the same block $\mathcal{B}_j$ at time t_r. ■

We consider two different kinds of node stability, i)*spatial* stability, which looks into how well the feature values of v_i match with those of the rest in $\mathcal{B}_j$ at the current time t_r; and ii) *temporal* stability, which looks into how much stable was v_i in the previous time points. Equation 5.1 shows the formulation to compute its measure, where μ_j^r denotes the mean feature value inside $\mathcal{B}_j$ at time t_r. The formula is an average of two quantities – the first one $stab^{(r-1)}(v_i)$, which is its stability measure from the previous time point t_{r-1}, stands for the temporal stability, and the second quantity, which is from the current time point t_r, stands for the spatial stability. The second quantity firstly gets the normalized distance of the node from the centroid of its block, and then subtracts it from 1 to get the closeness, which determines the spatial stability. Its value ($\in [0, 1]$) becomes 1 when the node feature value is exactly the same as the block mean value. A low value of this measure indicates that the node is less suitable for being part of the corresponding block.

$$stab^{(r)}(v_i) = \frac{\left(stab^{(r-1)}(v_i) + \left(1 - \text{abs}\left(\frac{v_i.f - \mu_j^r}{factor}\right)\right)\right)}{2} \tag{5.1}$$

$$factor = \max\{(\mu_j^r - v_j^{min}), (v_j^{max} - \mu_j^r)\} \tag{5.2}$$

5.4.3 Road Network Motifs

A large graph comprises a large number of nodes and edges, and it is generally not easy to analyze local structural components by looking into the overall graph. Graph (or network) motifs are small connected components that exist in significantly large numbers in a graph globally [137], and form the elementary structures or patterns to make up the whole graph. Motifs have been found to be very useful in understanding the local structural principles of real world graphs. Some examples of commonly found graph motifs are 2-path, 3-path, 3-cycle (triangle), and 4-cycle (rectangle), shown in Figure 5.3.

Definition 5.10. (**Path**) Given a graph $G = (V, E)$, a *path* $p = (v_1, v_2, \ldots, v_{l+1})$ is defined as a sequence of nodes such that each pair of nodes (v_i, v_{i+1}) is a link $\in E$, where $i \in [0, l]$. The *path length* is defined as the number of links l in a path, and a path of length l is called an $l-$path. ■

Definition 5.11. (**Cycle**) Given a graph $G = (V, E)$, a *cycle* $p = (v_1, v_2, \ldots, v_{l+1})$ is defined as a path such that $v_1 = v_{l+1}$. A cycle of length l is called an $l-$cycle, which is also an $l-$path. The shortest cycle is a 3-cycle. ■

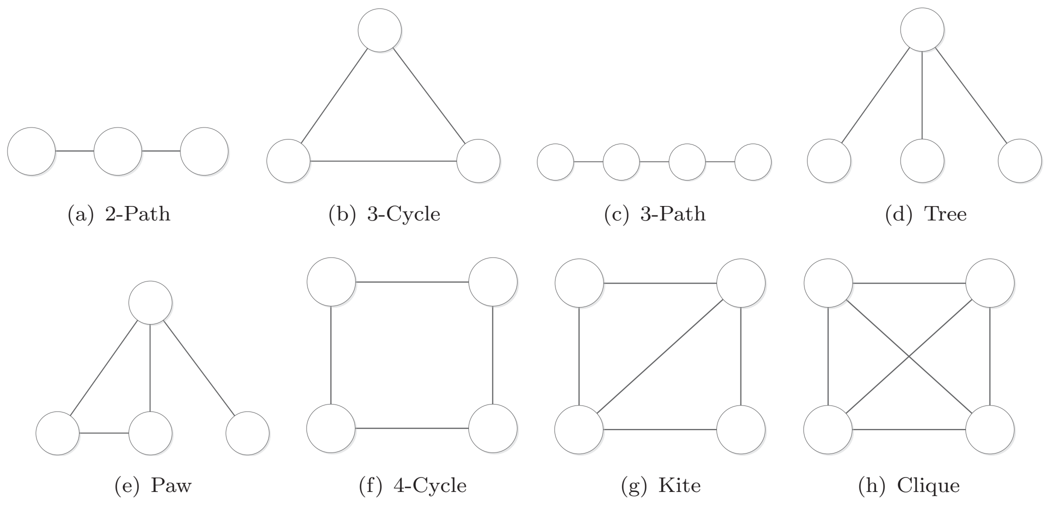


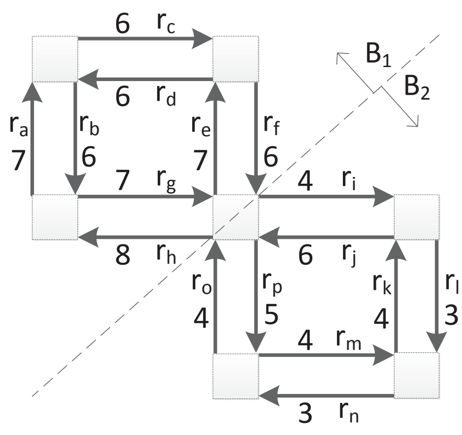
FIGURE 5.3: Graph motif examples with 3 and 4 nodes

Similar to general graph motifs, road networks too have small connected structures that are commonly found all over the network. Generally the intersection points connect four different roads, each of which have their own traffic in the two opposite directions. This kind of intersections lead to the formation of directed cycles of length 4 (4-cycle or rectangle), 6 (6-cycles), and other cycles of higher *even* length in the road network. Triangles, 5-cycles, and other cycles of higher odd length are found only in rare situations. Based on the concepts of graph motifs, we define the following motif concepts in road networks.

Definition 5.12. (**Road Cycle**) Given a road network $\mathcal{N} = (\mathcal{I}, \mathcal{R})$, a path $p = (r_1, r_2, \ldots, r_l)$ comprising l different road segments is said to be an l-*roadcycle* if there exists a path $p' = (r_1, r_2, \ldots, r_l, r_{l+1})$ such that $r_1 = r_{l+1}$ and the traffic flow is directed as $r_1 \to r_2 \to \cdots \to r_l \to r_{l+1}$. ■

Definition 5.13. (**Bounded Road Cycle**) Given a road network $\mathcal{N} = (\mathcal{I}, \mathcal{R})$, a path $p = (r_1, r_2, \ldots, r_l)$ is said to be a γ-*bounded-l-roadcycle* if it is an l-*roadcycle* and the Euclidean distance between each r_i and r_{i+1} in p' is less than or equal to γ. ■

A *bounded road cycle* ensures that it is a tightly bound cycle where all the participating road segments have similar traffic measures up to a certain extent. All such cycles are transformed into paths of the road graph $\mathcal{G}$ as $p = (v_1, v_2, \ldots, v_l)$ where each v_i refers to the node in $\mathcal{V}$ corresponding to road segment r_i in $\mathcal{R}$. Due to the static structure of



(a) A road sub-network

$\mathbf{B_1}$	$\mathbf{B_2}$
r_a-r_c-r_f-r_h	r_i-r_l-r_n-r_o
r_a-r_c-r_d-r_b	r_i-r_l-r_k-r_j
r_a-r_b-r_g-r_h	r_i-r_j-r_p-r_o
r_b-r_g-r_e-r_d	r_j-r_p-r_m-r_k
r_c-r_f-r_e-r_d	r_k-r_l-r_n-r_m
r_e-r_f-r_h-r_g	r_m-r_n-r_o-r_p
r_e-r_f-r_i-r_j	

(b) 2-bounded-4-roadcycles

FIGURE 5.4: Example of γ-*bounded*-$\boldsymbol{l}$-*roadcycles*

.

the road networks, the *road cycles* remain the same throughout, but the dynamic traffic keeps the set of *bounded road cycles* open to change at any point of time.

The building blocks maintained by our method have the properties of high homogeneity and high connectivity inside them. Figure 5.4 shows an example of a small road sub-network (each road segment has its own name and indicative traffic measure) and the list of all $\boldsymbol{\gamma}$-bounded-$\boldsymbol{l}$-roadcycles where $\boldsymbol{\gamma}$ and $\boldsymbol{l}$ are set to 2 and 4 respectively. The network in Figure 5.4(a) is divided by a dashed line into blocks $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$, in which both the blocks have homogeneous traffic measures inside them, with $\boldsymbol{B_1}$ having higher measures than $\boldsymbol{B_2}$. Their difference is clearly identified by looking into the traffic measures and the structure. Figure 5.4(b) shows all the 2-bounded-4-roadcycles. It is found that out of the total of 13 cycles, 6 lie completely inside $\boldsymbol{B_1}$, the same amount being inside $\boldsymbol{B_2}$, and only one cycle shown at the bottom is shared between the two blocks. This observation leads to the fact that the number of occurrences of bounded cycles inside each building block is significantly higher than those across different blocks. These bounded cycles thus give information about the block structure and can be used to locate their suitable boundaries. The lengthy cycles are mostly composed of short cycles (e.g., rectangles). Therefore, instead of all the possible cycles, we consider only the short-length bounded cycles to give an indication about the building block structures.

5.5 Tracking and Capturing the Evolution

We track the spatiotemporal evolution by incrementally updating the building blocks and capture this evolution with the help of the proposed in-memory index. In this section, we present our method in detail.

5.5.1 Index Structure

For efficient computation and economic memory consumption, we index the historical sets of building blocks, the current boundary nodes, and the static short cycles using suitable structures.

5.5.1.1 Building Block Index (`Bin`)

The tracking of the evolution of building blocks needs to maintain the historical information (of selected time points) in the main memory. It demands an index structure that provides a compact in-memory storage and efficient retrieval. We propose an in-memory block index called `Bin` (illustrated in Figure 5.5) to address these requirements. It consists of three different components: a tree structure $\boldsymbol{Tr}$, a list of linked lists $\boldsymbol{TL}$, and an inverted list $\boldsymbol{NI}$. Each block at time $\boldsymbol{t_i}$ is a set of road segments having a unique identifier. These blocks are indexed in the form of a tree $\boldsymbol{Tr}$ having a root node $\boldsymbol{R}$, a set of internal nodes that correspond to the road segments, and a set of leaf nodes that stand for the different time points. All nodes except the root have one parent, and all nodes except the leaves have one or more children. Figure 5.5 shows the index for sets of building blocks at three time points – $\boldsymbol{i)}$ $\boldsymbol{\{\{12,6,1\},\{4,8,7,3,2\},\{11,10,5,9\}\}}$ at $\boldsymbol{t_0}$, $\boldsymbol{ii)}$ $\boldsymbol{\{\{11,12,6,1\},\{8,7,3,2\},\{4,10,5,9\}\}}$ at $\boldsymbol{t_1}$; and $\boldsymbol{iii)}$ $\boldsymbol{\{\{10,11,12,6,1\},\{5,8,7,3,2\},\{9\},\{4\}\}}$ at $\boldsymbol{t_2}$. A block can be retrieved by traversing up from the leaf node until the root node is reached, and the time point of that leaf node indicate the time point of the retrieved block. $\boldsymbol{TL}$ maintains a list of all the time points where each one of them is linked to all the leaf nodes referring to that time. As shown in the figure, $\boldsymbol{t_0}$ from the list is linked to all the leaf nodes referring to $\boldsymbol{t_0}$ in different branches of the tree. To access blocks of a particular time point, say $\boldsymbol{t_0}$, the linked list for $\boldsymbol{t_0}$ can be accessed from $\boldsymbol{TL}$, using which all the leaf nodes corresponding to $\boldsymbol{t_0}$ can be accessed, which further lead to retrieval of all the building blocks by upward tree-traversal. For a fast lookup of the latest time point (supports faster incremental update computations), $\boldsymbol{NI}$ maintains an inverted list of the leaf nodes, which lead to access the corresponding node at the latest time point. The figure shows that at time $\boldsymbol{t_2}$, the node $\boldsymbol{v_1}$ can be accessed by the leaf node in the left most branch (child of 10), and $\boldsymbol{v_2}$ can be accessed by the leaf child of 5. $\boldsymbol{NI}$ is simply used to refer to the building block to which a particular node belongs.

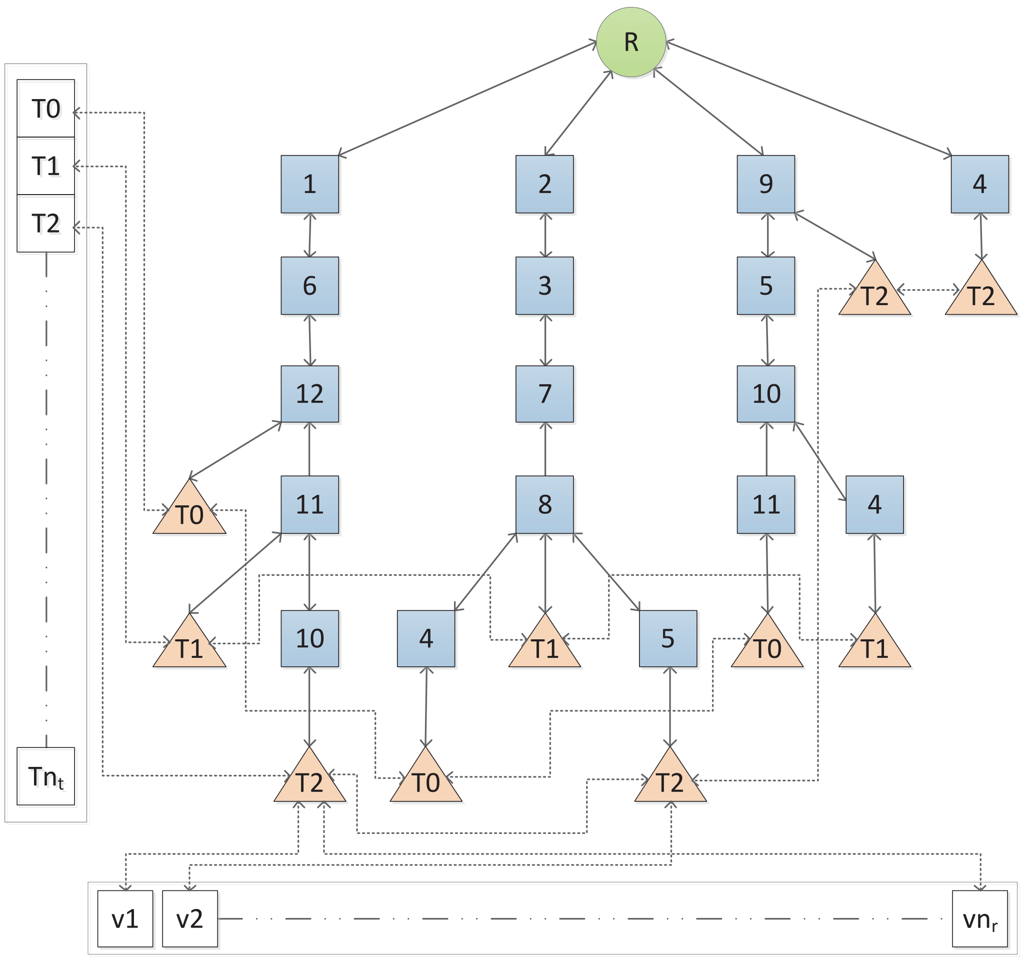


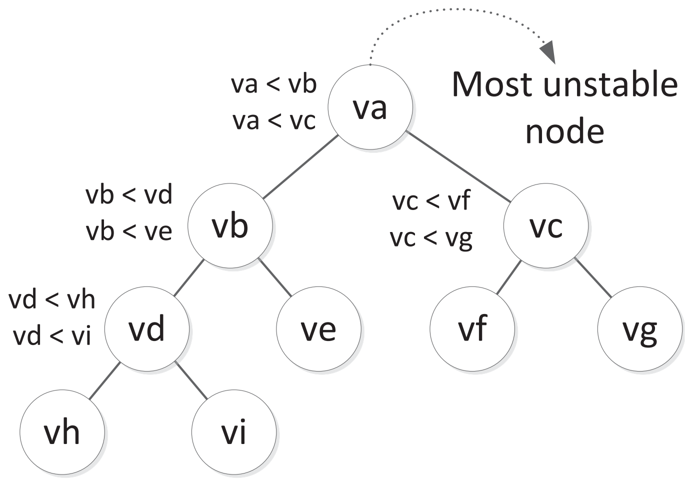
FIGURE 5.5: Bin

5.5.1.2 Stability Tree

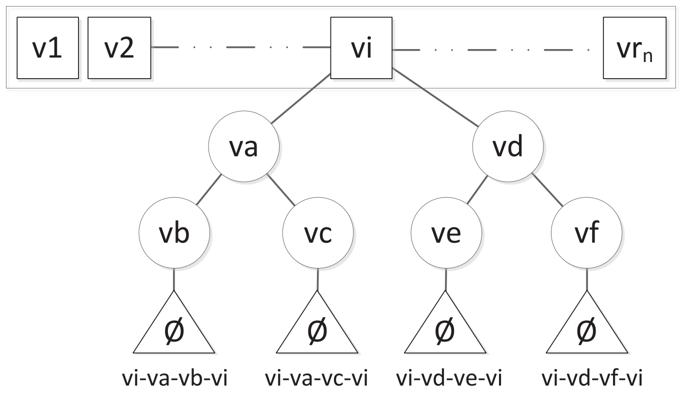
The building blocks have one set of nodes lying on the boundaries (boundary nodes) and another set lying in the interior. As the boundary nodes are directly linked to nodes of a neighboring building block, they are more likely to undergo a change (change in the block to which a node belongs) than the internal nodes in the next time point. Moreover, through boundary adjustment an internal node can undergo a change only after at least one of its linked nodes has undergone a change that makes this internal node a boundary node. Even among the boundary nodes, they all differ in their likelihood of change, which is captured by our stability measure. We use this measure to prioritize the boundary adjustment of nodes with high unstablity. At the beginning time point, we compute all the boundary nodes and index them as follows. A min-heap tree is created based on the stability measure, in which the least stable or the most unstable node becomes the root. This tree is called *stability tree* (denoted by $\boldsymbol{ST}$) and shown in Figure 5.6(a). During the process of boundary adjustment the node with highest priority is processed first, and this min-heap tree is maintained all throughout the incremental update by adding all the newly becoming boundary nodes and removing all the newly becoming internal nodes.

5.5.1.3 Short cycle Index

We pre-compute all the possible ***cycles*** of path length smaller than or equal to $\boldsymbol{\epsilon_{path}}$ in the road graph as an offline task and index them as follows. Firstly a sorted set of all nodes in $\boldsymbol{\mathcal{V}}$ is created based on their id. For each node $\boldsymbol{v_i} \in \boldsymbol{\mathcal{V}}$, all the cycles that involve this node in the path are computed. Let us suppose $\boldsymbol{\{(v_i, v_a, v_b, v_i), (v_i, v_a, v_c, v_i), (v_i, v_d, v_e, v_i), (v_i, v_d, v_f, v_i)\}}$ is the set of cycles involving $\boldsymbol{v_i}$. A trie tree is created having $\boldsymbol{v_i}$ as the root node, $\boldsymbol{v_a}$ and $\boldsymbol{v_d}$ as the children of $\boldsymbol{v_i}$, $\boldsymbol{v_b}$ and $\boldsymbol{v_c}$ as the children of $\boldsymbol{v_b}$, and $\boldsymbol{v_e}$ and $\boldsymbol{v_f}$ as the children of $\boldsymbol{v_d}$. Then the end-marker leaf nodes having the information of their depth in the tree are added as children of $\boldsymbol{v_b}$, $\boldsymbol{v_c}$, $\boldsymbol{v_e}$, and $\boldsymbol{v_f}$. In this way the trie trees corresponding to all $\boldsymbol{v_i} \in \boldsymbol{\mathcal{V}}$ are created and attached to the sorted set as shown in Figure 5.6(b). This structure is called *short cycle index* (denoted by $\boldsymbol{SCI}$).



(a) Stability tree



(b) Short cycle index

FIGURE 5.6: Index for boundary nodes and their short cycles ('<' denotes *less stable than*)

5.5.2 Incremental Update

The incremental update algorithm looks into all the unstable nodes and moves them to the most suitable building blocks at each time point. Broadly it consists of two main tasks – the incremental computation of the set of building blocks for the current time point followed by updating the index `Bin`. Let us suppose we have a given `Bin` and short cycle index $\boldsymbol{SCI}$ at time $\boldsymbol{t_{i-1}}$ for sets of building blocks from time $\boldsymbol{t_0}$ to $\boldsymbol{t_{i-1}}$. The complete algorithm to incrementally compute and update `Bin` at time $\boldsymbol{t_i}$ is shown in Algorithm 8. It starts with computing the stability measures from the traffic data at $\boldsymbol{t_i}$ for all the boundary nodes and creating the stability tree $\boldsymbol{ST}$ (lines 1-2). Then the $\boldsymbol{Tr}$, $\boldsymbol{TL}$, and

$\boldsymbol{NI}$ components of `Bin` are updated to add the leaf nodes for $\boldsymbol{t_i}$ and re-direct the links in $\boldsymbol{TL}$ and $\boldsymbol{NI}$ (lines 3-5). The iterative steps of computing the most suitable building block for the most unstable boundary node and updating `Bin` is carried out until all the boundary nodes having their stability measure less than the threshold $\boldsymbol{\epsilon_{stab}}$ have been processed (lines 6-15). After getting the most unstable node $\boldsymbol{v}$ by deleting the root of $\boldsymbol{ST}$ (line 6), the short cycle tree $\boldsymbol{sctree}$ is accessed from $\boldsymbol{SCI}$ (line 9), and the most suitable leaf node $\boldsymbol{lnext}$ $(\in$ `Bin`$.\boldsymbol{Tr})$ is computed using the function `IdentifyMSL(.)` described later in Algorithm 9 (line 10). This computation is based on the new traffic data at $\boldsymbol{t_i}$ in contrast to the current leaf node $\boldsymbol{lcurrent}$ based on the data at $\boldsymbol{t_{i-1}}$. If the new leaf node is different than the existing one (line 11), then $\boldsymbol{v}$ is deleted from the branch of $\boldsymbol{lcurrent}$ using function `DeleteFromBranch(.)` described later in Algorithm 10 (line 12). It is followed by inserting $\boldsymbol{v}$ into the branch of $\boldsymbol{lnext}$ using function `InsertIntoBranch(.)` described later in Algorithm 11 (line 13). This step completes the update of `Bin` for $\boldsymbol{v}$. The stability tree $\boldsymbol{ST}$ is then updated by inserting all the newly created boundary nodes and deleting those nodes which no more lie on the boundary because of the change using function `AddRemoveSTNodes(.)` described later in Algorithm 12 (line 14). After completing the processing of $\boldsymbol{v}$, the next most unstable node is extracted from $\boldsymbol{ST}$ (line 15) and the same process (lines 7-15) is carried out repeatedly until all the unstable nodes have been processed.

5.5.3 Computing the most suitable block

As mentioned earlier in Section 5.4.3, on the basis of the properties of network motifs and building blocks, the short length bounded road cycles are likely to be found in significantly large numbers within the building blocks rather than crossing multiple of them. A naive way to find the most suitable block for a node $\boldsymbol{v}$ at time $\boldsymbol{t_i}$ is to select the one having the highest number of bounded road cycles with all the nodes lying in the same block. But often there are cycles passing through multiple blocks, where most part of the cycle lie within the most suitable block leaving some fractions in neighboring blocks. The naive method ignores these fractions. Our main idea here is to identify the block that is involved in most part of the bounded road cycles of node $\boldsymbol{v}$. For this, we consider all the road cycles of path length shorter than or equal to $\boldsymbol{\epsilon_{path}}$, making the range as $[\boldsymbol{3}, \boldsymbol{\epsilon_{path}}]$. Each block

Algorithm 8: IncrementalUpdate(Block index `Bin`, Stability tree ST, Short cycle index SCI)

```
1  Compute stability of boundary nodes at $t_i$;
2  Stability tree $ST \leftarrow$ Create a max-heap tree;
3  Update Bin.$Tr \leftarrow$ Add a leaf node $T_i$ as sibling of each $T_{i-1}$;
4  Update Bin.$TL \leftarrow$ Link all $T_i$ and add the list to $TL$;
5  Update Bin.$NI \leftarrow$ Set $(node.leaf \leftarrow T_i)$ instead of $T_{i-1}$, $\forall node \in$ Bin.$NI$;
6  Node $v \leftarrow$ Delete root of $ST$;
7  while $stablity(v) \leq \epsilon_{stab}$ do
8      Leaf $lcurrent \leftarrow$ Bin.$NI.v.leaf$ ;
9      $sctree \leftarrow SCI[v]$;
10     Leaf $lnext \leftarrow$ IdentifyMSL(Bin, $sctree$);
11     if $lnext \neq lcurrent$ then
12         Update Bin $\leftarrow$ DeleteFromBranch(Bin , $lcurrent, v$);
13         Update Bin $\leftarrow$ InsertIntoBranch(Bin, $lnext, v$);
14         Update $ST \leftarrow$ AddRemoveSTNodes(Bin, $ST, v, lcurrent, lnext$);
15     Node $v \leftarrow$ Delete root of $ST$;
16 return Bin;
```

is quantified by a weight function $W(.)$ that considers the total of fractions from all the cycles lying in the respective block. For this quantification, all cycles account for a weight of 1, which is equally divided among all the cycle nodes other than v. For longer cycles the value being divided among more nodes gives lesser power to each. Therefore, the shorter the cycle, the bigger the impact of its nodes.

Formulated in Equation 5.3, $W(v, \mathcal{B}_j)$ computes the weight assigned to block $\mathcal{B}_j$ to identify the most suitable block for v, where $RCycles(v)$ gives all the γ-bounded short road cycles involving v, and u is another node that is involved in the same cycle and belongs to $\mathcal{B}_j$ at time t_i.

$$W(v, \mathcal{B}_j) = \sum_{\substack{\forall C \in RCycles(v) \\ (u \in C) \wedge (u \in \mathcal{B}_j)}} \frac{1}{pathlength(C) - 1} \tag{5.3}$$

In other words, each road cycle C that involves v is traversed, and for each node u in that cycle, if it belongs to block $\mathcal{B}_j$, the weight for $\mathcal{B}_j$ is incremented by $\frac{1}{pathlength(C)-1}$.

Algorithm 9: IdentifyMSL(Block index `Bin`, Short cycle tree ***sctree***)

```
1  $s1, s2, s3 \leftarrow$ initialize new stack;
2  Node $nparent \leftarrow sctree(root)$;
3  forall Node nchild $\in sctree(root).child$ do
4      if $nchild \neq \phi$ AND $dist(nparent, nchild) \leq \gamma$ then
5          Push $nchild$ into $s1$;
6  repeat
7      Node $nparent \leftarrow$ pop out from $s1$;
8      Push $nparent$ into $s2$;
9      $count = 0$;
10     forall Node nchild $\in ctree(nparent).child$ do
11         if $nchild \neq \phi$ AND $dist(nparent, nchild) \leq \gamma$ then
12             Push $nchild$ into $s1$;
13             $count \leftarrow count + 1$;
14     Push $count$ into $s3$;
15 until $s1 \neq empty$;
16 $s1 \leftarrow$ re-initialize stack;
17 $wleaf[\,] \leftarrow$ initialize an array of values for each leaf of current time point in Bin.$Tr$;
18 repeat
19     Node $node \leftarrow$ pop out from $s2$;
20     $countchild \leftarrow$ pop out from $s3$;
21     if $countchild = 0$ then
22         Value $weight = \frac{1}{depth(node)}$;
23     else
24         Value $weight \leftarrow$ initialize with 0;
25         for $i \leftarrow 1$ to $countchild$ do
26             Value $childweight \leftarrow$ pop out from $s1$;
27             $weight \leftarrow weight + childweight$;
28     Push $weight$ into $s1$;
29     $wleaf$[Bin.$NI.node.leaf$] $\leftarrow$ Increment by $weight$;
30 until $s2 \neq empty$;
31 $wmax \leftarrow wleaf[0]$;
32 forall $wl \in wleaf[\,]$ do
33     if $wmax < wl$ then
34         $wmax \leftarrow wl$;
35 $msl \leftarrow$ leaf corresponding to $wmax$;
36 return $msl$;
```

For example, if there is a cycle $\{v_1(\in \mathcal{B}_1), v_2(\in \mathcal{B}_2), v_3(\in \mathcal{B}_1), v_4(\in \mathcal{B}_1)\}$, each node (except the one for which the weight is being computed) will have the power to make an affect by $\frac{1}{3}$, which in total equals to 1. To compute $W(v_1, \mathcal{B}_1)$, v_3 and v_4 both belonging to $\mathcal{B}_1$ adds up to $\frac{2}{3}$, and the weight $\frac{1}{3}$ of $v_2 \in \mathcal{B}_2$ is added to $W(v_1, \mathcal{B}_2)$. In this manner, the weights for all the blocks are computed by adding the values from all the different cycles, and the one with the highest weight is selected as the most suitable block. However, as the number of road cycles is usually large, traversing all of them individually to compute the weight for the building blocks in this way adds a lot of computations, and affects the running time.

In most of the road cycles in which a node v is involved, there exists overlapping of some parts of the complete path of multiple cycles. For example, the cycles $C_1 = \{v, v_1, v_2, v_3\}$ and $C_2 = \{v, v_1, v_2, v_4\}$ of v have three overlapping nodes (v, v_1 and v_2). Computing the weights by traversing through C_1 and C_2 independently, repeats the computations done for v, v_1 and v_2. We make use of these overlappings in computing the building block weights, thereby avoid redundant computations. This is done by our **multi-stack based algorithm** (shown in Algorithm 9) with the help of our short cycle index SCI that keeps the cycles indexed as a tree, having no repeating nodes even for the overlapping cycles. It accesses the block index `Bin` and the short cycle tree $sctree$ from SCI, and computes the most suitable leaf ($\in$ `Bin.`Tr) for node v (root node of $sctree$) at time t_i. The stacks used in the algorithm explore the cycles in $sctree$ and keep part of the computed information saved for its reuse later for the overlapping cycles.

The algorithm starts with pushing all the children of root node of $sctree$ to the stack $s1$, if the Euclidean distance between the parent and the child $dist(nparent, nchild)$ is less than or equal to γ (lines 1-5). The $dist(.)$ function ensures that only the γ-bounded road cycles are explored. It is followed by iterative steps of popping out a node from $s1$ (line 7), pushing it into the stack $s2$ (line 8), pushing all its children back into $s1$ if the parent-child satisfies the γ distance condition (lines 10-13), and pushing the count of these children into the stack $s3$. These steps are repeated until $s1$ becomes empty (line 15). They compute the number of children of each node and keep them saved in the stacks for computing the block weights (or leaf weights) later. Thereafter an array of values is initialized to store the weights $W(.)$ computed for each block in order to select the most suitable one (line 17). As the blocks are retrieved by the leaf nodes in `Bin`, the weights correspond to the leaf nodes of `Bin.`Tr at time t_i. Then the nodes are popped out from $s2$ one after another (line 19), the count of their children are popped out from $s3$ (line 20), followed by a set of

Algorithm 10: DeleteFromBranch(Block index `Bin`, Leaf $\boldsymbol{lcurrent}$, Node $\boldsymbol{v}$)

```
1  Node $\boldsymbol{nodepre} \leftarrow \boldsymbol{lcurrent.parent}$;
2  Node $\boldsymbol{node} \leftarrow \boldsymbol{nodepre.parent}$;
3  if $\boldsymbol{nodepre} \neq \boldsymbol{v}$ then
4      repeat
5          if $\{\boldsymbol{node.child} \backslash \boldsymbol{nodepre}\} \neq \phi$ then
6              Node $\boldsymbol{vclone} \leftarrow$ new clone of $\boldsymbol{v}$;
7              $\boldsymbol{vclone} \xleftrightarrow{\text{parent-child link}} \{\boldsymbol{node.child} \backslash \boldsymbol{nodepre}\}$;
8              $\boldsymbol{node} \xleftrightarrow{\text{parent-child link}} \boldsymbol{vclone}$;
9          $\boldsymbol{nodepre} \leftarrow \boldsymbol{node}$;
10         $\boldsymbol{node} \leftarrow \boldsymbol{node.parent}$;
11     until $\boldsymbol{nodepre} = \boldsymbol{v}$;
12     $\boldsymbol{node} \xleftrightarrow{\text{parent-child link}} \{\boldsymbol{nodepre.child}\}$;
13     $\boldsymbol{lcurrent.parent} \xleftrightarrow{\text{parent-child link}} \boldsymbol{nodepre}$;
14     $\boldsymbol{nodepre} \xleftrightarrow{\text{parent-child link}} \{\boldsymbol{lcurrent.parent.child}\}$;
15 Set $\boldsymbol{T_i}$ as child of $\boldsymbol{nodepre.parent}$;
16 return $Bin$;
```

steps to compute the weights. A value of 0 as the count of children indicates that it is the last node in the branch and its *depth* gives the path length of the cycle. Hence the weight of $\frac{1}{\boldsymbol{pathlength(C)-1}}$ that is carried by each node in the cycle $\boldsymbol{C}$ is computed as $\frac{1}{\boldsymbol{depth(node)}}$ (line 22) and pushed into $\boldsymbol{s1}$ (line 28) to be used later to compute weights for its parents. The depth of the node is found from its leaf node that contains the depth information. If the count of children is non-zero, the weight is computed by adding the weights of all its children obtained by popping up $\boldsymbol{s1}$ as many times as the count (lines 23-27), which is again pushed back into $\boldsymbol{s1}$ (line 28). In addition to pushing the weight of nodes into $\boldsymbol{s1}$, the weight for the leaves in the array $\boldsymbol{wleaf}[\ ]$ is updated by adding $\boldsymbol{weight}$ to the array element corresponding to the leaf of $\boldsymbol{node}$, i.e., `Bin`.$\boldsymbol{Tr.node.leaf}$. These steps of popping out from $\boldsymbol{s2}$ and $\boldsymbol{s3}$ and computing the values of $\boldsymbol{wleaf}[\ ]$ using $\boldsymbol{s1}$ (lines 19-29) are repeated until $\boldsymbol{s2}$ becomes empty (lines 18,30), which marks the completion of processing of all the bounded short road cycles. At last we select the most suitable leaf for the current time $\boldsymbol{t_i}$ by getting the leaf corresponding to the maximum weight in $\boldsymbol{wleaf}[\ ]$ (lines 31-36).

5.5.4 Updating Index

After identifying the most suitable leaf in the index `Bin` at time t_i for a node v, the remaining task is to update `Bin` and ST with this change. This is done by the functions `DeleteFromBranch(.), InsertIntoBranch(.)`, and `AddRemoveSTNodes(.)`. The operations of inserting/deleting a node into/from a branch defined on `Bin` are performed in such a manner that the data redundancy at different time points is minimum and there is no information loss in the historical data. This section explains each of the three mentioned functions individually.

The function `DeleteFromBranch(.)` (Algorithm 10) is called to delete a node v from its current branch in `Bin` Its main objective is to delete v in such a way that the block retrieved for time t_{i-1} includes v in the set of nodes but excludes it for time t_i. For this we start from the leaf of its branch, and traverse upwards towards the root using two pointers $\boldsymbol{nodepre}$ and $\boldsymbol{node}$ until v is found (lines 1-11). During the traversal, we check if there are other nodes that do not lie in the traversal path (branch out somewhere). It is done by looking into the set of children of $\boldsymbol{node}$. If there are other children in addition to $\boldsymbol{nodepre}$ (line 5), then a new node $\boldsymbol{vclone}$ is created as a clone of v (line 6), and inserted as parent of all the siblings of $\boldsymbol{nodepre}$ (line 7) and as child of $\boldsymbol{node}$ (line 8). This process is continued until $\boldsymbol{nodepre}$ reaches v (line 11), after which v is simply removed from there by making all its children (or that of $\boldsymbol{nodepre}$) as the children of $\boldsymbol{node}$ (line 12). This node is added as child of the parent of leaf $\boldsymbol{lcurrent.parent}$ (line 13) and as parent of all the children of $\boldsymbol{lcurrent.parent}$ (line 14). At the end, the leaf $\boldsymbol{lcurrent}$ (referring to T_i) is set as the child of $\boldsymbol{nodepre.parent}$ or sibling of $\boldsymbol{nodepre}$ (line 15). After doing this change, the block retrieved by traversing from T_i to root includes all the previous nodes except v, whereas the set of nodes remain the same for all other time points or leaves. By inserting additional clones of v, we add some redundancy, but it is kept the minimum that is required to keep all the information for previous time points without any loss. A *delete* operation takes the node closer to the leaf. The unstable nodes, which frequently undergo *deletion* from a branch, are quickly found in the upward traversal from the leaf, thereby improves its efficiency for such nodes.

Algorithm 11: InsertIntoBranch(Block index `Bin`, Leaf $\boldsymbol{lnext}$, Node $\boldsymbol{v}$)

```
1 Node $\boldsymbol{node} \leftarrow \boldsymbol{lnext.parent}$;
2 if $\boldsymbol{v} \notin \{\boldsymbol{node.child}\}$ then
3     Create a node for $\boldsymbol{v}$;
4     $\boldsymbol{node} \xleftrightarrow{\text{parent-child link}} \boldsymbol{v}$;
5 Set $\boldsymbol{T_i}$ as child of $\boldsymbol{v}$;
6 return Bin;
```

Algorithm 12: AddRemoveSTNodes(Block index `Bin`, Stability tree $\boldsymbol{ST}$, Node $\boldsymbol{v}$, Leaf $\boldsymbol{lcurrent}$, Leaf $\boldsymbol{lnext}$)

```
1 forall Node $\boldsymbol{u} \in \boldsymbol{neighbour(v)}$ do
2     if Bin.$\boldsymbol{NI.u.leaf} = \boldsymbol{lcurrent}$ AND $\boldsymbol{u} \notin \boldsymbol{ST}$ then
3         Insert $\boldsymbol{u}$ into $\boldsymbol{ST}$;
4     if Bin.$\boldsymbol{NI.u.leaf} = \boldsymbol{lnext}$ AND $\boldsymbol{u} \in \boldsymbol{ST}$ AND $\boldsymbol{u}$ does not lie on boundary then
5         Delete $\boldsymbol{u}$ from $\boldsymbol{ST}$;
6 return $\boldsymbol{ST}$;
```

After the deletion of $\boldsymbol{v}$ from its previous branch it is inserted into the most suitable branch $\boldsymbol{lnext}$ in `Bin` using the function `InsertIntoBranch(.)` (Algorithm 11). The main idea here is that if $\boldsymbol{v}$ already exists there as a sibling of $\boldsymbol{lnext}$ (referring to $\boldsymbol{T_i}$), then without adding any new node, $\boldsymbol{lnext}$ is simply set as the child of $\boldsymbol{v}$ (line 5). Otherwise (line 2), a new node is created for $\boldsymbol{v}$ (line 3) and added as a child of the parent of $\boldsymbol{lnext}$ (line 4), after which $\boldsymbol{lnext}$ is set as the child of $\boldsymbol{v}$ (line 5). After this update, the complete block including $\boldsymbol{v}$ can be retrieved directly by traversing upwards from $\boldsymbol{lnext}$ to the root. As the insertion takes place at the leaf, the time complexity of this operation is always $\mathbf{O}(1)$.

The stability tree $\boldsymbol{ST}$ contains all the boundary nodes in the road network at a point of time, heapified based on their stability measure. After deleting its root to get the unstable nodes and process them, if the node changes its branch in `Bin`.$\boldsymbol{Tr}$ (or changes it block), several nodes newly become boundary nodes. On the other hand, several nodes that were on the boundary earlier, no more lie on the boundary. The function `AddRemoveSTNodes(.)` (Algorithm 12) updates $\boldsymbol{ST}$ by adding the new boundary nodes into the tree and removing the internal nodes from the tree. It starts with getting all the nodes $\boldsymbol{u}$ linked to $\boldsymbol{v}$ in the road graph (line 1). If the leaf of $\boldsymbol{u}$ referred in the node inverted list `Bin`.$\boldsymbol{NI.u.leaf}$ is same as the leaf node of $\boldsymbol{v}$ before the update $\boldsymbol{lcurrent}$ (line 2), it means that $\boldsymbol{u}$ is a boundary node. If it does not exist in $\boldsymbol{ST}$, being a new boundary node it is added to the tree (lines 2-3). If the leaf of $\boldsymbol{u}$ referred in `Bin`.$\boldsymbol{NI.u.leaf}$ is same as the leaf node of $\boldsymbol{v}$ after the update $\boldsymbol{lnext}$ (line 4), it means that $\boldsymbol{u}$ may not be a boundary node anymore. If it does not exist in $\boldsymbol{ST}$ and does not lie on the boundary, it is removed from the tree (lines 4-5). Everytime a node $\boldsymbol{v}$ changes its branch in `Bin`.$\boldsymbol{Tr}$, all its linked nodes are checked to keep an up-to-date collection of the boundary nodes.

5.6 Experiments

We conducted experiments on four datasets, and present our evaluation results in three main aspects: the quality of incremental results, the efficiency of incremental computations, and the memory consumption for storing the historical information in `Bin`. We also show the effects of external parameters and visualize some obtained results.

5.6.1 Datasets

Our datasets, shown in Table 5.1, include one real ($\mathtt{M}_{\boldsymbol{s}}$) and three semi-synthetic ($\mathtt{M}_{\mathbf{1}}$, $\mathtt{M}_{\mathbf{2}}$, and $\mathtt{M}_{\mathbf{3}}$) datasets. $\mathtt{M}_{\boldsymbol{s}}$ is recorded by the SCATS System (please refer to Chapter 6, Section 6.1, for a detailed description of the SCATS system) from the Melbourne road network provided to us by VicRoads (road and traffic authority in the state of Victoria, Australia)[4]. It is an accumulation of the traffic records of individual road segments for each signal cycle from 1st Jan 2011 to 1st Jan 2013. The considered Melbourne network consists of 7245 road segments and 2928 intersection points. The traffic measures include traffic volume (number of vehicles crossing a road segment during the green time) and degree of saturation (the ratio of the effectively used green time to the total available green time). We consider degree of saturation as the feature value of road segments, which approximates the traffic density. $\mathtt{M}_{\mathbf{1}}$, $\mathtt{M}_{\mathbf{2}}$, and $\mathtt{M}_{\mathbf{3}}$ are synthetically generated on the real road network of Melbourne using a web-based[5] random road traffic generator MNTG [133]. We populated 25,246, 62,300, and 84,999 vehicles respectively, and obtained their movement trajectories for 100 continuous time points. The trajectories are sequences of

[4]`https://www.vicroads.vic.gov.au/`

[5]It can be accessed through `http://mntg.cs.umn.edu/tg/`

100 or less ⟨latitude,longitude⟩ pairs corresponding to vehicle positions at each timestamp. The vehicle positions are mapped to corresponding road segments and the traffic density measures (in terms of vehicles/meter) are computed at each time point. We use traffic density as the feature value of road segments for these datasets.

Table 5.1: Dataset statistics

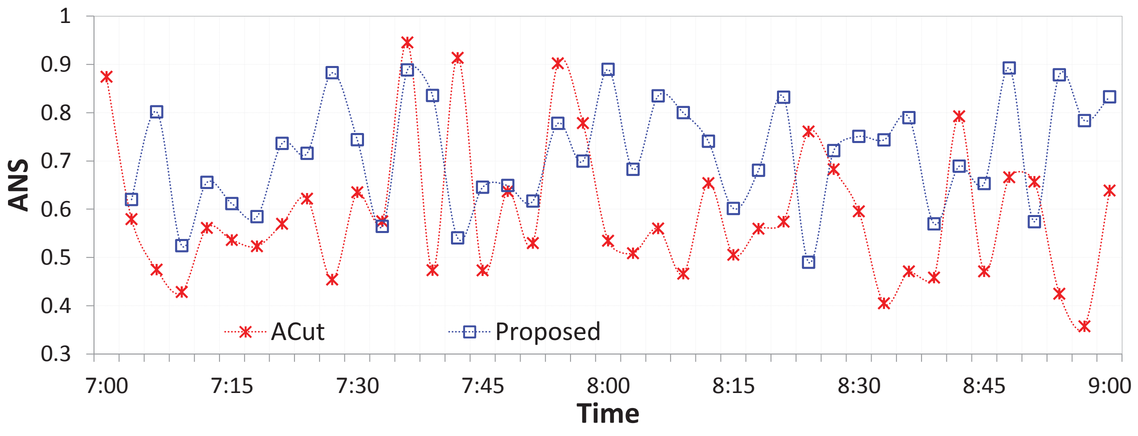
Dataset	**Place**	**Area(sq ml)**	**Road seg**	**Inter pts**
$\mathtt{M}_{\boldsymbol{s}}(\textit{real})$	Melbourne	627.5	7245	2928
$\mathtt{M}_{\mathbf{1}}$	CBD Melbourne	6.6	17,206	10,096
$\mathtt{M}_{\mathbf{2}}$	CBD(+) Melbourne	31.5	53,494	28,465
$\mathtt{M}_{\mathbf{3}}$	Melbourne	42.03	79,487	42,321

5.6.2 Evaluation Metrics

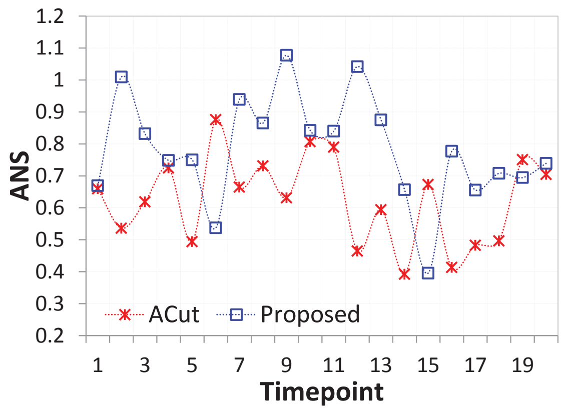
The quality of the incremental partitioning results are evaluated by using the metric *ANS* that looks into the inter-partition heterogeneity and intra-partition homogeneity and quantify the quality of the obtained partitions [3, 4, 61]. It is derived from the standard Silhouette measure used for cluster evaluation. A value less than 1 for this measure indicates a good partitioning, and lower values indicate better partitioning.

5.6.3 Quality of Incremental Results

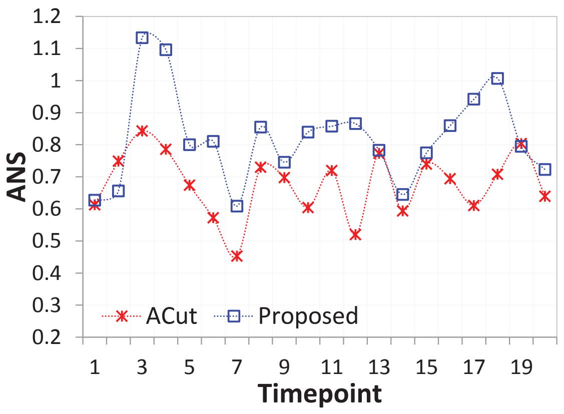
We start the experiments by firstly mining a set of building blocks that are to be maintained and incrementally updated in the physical layer. For this, we partition the road network into a relatively large number of small sized partitions and consider them as the building blocks for the beginning time point. As a preprocessing step, all the short road cycles are computed and indexed in SCI. We evaluate the quality of our results by comparing them with the partitions obtained by directly applying the $\boldsymbol{\alpha}$-Cut partitioning algorithm [3, 4] on the considered dataset. Figure 5.7 shows this comparison on `Ms`, `M1` and `M2` datasets in terms *ANS*. The measure for the proposed method is computed by incrementally maintaining a set of 200 building blocks in the physical layer and obtaining a set of 10 partitions from the logical layer at each of the shown time points. The results shown on the `Ms` dataset refer to the real scats traffic data on 03-12-2012 (Monday). We



(a) ANS on `Ms`



(b) ANS on `M1`



(c) ANS on `M2`

FIGURE 5.7: Partitioning quality

observe that the curve of the proposed method is most of the time higher but close to that of the α-Cut, which means that the incrementally obtained partitions by the proposed method are not significantly inferior than those obtained by partitioning the road network directly using α-Cut. Also, the ANS measure is always less than 1 for the real `Ms` dataset and most of the times less than 1 for the `M1` and `M2` datasets, which indicates that our results are still considerably good in quality. The main objective of the proposed method is to maintain the partitions efficiently, which lacks in α-Cut or any other road network (or graph) partitioning algorithm.

5.6.4 Efficiency of Incremental Computations

As the proposed method continuously operates to incrementally update the building blocks in real-time, its efficiency is of great importance. Table 5.2 shows the average running time[6] of the logical layer for producing partitions from the building blocks and that of the physical layer for incrementally updating the building blocks at each time point, for all the considered datasets. Its shows the running time by varying the number of maintained building blocks, in comparison to [3]. We observe that the lower the number of maintained building blocks, the faster is the method. The logical layer performs faster because it has to partition a smaller graph where the number of nodes is the number of maintained building blocks, and the physical layer is faster because of lesser overhead. When there is a large number of building blocks, it creates a large number of unstable and noisy road segments lying on the boundaries, which too often shift themselves from one block to another, leading to an overhead. Therefore selecting the right number of building blocks for an application environment is important for the method to have stable building blocks and partitions. The longest running time for the real dataset `Ms` (Melbourne road network) is just a few seconds, which shows its applicability for the real urban road networks. It can also be performed in fractions of a second by maintaining less number of blocks. The longest running time shown in the table is around 6 minutes for the largest dataset (`M3`) on an ordinary PC, which may drastically improve on a high performance computer. Moreover it can also be performed in shorter time with less number of blocks. Comparing these times with the existing re-partitioning method [3], we observe that even our longest running time is significantly lower than the existing method. Thus it suggests that our method can be effectively used with any real traffic management systems by correctly setting its parameters to deal with the real situation.

5.6.5 Memory Consumption in `Bin`

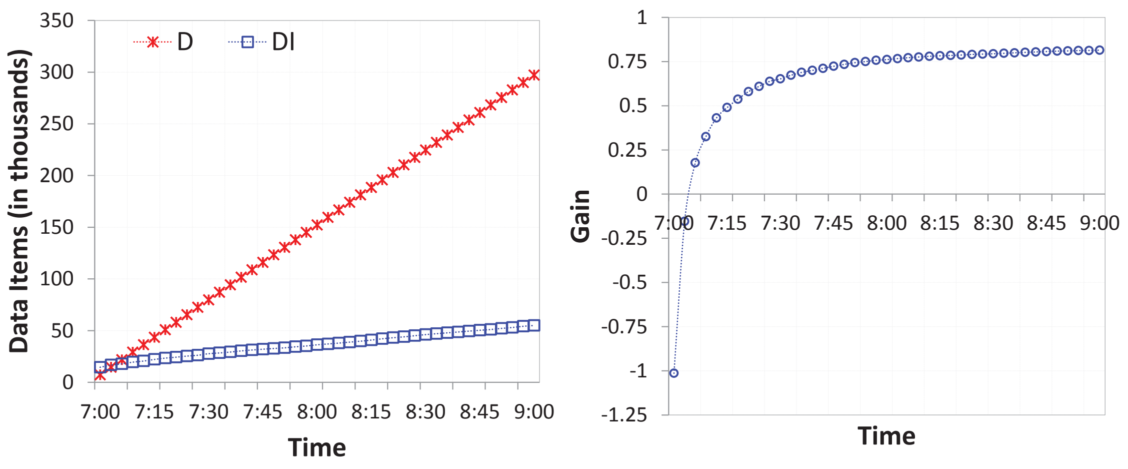
As the incremental update continuously generates data, the compact in-memory storage of this historical data for later use is important. Without using any index, a system would simply dump the sets of building blocks at each new time point, which would proportionally

[6]On a Core i5 computer with 8GB RAM

Table 5.2: Running time (in seconds)

Number of maintained building blocks		**Dataset**			
		`Ms`	`M1`	`M2`	`M3`
100	Logical layer	<1	<1	<1	<1
	Physical layer	<1	1	72	129
200	Logical layer	<1	<1	<1	<1
	Physical layer	<1	1	86	184
400	Logical layer	<1	<1	1	1
	Physical layer	<1	1	96	218
600	Logical layer	1	1	1	1
	Physical layer	<1	1	117	265
800	Logical layer	3	3	5	7
	Physical layer	1	1	145	307
1000	Logical layer	10	12	15	16
	Physical layer	1	6	173	354
[3, 4] (α-Cut)		98	129	1905	5907

keep on increasing the memory usage with lots of redundant information. In contrast, our `Bin` stores the complete information with minimum redundancy. The compactness of our index is quantified by the measure $\boldsymbol{gain} = \frac{\boldsymbol{D-DI}}{\boldsymbol{D}}$, where $\boldsymbol{D}$ is the number of data items stored in memory when no index is used, and $\boldsymbol{DI}$ is the number of data items stored in memory using `Bin`. Figure 5.8(a) shows the number of data items stored for maintaining 100 building blocks of the `Ms` dataset, and 5.8(b) shows our gain in memory consumption. The gain starts with even less than -1, but keeps on increasing with time and crosses 0.8 within two hours. The reason for a negative gain in the beginning is that it needs to store the extra information in `Bin`.$\boldsymbol{TL}$ and `Bin`.$\boldsymbol{NI}$ in addition to the blocks in `Bin`.$\boldsymbol{Tr}$. For the `Ms` dataset, the number data items stored in these components in the first time point are 7245 (for 7245 road segments), 100 (for maintaining 100 blocks), and 7246 (road segments + root node in the tree), respectively. Over the period of time, `Bin`.$\boldsymbol{NI}$ consumes the same space, and `Bin`.$\boldsymbol{TL}$ and `Bin`.$\boldsymbol{Tr}$ keep increasing, but in a slow rate.



(a) Stored data items (b) Gain

FIGURE 5.8: Memory consumption in `Bin`

.

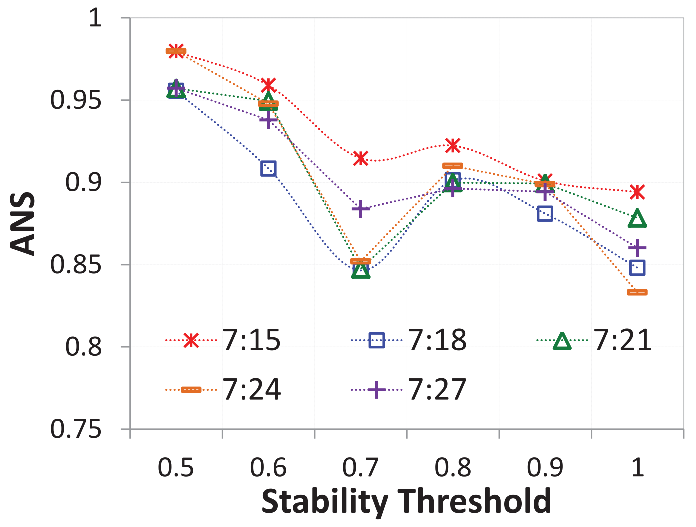
5.6.6 Effects of external parameters on blocks

There are some external parameters that are required to be set for the experiments, and have some effects on the results. Figure 5.9 shows the effects of stability threshold $\boldsymbol{\epsilon_{stab}}$, the number of incrementally maintained building blocks $\boldsymbol{n_b}$, and the roadcycle length $\boldsymbol{l}$. The results are shown from the experiments on the real data `Ms`. Figure 5.9(a) shows the ANS measures of the set of building blocks at varying $\boldsymbol{\epsilon_{stab}}$ at five different time points. We observe that the quality of results are best at $\boldsymbol{\epsilon_{stab} = 0.7}$ and 1. When it is 1, all the boundary nodes are considered unstable processed once, because of which it gives good results. Interestingly 0.7 also produces good results for all the time pints, which may be because nodes above this $\boldsymbol{\epsilon_{stab}}$ are very stable, and processing them worsens the situation until all the nodes in the stability tree are processed. Similarly, Figure 5.9(b) shows the ANS measure of the building blocks at five different time points when the number of such maintained blocks are from 200 to 1000. We observe that the more the number of partitions maintained, the better is the quality. Figure 5.9(c) shows the ANS measure of the building blocks obtained by setting the roadcycle length $\boldsymbol{l}$ to 4, 6, and 8. There exists 26622, 165971, and 959528 roadcycles respectively for each value of $\boldsymbol{l}$ in the considered network. It is very rare to find roadcycles of odd length, because of which we test with only even lengths. We observe that the quality of results generally gets better upto some extent with increasing cycle length. The reason for this behavior is that the cycles with high $\boldsymbol{l}$ include all the shorter cycles, and thus check the homogeneity in a larger region. With a small $\boldsymbol{l}$, it checks only a very local region, and may lead to results that are not very good with respect to the global data. However, exceptional situations may occur in case of abrupt changes in the traffic. One such example is 07:24 AM, where 4-roadcycles give better result than 6-roadcycles.

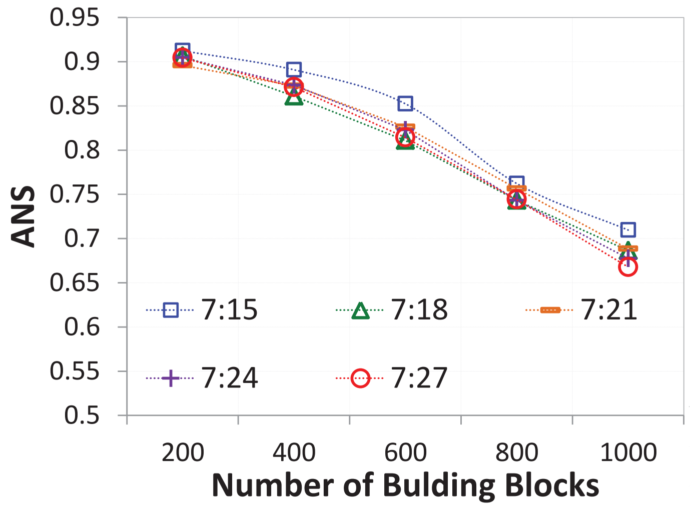
5.6.7 Visualization comparison with Google Traffic

This section demonstrates the usefulness of the proposed framework in traffic visualization in an informative way. As shown in Figure 5.1, `Google Traffic` simply visualizes a heatmap of the traffic data on individual road segments. Its limitations are, ***i***) there is no information about how the multiple independent or linked congestions are spread, and ***ii***) very limited features to understand the temporal evolution. In contrast, our framework is able to produce an information rich visualization shown in Figures 5.10 and 5.11. To keep the pictures simple and easy to understand, only one direction[7] of traffic flow is shown. Figure 5.10 shows 4 color-coded partitions from the `Ms` at 07:06 AM computed in the logical layer. It gives a high level information of the traffic scenario at this time. Each single color has homogeneous traffic inside. If we are interested further, we can zoom-in into the partitions to see the actual building blocks being maintained in `Bin` in the physical layer. Figures 5.11(a), 5.11(b), 5.11(c), and 5.11(d) show the building blocks of the four partitions. A large number of maintained building blocks would produce the zoomed-in view in a fine granularity. Looking into the partitions and building blocks, it is easy to understand how the congestion is spread. Some partitions or building blocks may be congested, others may be non-congested (can be identified from the color codes). There may be multiple independent congestion blocks in different regions at the same time. In the temporal domain, the evolution in those congestion blocks can be seen. A common phenomenon is that different independent congestions start growing from different regions in busy hours, merge with others to form a big congestion in a the peak time, and then gradually disperse back again. These evolutions can be understood clearly with the help of our visualizations. A commuter can make use of this information in planning a journey.

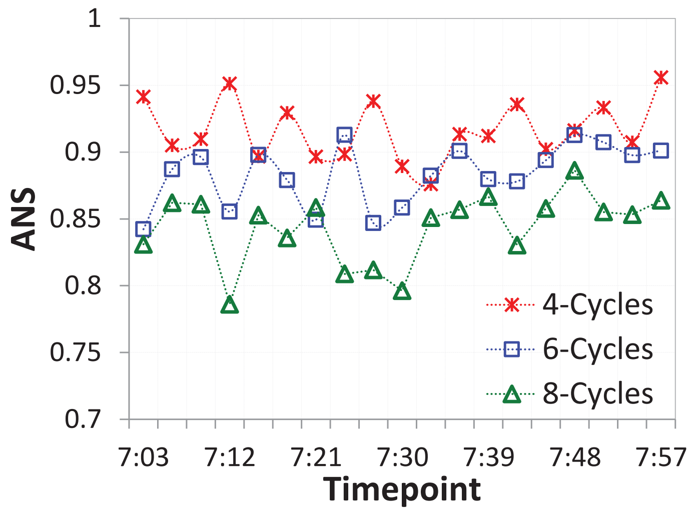
[7]The framework considers the two traffic directions as separate connected road segments.



(a) Stability threshold



(b) Number of building blocks



(c) Length of roadcycle

FIGURE 5.9: Effects of external parameters

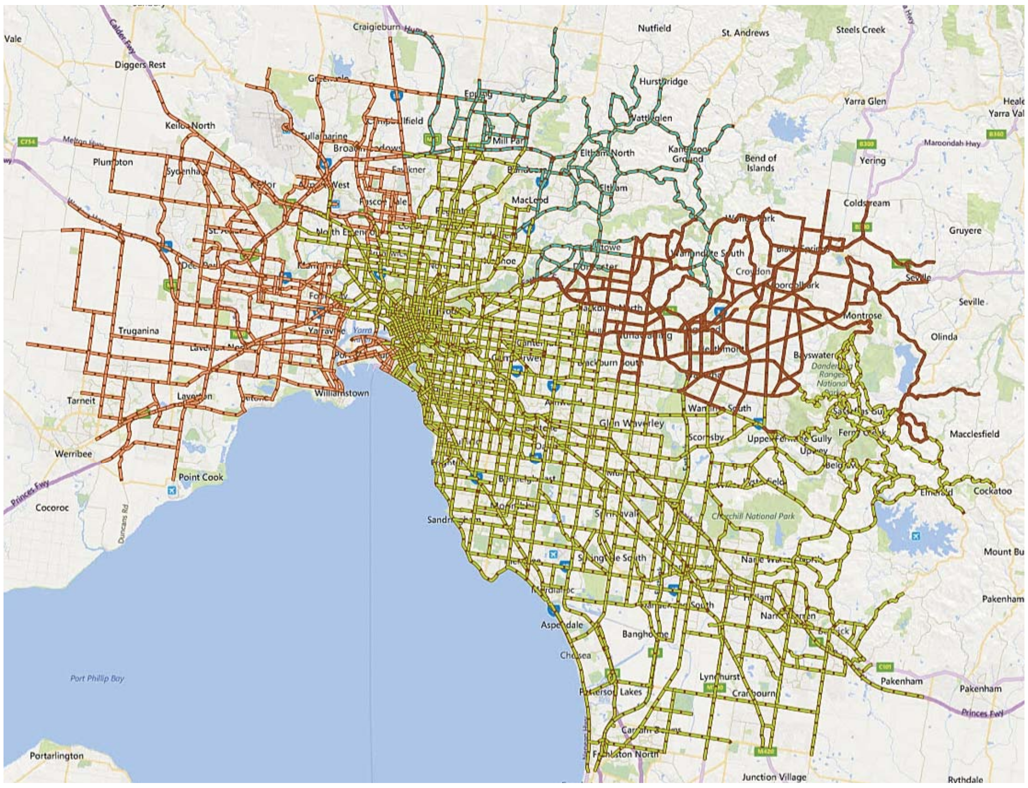
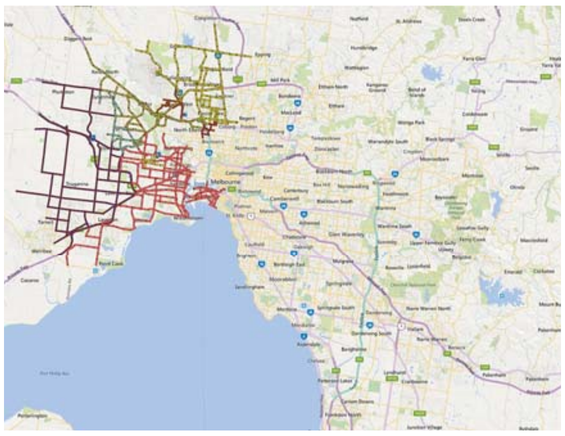


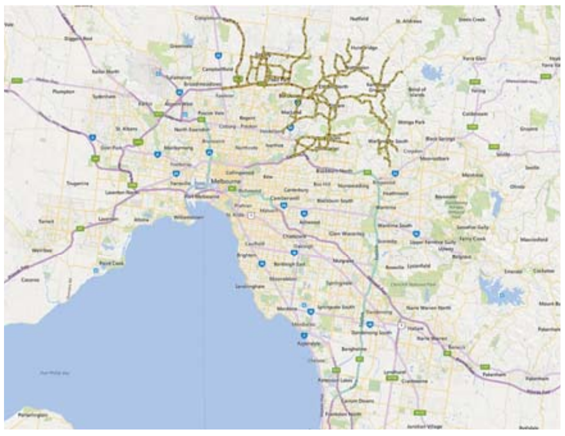
FIGURE 5.10: Partitions in the Logical layer at 07:09 AM

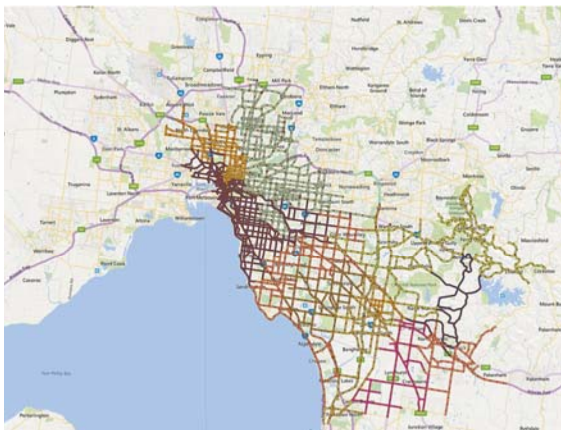
.

5.7 Summary

In this chapter, we proposed a comprehensive framework using a two-layer approach to incrementally update the differently congested partitions of a road network in an efficient manner, and capture the evolution of traffic congestion. The physical layer incrementally maintains a set of small-sized building blocks of the network, and the logical layer provides an interface to query the physical layer about the congested partitions. We also proposed `Bin`, an in-memory index for a compact storage and efficient retrieval of the historical building blocks. We conducted extensive experiments on real and synthetic datasets to demonstrate the efficacy of our method. The method is more efficient than the existing re-partitioning methods without significant sacrifice in accuracy. It is also found that the in-memory building block index saves a significant amount of memory space to store the history in the main memory. Thus, it can effectively serve real traffic management systems


(a) Building blocks in Partition 1


(b) Building blocks in Partition 2


(c) Building blocks in Partition 3

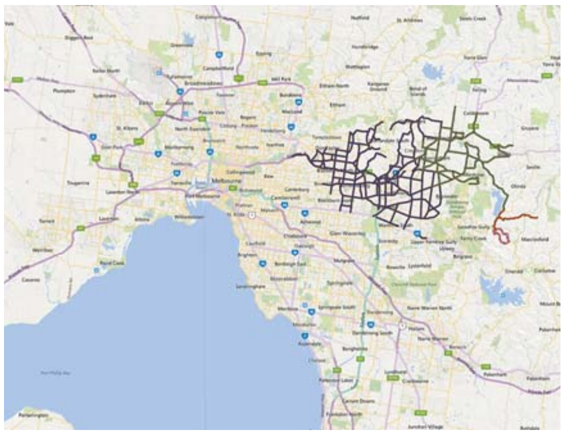

(d) Building blocks in Partition 4

FIGURE 5.11: Building blocks in the Physical layer at 07:09 AM
.

for continuously capturing the evolution of congestion and aid in smart transportation services.

Chapter 6

Applications using Real Traffic Data

The real road traffic networks experience different kinds of situations at different times of the day. The traffic pattern keeps varying depending on specific factors. For example, school going time and office going time may lead to different traffic patterns. This kind of patterns are not always easy to notice, without an experimental study. Therefore analyzing real traffic data in light of specific applications is very important. In this chapter, we present an experimental study to discover the congestion propagation patterns. For this, we considered two specific applications. First one is the temporal tracking of congested partitions, and the second one is the traffic diffusion and influence estimation. The first application of temporal tracking of congested partitions is related to our work presented in Chapter 5, but our objective here is to analyze the real data to discover the unknown traffic propagation patterns. We define certain traffic measures (e.g. volume and green time utilization) available from the traffic signal control system to identify the congested partitions, and track their evolution with time. We conducted experiments using real historical traffic data collected from the 493 signalized traffic sites in Melbourne (Australia) with a total of 1444 road segments and 581 intersection points. Our experimental results show that the large-scale urban traffic networks undergo many rapid but regular and frequent traffic patterns, which often go unnoticed by the traffic network operators. Tracking this kind of changes in real-time by means of our framework can improve the reaction time of the traffic management team resulting in less congestion. The second application of traffic diffusion and influence estimation is on the way traffic diffuses from one road segment to another and thereby keeps influencing the traffic of connected road segments. Traffic congestions generally originate from some crowded road segments, and diffuse towards other parts of the urban road networks creating further congestions. This behavior of road networks motivates the need to understand the influence of individual road segments on others in terms of congestion. In this work, we propose `RoadRank`, an algorithm to compute the influence scores of each road segment in an urban road network, and rank them based on their overall influence. It is an incremental algorithm that keeps on updating the influence scores with time, by feeding with the latest traffic data at each time point. The method starts with constructing a directed graph called *influence graph*, which is then used to iteratively compute the influence scores using probabilistic diffusion theory. We show promising preliminary experimental results on real SCATS traffic data of Melbourne.

6.1 Introduction

The developed urban areas these days are well equipped with urban traffic control systems (UTCS) like Sydney coordinated adaptive traffic system (SCATS) and split cycle offset optimisation technique (SCOOT). These traffic management systems log the aggregated traffic movement data on each of the road segments in real-time. This data do not contain individual movement trajectories, rather they are different traffic measures indicating the traffic load on the different road segments at each point of time.

The Sydney Coordinated Adaptive Traffic System (SCATS)[1] is a fully adaptive urban traffic control system developed in Australia in 1970. This intelligent transport system requires a minimal manual intervention and thus saves substantial operational costs. It has been continually developed for over 40 years and sold to 27 countries, delivering real and measurable reductions in road travel times and delays. It manages the signal phases (cycle times, phase splits and offsets) of the traffic signals dynamically in real-time, based on the traffic data collected by the vehicle sensors (inductive loops) installed within road

[1]http://www.scats.com.au/

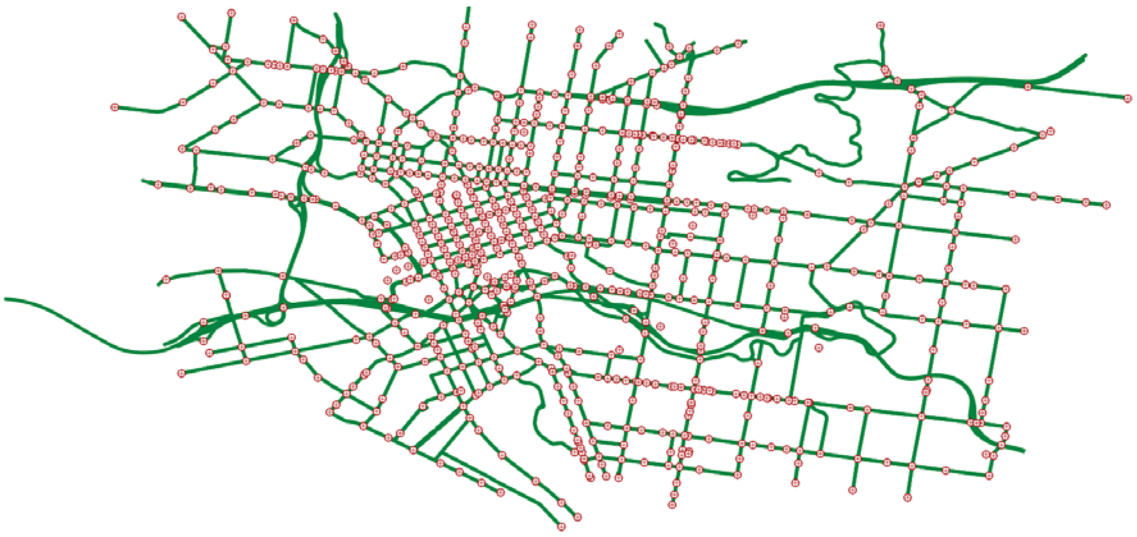


FIGURE 6.1: Melbourne Road Network

pavements of each traffic signal. The recorded traffic data is used by SCATS to calculate and adapt the timing of traffic signals in the network. A standard SCATS database contains records logged by loop detectors (having a detector identifier) corresponding to each lane (having lane number) of each specific road segment (having a road identifier) at each SCATS site (road intersection points operated by traffic signals). The overall measures corresponding to road segments are computed by referring to those of the lanes in them (having a site identifier). The traffic records include the following main fields.

- Traffic data (corresponding to detector id, lane id, road id, scats id, and time-stamp): traffic count (Vo), maximum traffic flow (MF), and corresponding vehicle occupancy (KP).
- Control data (corresponding to detector id, lane id, road id, scats id, and time-stamp): signal cycle time ($\boldsymbol{c}$), green time ($\boldsymbol{g}$, for each traffic movement) and information related to the offset between signals.
- The other important data calculated from the above data are the so called degree of saturation (DS), equivalent count (Vk), and volume ratio (VR). DS gives a measure of the density of traffic flow on a road segment and used in traffic control. It is defined in equation 6.1 as the ratio of the effectively used green time to the total available green time on the approach, where $\boldsymbol{T}$ is the total space (non-occupancy)

time, $\boldsymbol{t}$ is the average space time during saturated flow conditions, and $\boldsymbol{n}$ is the number of vehicles counted. Equation 6.1 may result into a DS of more than 100 %, which indicates an over-saturated traffic [138].

$$\boldsymbol{DS} = \frac{\boldsymbol{g - (T - n \times t)}}{\boldsymbol{g}} \tag{6.1}$$

Vk is defined in equation 6.2 as the estimated traffic count having the same DS under free flow conditions.

$$\boldsymbol{Vk = g \times DS \times MF} \tag{6.2}$$

VR is defined the ratio of the estimated traffic count to that in real (i.e., $\boldsymbol{\frac{Vk}{Vo}}$), and is considered as another indicator of traffic congestion. A value greater than 1 indicates an over-saturated traffic [138].

Theoretically DS alone should be able to identify the congestion in SCATS based traffic networks. A value of DS close to or greater than 1 (100 %) indicates a saturated or over-saturated traffic. However, [138] argued that since this value is continually modified within SCATS control algorithms, it no longer correctly demonstrates the state of congestion. On the other hand, Vo gives the actual number of vehicles crossing a road segment in a signal cycle. A high value of this measure indicates a free traffic flow, but a low value does not give a clear indication. It could mean that there are very few vehicles on the road and the traffic is very smooth, or the traffic is heavily congested and very few vehicles are able to cross in one signal cycle. Previous research shows that none of the fields in the SCATS traffic data give a direct indication of congestion independently.

VicRoads (aka Roads Corporation of Victoria) is the road and traffic authority in the state of Victoria, Australia. The operation of SCATS based traffic signals in Victoria are managed by VicRoads. Our database is an accumulation of the data collected from the multiple SCATS sites in Victoria over a period of time. As mentioned in [139], dealing with large traffic data is not an easy task. In many instances, the SCATS sensors fail and give inaccurate (or no) measurements. Sometimes the measures face unexpected interference of mediators. On many roads there exist no installed sensors. These factors altogether lead the SCATS sensors to introduce uncertainty into the system, and the collected data has chances of being poor in quality. There exist many roads for which no measures exist in the data. One major challenge is to handle this incompleteness of the sparse data. To overcome this, the data requires to go through some pre-processing and repairing tasks. Our database primarily consists of five tables interlinked by primary keys and foreign keys, which are SiteLayout, SiteList, SMData, SMDetectorInfo, and VolumeData. To recover the missing values for the roads that have no traffic measures in the database, we apply some pre-processing and repairing tasks. They are assigned an average of the corresponding measures of their adjacent roads. If their adjacent roads themselves do not have these measures, then firstly the measures of these roads (adjacent to the first one) are found, from their own adjacent roads (adjacent to the adjacent to the first one). It goes on recursively until such a road is encountered, which has its adjacent roads' traffic measures recorded in the database.

Urban roads exist in the form of a physical transportation network spatially spread over a large urban area. Figure 6.1 shows the Melbourne road network and the signalized intersection points used in SCATS. In the following definitions, we present the main components and the terminology of these networks.

Definition 6.1. (**Road**) A *road* is a generally used term to mean a publicly accessible way for transportation. It has no standard range for its length and can vary from a very short one to a very long. ■

Definition 6.2. (**Road Segment**) A *road segment* $\boldsymbol{r_i}$ is defined as the smallest unit of a road having its traffic flow in a single direction. Thus a *road* is composed of one or more *road segments*, where the two opposite directions of traffic flow form different road segments. ■

Definition 6.3. (**Intersection Point**) An *intersection point* (or road intersection point) $\boldsymbol{\iota_i}$ is defined as the point connecting two or more road segments. ■

Definition 6.4. (**Road Network**) A road network is defined as $\boldsymbol{\mathcal{N} = (\mathcal{I}, \mathcal{R})}$ comprising a set of road intersection points $\boldsymbol{\mathcal{I} = \{\iota_1, \iota_2, \ldots, \iota_{n_\iota}\}}$ as nodes that are connected among themselves by the set of directed road segments $\boldsymbol{\mathcal{R} = \{r_1, r_2, \ldots, r_{n_r}\}}$ as its links, where each road segment $\boldsymbol{r_i}$ associates a set of feature values with itself, including the different traffic measures $\boldsymbol{r_i.f}$. ■

In this chapter, we present two important applications on road traffic networks. The first one is the temporal tracking of congested partitions, in which we define certain traffic measures (e.g. volume and green time utilization) available from the traffic signal control system to identify the congested partitions, and track their evolution with time. The second application, we propose an algorithm (referred as `RoadRank`) to compute the influence scores of each road segment in an urban road network, and rank them based on their overall influence.

The rest of the chapter is organized as follows. It starts with the first application of temporal tracking of the congested partitionsin Section 6.2, followed by the second application of traffic diffusion and influence estimation in Section 6.3. Finally Section 6.4 presents a summary of the chapter.

6.2 Temporal Tracking of Congested Partitions in Dynamic Urban Road Networks

Most of the people living in urban areas are daily or frequent travelers within the urban road network. Traffic congestion is indeed a practical issue. The currently operating traffic management systems, such as SCATS and SCOOT log the traffic data continuously in real time. Based on those data, the systems adaptively control the traffic flow. To cover our journey in minimum possible time and avoid any unexpected circumstances, journey planning services on the Internet and mobile applications are of great use. The Google Traffic service on Google Maps visualizes the traffic condition using simple heat map in real time by anonymously collecting GPS data from a large number of mobile phone users. However, further efficiency improvement in the traffic management and advanced traffic visualization techniques are necessary to meet the growing traffic demand.

As discussed in the previous chapters, the traffic flow patterns vary significantly in different sub-networks, greatly influenced by their *spatial* and *temporal* importance. Even though we are accumulating huge amounts of traffic data these days through various sources, not much work have been done on studying the pattern of movement of congestion and its evolution with time. In this work, we develop a simple framework to monitor the evolution of traffic congestion in real time. It firstly identifies a set of differently congested road network partitions in the beginning, incrementally updates them at each subsequent time points in real time, identifies the congested partitions, and tracks their evolution with time.

The main contributions made in this application are summarized below.

– We propose a framework for identifying and monitoring the congested partitions over a period of time, that considers several key parameters in SCATS data to identify the congestion.

– Our method to incrementally update the identified partitions without re-partitioning the network each time is much more efficient for large scale urban road networks. This is achieved by defining a stability measure of road segments that is used to prioritize the processing and boundary adjustment of relatively unstable road segments.

– We identify and track the evolution of congested partitions using different visualization schemes and statistical measures.

– Our experimental study based on SCATS data give insights to the real traffic congestion scenario. We found that congestion in Melbourne and its pattern can change quickly over the course of as short as 3 minutes during peak hours. Tracking these changes by means of our framework can improve the reaction time of the traffic management team resulting in less congestion.

6.2.1 Proposed Congestion Monitoring Method

In the proposed framework, we consider congestion as a road network partition having a homogeneous level of congestion inside, which is higher than a given threshold, and call it as a *congested partition*. The level of congestion is defined based on key parameters (DS and VR) recorded in SCATS, detailed later. Monitoring such partitions over a period of time requires to firstly identify them and then track the change in them over a period of time. Our method for identifying and monitoring these congested partitions, shown in Figure 6.2, consists of three main tasks. It starts with transforming the given large

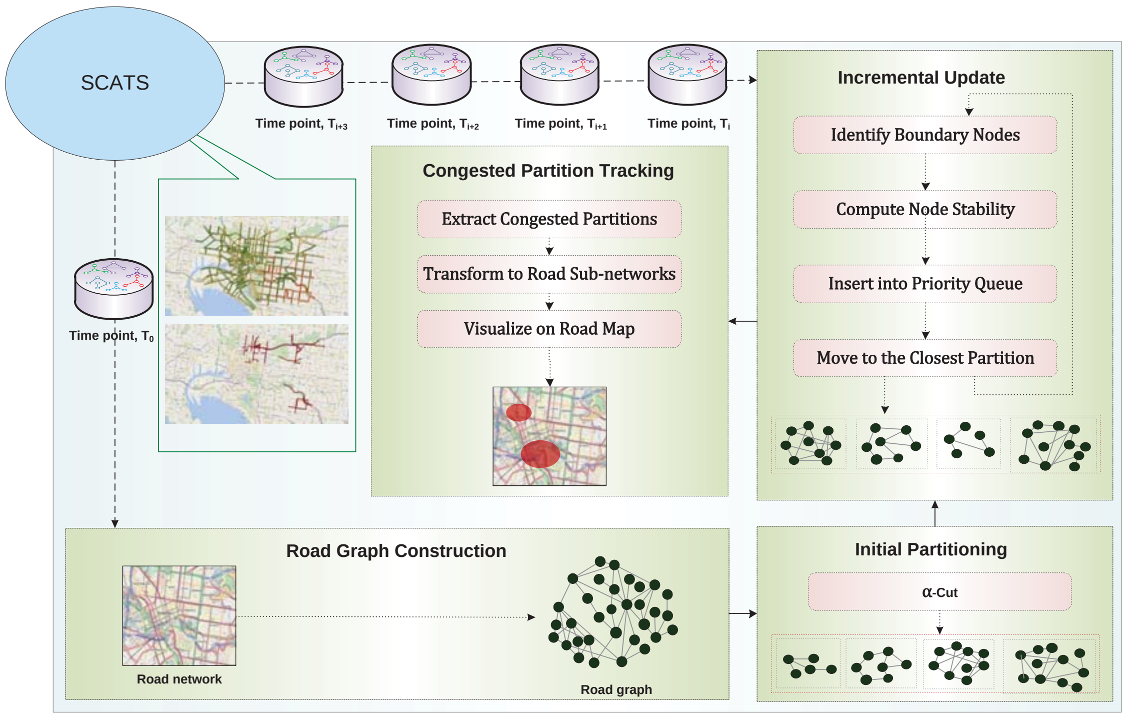


Figure 6.2: Proposed congestion monitoring method based on real-time traffic data from SCATS

urban road network into a road graph as a preliminary step. Then the first task identifies the differently congested partitions in the road graph based on the traffic data collected from SCATS. We approximate the freeway traffic with the SCATS data collected at the intersections next to on-ramp location, which serve as an indicator for the congestion of the freeway section between those on-ramps (note that Melbourne freeways are regulated by ramp metering that aims to avoid congestion on the freeways and thus congestion will be observed at intersections around the proximity of freeways due to traffic overflow from the on-ramps). Though we consider only the SCATS data in this work, the method can be easily adopted to include more traffic data collected from different sources. The second task accepts the obtained set of partitions as its input and incrementally updates them in an efficient manner at each subsequent time point based on the newly collected traffic data in real time. The third task identifies the congested partitions from the complete set, and effectively presents them to the end user by their geographic visualization and statistical measures. The first task is performed only at the beginning time point, whereas the second and third tasks are a continuous process performed indefinitely to track the change in congestion in terms of its size and location happening in real time. The first and second tasks consider both congested and non-congested partitions and sets the base of the system, whereas the third task focuses only on the congested partitions. The following subsections describe these tasks in detail.

6.2.1.1 Partitioning Road Network in the Beginning

Definition 6.5. (**Road Graph**) Given a road network $\mathcal{N}$, the corresponding road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed as the dual of $\mathcal{N}$ by adding each road segment r_i as a node v_i, and establishing a link e_i between each possible pair of nodes (v_j, v_k) if there exist at least one intersection point ι_l which is a common intersection for the roads r_j and r_k in $\mathcal{N}$, and the traffic can flow either from r_j to r_k or vice versa. Each node v_i $(node(r_i)) \in \mathcal{V}$ associates with it a feature value $v_i.f$ which is the road traffic measure $r_i.f$. ■

The partitioning task accepts the road network and the feature values, DS and VR, corresponding to each road segment in the network at the beginning time point t_0 as input, and partitions the network based on road segment connectivities (i.e., the connectivity of one road segment to other subsequent and preceding segments through the intersection points) and similarity of their feature values. It goes through two main steps, i) transforms the real road network into the road graph; and ii) applies a partitioning algorithm on the road graph based on DS and VR. After the construction of road graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ following definition 6.5, all the road segments r_i which exist as nodes v_i in the graph are assigned their feature vector $v_i.f = \langle DS(r_i), VR(r_i)\rangle$ consisting of DS and VR as feature values at time t_0. Each of the links in $l_i \in \mathcal{E}$ between the nodes v_p and v_q are assigned a weight ω_i using Equations 6.3 and 6.4, where $\sigma^2(v)$ is the variance of node feature vectors with respect to the mean vector $\mu = \langle \mu^{ds}, \mu^{vr}\rangle$ of DS mean and VR mean. It makes the road graph a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$.

$$\omega_i = \exp\left(\frac{-(v_p.f - v_q.f)^2}{2 \times \sigma^2(v)}\right) \tag{6.3}$$

$$\sigma^2(v) = \frac{1}{|\mathcal{V}|} \times \sum_{i=1}^{|\mathcal{V}|} (v_i.f - \mu)^2 \tag{6.4}$$

Throughout the process in this work, the network undergoes transformation twice. At the beginning, the given actual road network is transformed into the road graph representation to apply the algorithm. While it is simple and easier to mathematically process the network in this representation, it is very hard to visually interpret the obtained results in this form. Therefore after treating the road graph with the proposed method to get congested partitions, it is transformed back again into its original representation form.

After construction of the road graph $\mathcal{G}$, it is subjected to a partitioning algorithm based on α-Cut [3] (proposed in Chapter 3) to obtain a set of graph partitions or subgraphs, each of which are expected to have homogeneous feature values inside them, and heterogeneous to those in other partitions. The obtained partitions are a set of differently congested partitions (high as well as low) based on DS and VR.

6.2.1.2 Incremental Update of Partitions with Time

The previous section provides the k differently congested partitions at the beginning time point t_0. In this section, we fix the number of partitions to k and track their evolution by adjusting the boundary nodes and incrementally updating the partitions based on the newly obtained traffic data in real-time. It starts with computing the mean VR $\mu^1_{vr}(\mathcal{P}_j)$ and mean DS $\mu^1_{ds}(\mathcal{P}_j)$ inside all the partitions $\mathcal{P}_j$ individually from the traffic data obtained at the next time point t_1. The VR ($v_i.vr$) and DS ($v_i.ds$) values of nodes lying on boundary of the partitions are compared with $\mu^1_{vr}(\mathcal{P}_j)$ and $\mu^1_{ds}(\mathcal{P}_j)$ of the current partition and also the neighboring partitions. They are moved to the partition whose mean values are closest to their own feature values. Comparing the node feature values with the partitions' mean values for all the nodes becomes a tedious and computationally expensive task. To efficiently do this in real-time, we define a novel measure called *stability* of a node.

Definition 6.6. (**Node Stability**) If a node v_i belongs to partitions $\mathcal{P}_j$ at time t_r, then the stability $stab(v_i)$ is defined as the likelihood of v_i to remain in the same partition $\mathcal{P}_j$ at time t_{r+1}. To compute its measure we consider two of its kinds- i)*spatial* stability, which looks into how well the feature values of v_i match with those of the rest in $\mathcal{P}_j$ at time t_{r+1}; and ii) *temporal* stability, which looks into how much stable v_i was in the previous time points. ■

$$stab^{(r+1)}(v_i) = \frac{1}{2} \times \left(stab^{(r)}(v_i) + \exp\left(\frac{-1}{2} \times \left(\text{abs}\left(\frac{v_i.vr + 1}{\mu_{vr}^{(r+1)}(\mathcal{P}_j) + 1} - 1 \right) + \text{abs}\left(\frac{v_i.ds + 1}{\mu_{ds}^{(r+1)}(\mathcal{P}_j) + 1} - 1 \right) \right) \right) \right) \tag{6.5}$$

The first part $stab^{(r)}(v_i)$ coming from the previous time point r stands for the temporal stability, and the remaining part stands for the spatial stability. The spatial stability is based on the distance of the node from its partition centroid by ratio of their feature values (VR and DS) to partition mean values. We maintain all the boundary nodes in a heap tree based priority queue. Firstly, the stability measure is computed for all the boundary nodes, based on which they are inserted into the queue. The node with least stability measure is given the highest priority, and this order of stability and priority is maintained for all in the queue. One after another, a node v_i with highest priority is picked from the queue, and its feature vector $v_i.\langle vr, ds \rangle$ is compared with the partition mean vector $\langle \mu_{vr}^r(\mathcal{P}_j), \mu_{ds}^r(\mathcal{P}_j) \rangle$ using Euclidean distance, $\forall j$ such that $\mathcal{P}_j \in \{$current partition(v_i), neighboring partition$(v_i)\}$. It is then removed from the queue and allotted to the closest partition based on the comparison. Among the nodes linked to v_i, all those that became new boundary nodes are inserted into the queue, and all those that were on boundary earlier but not now after this change are removed from the queue. This process of getting a node from the queue, allotting to a partition, and updating the queue is carried out until $stab^{(r)}(v_i) < \epsilon_{stab}$, where ϵ_{stab} is a predefined threshold for stability comparison. All the stable boundary nodes (having their stability measure greater than the threshold) are assumed that they already belong to the correct partition, and therefore are not checked. This task of incrementally updating the partitions is a continuous process, performed at each time point when new traffic data arrives into the system.

6.2.1.3 Identification of Congested Partitions and their Tracking

In a road network, the road segments often have hierarchy, where big roads are connected by small roads. Some roads may experience heavy congestion while other connected ones may have no congestion at a point of time. As connectivity is an important factor in partitioning the network, it may also happen that a road with very low congestion lies in a partition that has a high congestion overall, because of the reason that it is surrounded by roads with high congestion. This property leads all the partitions to have roads with both high and low congestion inside, but their overall level of congestion in the subnetwork are different and unique. Based on this property, we define the congestion measures below.

Definition 6.7. (**Congested Road Segment**) A road segment $\boldsymbol{r_i}$ is said to be *congested*, if its VR and DS measures are greater than certain thresholds, i.e., $\boldsymbol{r_i.vr > \epsilon_{vr}}$ and $\boldsymbol{r_i.ds > \epsilon_{ds}}$. ■

Definition 6.8. (**Congestion Ratio**) The *congestion ratio* (CR) of a partition $\boldsymbol{\mathcal{P}_i}$ is defined as the ratio of the number of congested road segments to their total number in $\boldsymbol{\mathcal{P}_i}$. ■

Definition 6.9. (**Congested Partition**) A partition $\boldsymbol{\mathcal{P}_i}$ is said to be *congested* if at least half of its road segments are congested. It also means that the congestion ratio of $\boldsymbol{\mathcal{P}_i}$ has to be at least 0.5 to call it as congested. ■

While all the differently congested partitions have their own significance in the road network from different perspectives, those having high level congestion are of particular interest to the travelers and traffic management authorities. This task identifies the congested partitions based on Definition 6.9 and selectively tracks the change in their structure and movement with time. As the representation of these partitions is not in the form the actual network topology, they are transformed back into their original representation in the last step.

The tracking of the congested partitions over the period of time is done in two ways, $\boldsymbol{i}$) through geographic visualization, and $\boldsymbol{ii}$) through statistical measures. The visualization displays the congestion on geographic map that keeps on updating with time to reflect the latest scenario. It gives a way to visually analyze the congestion in real time for the traffic management authorities as well as the commuters. We found three main visualization schemes that give a good understanding of the traffic scenario.

- Simple heat map: All the road segments are assigned their feature values, and their strength is visualized by using color codes on a geographic map.
- Partition visualization: All the differently congested partitions are visualized on a geographic map by using light and dark color codes to represent the different levels of congestion in them. It is updated at each new time point to reflect the latest scenario, and the change in their structure and location can be seen clearly with time. As our method takes into account both the volume and degree of saturation data in identifying the congested partitions, this scheme gives a summarized view of congestion altogether on the network, and makes its analysis easy to interpret.
- Congested partition visualization: It focuses only on the congested partitions, and ignores the remaining sections of the road network. The main advantage of this scheme is that it helps in understanding how the congestion grows, shrinks, and moves over a period of time.

We also compute some congestion related statistical measures and show the change in them over the period of time in the form of charts.

- Congestion size in terms of road segments (CSR): It is defined as the number of road segments lying in the set of congested partitions. It gives the information about how big is the congestion, but do not tell whether the whole congestion is interdependent or not.
- Congestion size in terms of partitions (CSP): It is defined as the number of congested partitions. A large CSP indicates that there exists several independent congestions in different portions of the network.
- Congestion Magnitude (CM): It is defined as the average of CR of the congested partitions, and gives the information about how much congested are the congested partitions at a point of time.

- Average congestion ratio (ACR): It is defined as the average of CR of all the partitions irrespective of whether they are congested or not, and gives the information how much is the complete road network congested at a point of time.

6.2.2 Experimental Results

We performed experiments using SCATS data collected from the Melbourne road networks provided to us by VicRoads[2]. This dataset is an accumulation of the traffic records of individual road segments for each signal cycle from 1st Jan 2011 to 1st Jan 2013, and stored as a database of five interlinked tables on a local server. As mentioned in Section 6.1, many road segments have their traffic measures missing, for which we applied a data repairing technique to regain those measures. The considered Melbourne sub-network consists of 1444 road segments and 581 intersection points, where the traffic measures are logged by 493 SCATS sites.

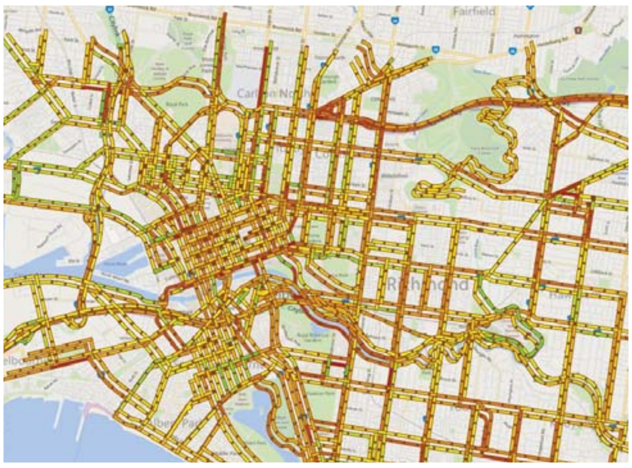
6.2.2.1 Simple Heatmap Visualization

Figure 6.3 shows the heat map based on Vo, VR and DS at 08:30 AM on 03-12-12 (Monday) in 6.3(b), 6.3(c), and 6.3(a) respectively. The color ranges from light green to dark red, where the darkness indicates a higher value. Though it helps to visually interpret those values at a particular time point, but as they change with time, it becomes very hard to perceive and track the change, as is evident from the figure. It urges for a more informative and effective visualization. Therefore we apply our proposed method on the SCATS data in the next section.

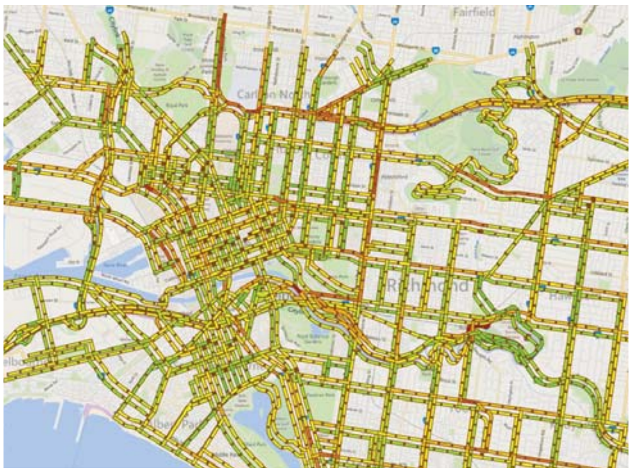
6.2.2.2 Visualization Comparison with Google Traffic

Congestions in the network can be simply (but not necessary effectively) visualized as a heat map using individual parameters. Figure 6.4(a) shows the `Google Traffic` visualization of a typical Monday traffic at 11:00 AM in Melbourne, which is a simple heat map.

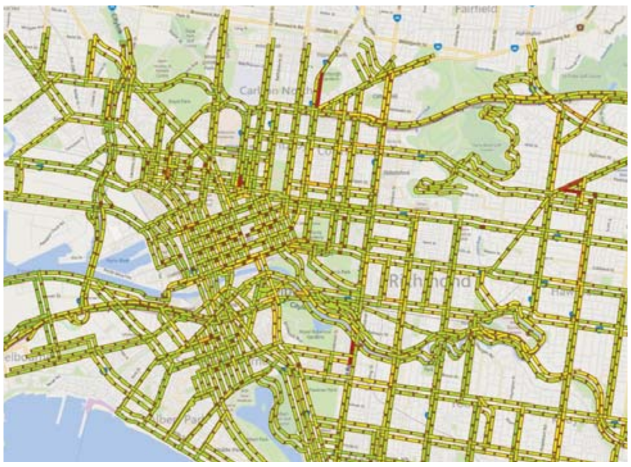
[2]http://www.vicroads.vic.gov.au/



(a) Heat map generated using DS at 08:30 AM

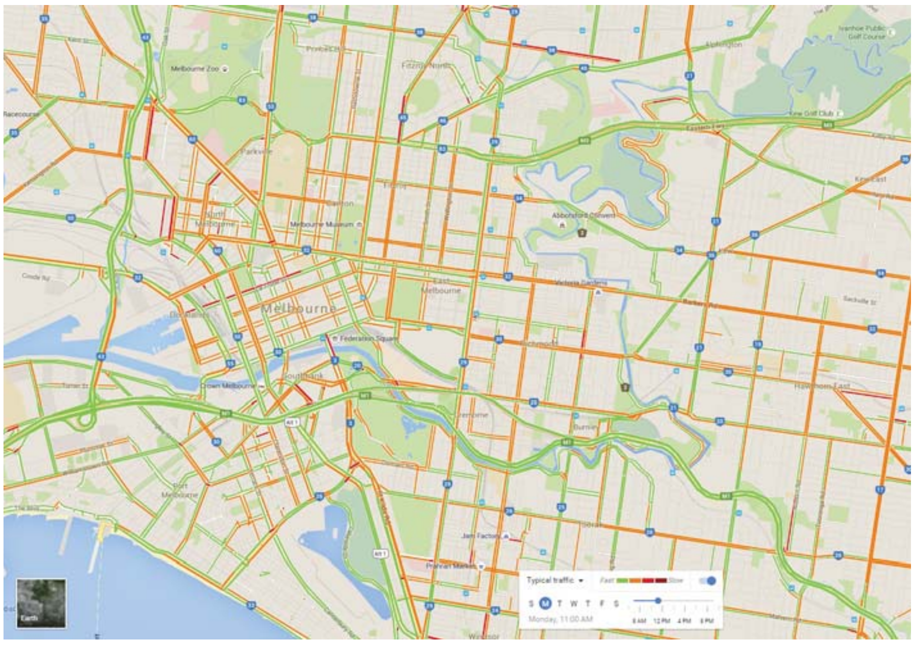


(b) Heat map generated using Vo at 08:30 AM

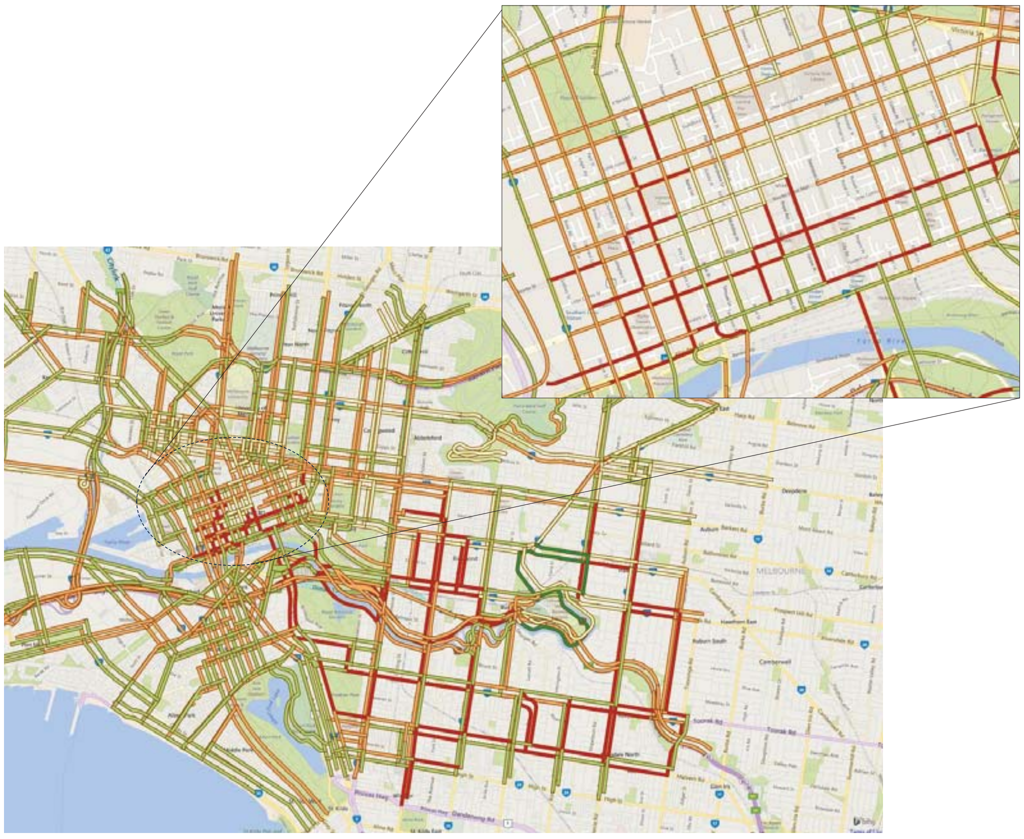


(c) Heat map generated using VR at 08:30 AM

Figure 6.3: Visualization of traffic data



(a) `Google Traffic` visualization of at 11:00 AM (typical, Monday)



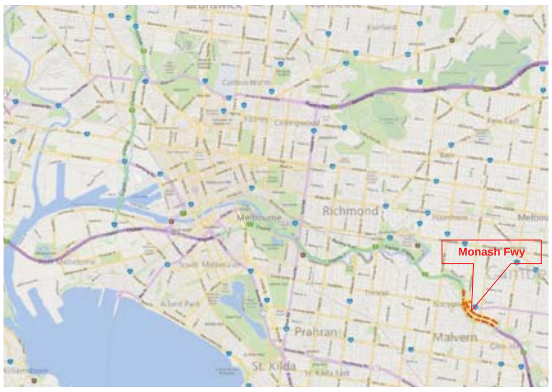
(b) Partition visualization by our framework at 11:00 AM (03-12-12, Monday)

Figure 6.4: Visualization of traffic data

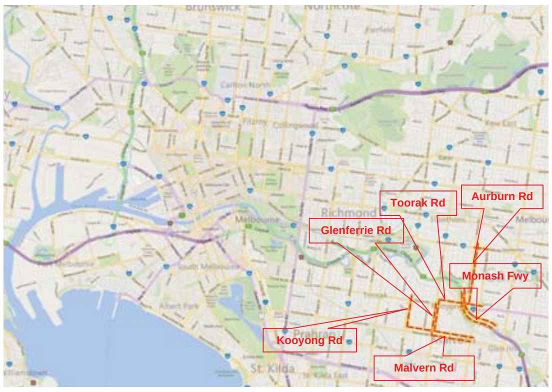
A commuter can see which of the roads are congested at a point of time. The same colored road segments are not necessarily connected, which does not give information about how the congestion is actually spread or moving. Figure 6.4(b) shows a snapshot of the visualization of all the differently congested partitions by our framework at 11:00 AM on 03-12-12 (Monday). We run the $\boldsymbol{\alpha}$-Cut algorithm in the beginning for initial partitioning and run the incremental update algorithm after each 3 minute intervals of time. The two directions of traffic flow of the partitions are shown separately side by side in the figure. All the 5 partitions are shown in different colors ranging from green (indicating low congestion) to red (indicating high congestion). Comparing the two figures, there may be difference in the level of congestion (color) shown for particular road segments because the underlying traffic data for the two methods is not same. Our aim here is to make the visualization informative and effective. Each partition in our visualization is a sub-network (connected from inside) and represents a certain level traffic congestion shown by the color. It shows how the different sub-networks are spread across the network and gives a more informative high level view of the congestion. The zoom in and zoom out features give further information regarding how the sub-networks are spread or connected.

6.2.2.3 Congested Partition Identification

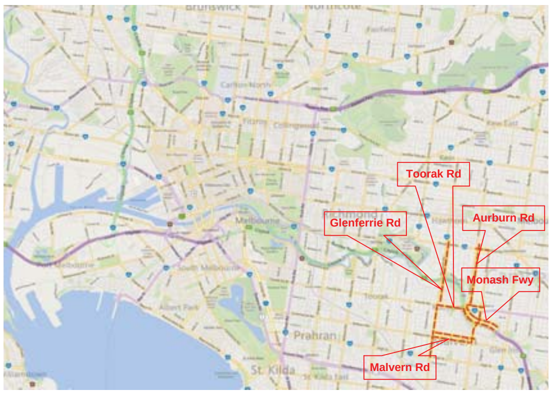
Generally the regions or road segments where the congestion originates and comes to end, the path or direction of congestion movement, and the rate in which the congestion grows and shrinks are of keen interest to the traffic management authorities. Our third form of visualization sets a threshold to accept a partition as congested, and shows only those congested partitions for a close look with the aim to discover the mentioned congestion evolution patterns. Based on our experimental study and an existing work [138], we set the thresholds $\boldsymbol{\epsilon_{stab}}$, $\boldsymbol{\epsilon_{ds}}$ and $\boldsymbol{\epsilon_{vr}}$ to 0.99, 0.7 and 1.1, respectively, and obtain the congested partitions. The number and size of these congested partitions vary from time to time. Figure 6.5 shows snapshots of the congested partitions (in red color) at different times in the morning of 03-12-12 (Monday). We found that congestion starts building up in early morning and the first to face is the area around the intersection of Monash Freeway and Toorak Rd at 07:21 AM, as shown in Figure 6.5(a). It starts spreading to neighboring road segments at 07:39 AM (Figure 6.5(b)). At 07:45 AM the congestion



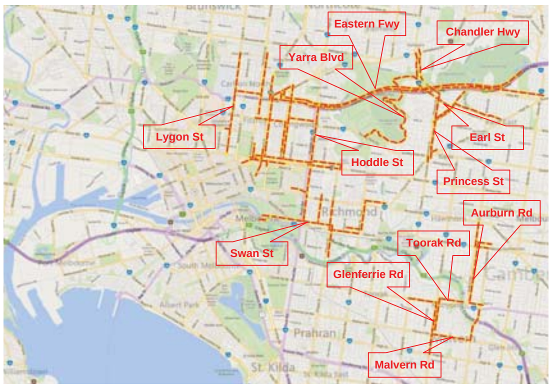
(a) At 07:21 AM



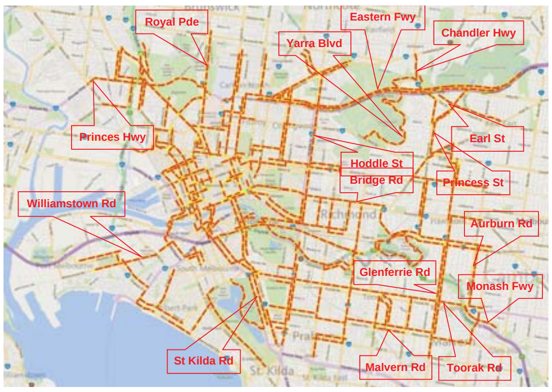
(b) At 07:39 AM



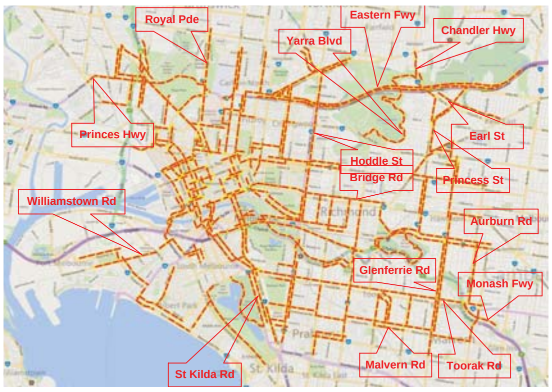
(c) At 07:45 AM



(d) At 07:51 AM

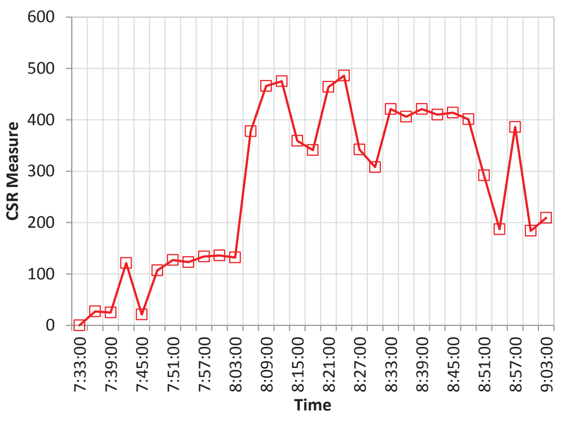


(e) At 08:06 AM

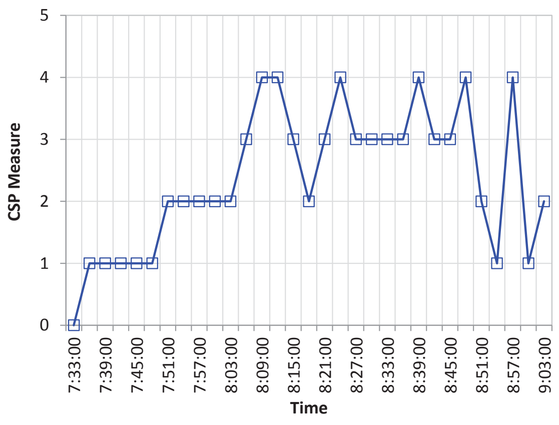


(f) At 08:24 AM

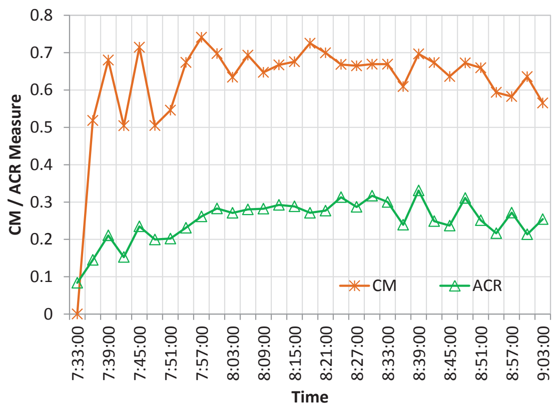
FIGURE 6.5: Congested partitions obtained by the proposed method



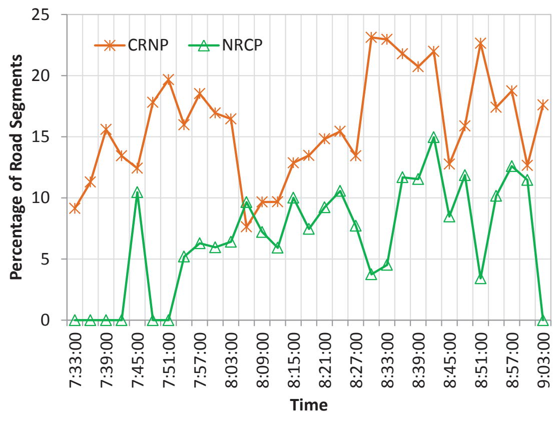
(a) CSR



(b) CSP



(c) CM and ACR



(d) Accuracy of the proposed framework

Figure 6.6: Congestion statistics

moves in anti-clockwise direction on Toorak Rd, Glenferrie Rd, Malvern Rd, Tooronga Rd, and the Monash Freeway, and forms a round (Figure 6.5(c)). The other congestion hotspot in the early morning is the Eastern Fwy that is used by the people from far east to enter towards the city (Figure 6.5(d)). Though the congestion starts a little later on this freeway, the big size of this partition indicates its bigger effect on the whole network. The congestion spreads around the intersection of Chandler Hwy and Eastern Fwy to connecting roads like Princess St and Earl St. We see that people also try to avoid the freeway congestion by taking early exits (at Princess St) or going through the Yarra Blvd, and in turn they also become affected. Moving towards the west, after the intersection of the freeway with Hoddle St, the congestion spreads all around in the city, and starts growing rapidly. During this time the change in the size and location of congestion is very frequent. We observe that at 08:03 AM the congestion on intersection of Monash Freeway and Toorak Rd also starts growing, and meets the others at 08:06 AM (Figure 6.5(e)). At 08:24 AM the congestion spreads on all the main roads all over the network, especially the CBD area (Figure 6.5(f)). In our experiments we found that the level and the size of congestion keeps on increasing until 08:45 AM, then decreases gradually till 09:00 AM, and decreases drastically after that. The reason could be that by that time most of the people might have reached to their destinations.

We present the statistical measures in Figure 6.6. Figure 6.6(a) shows the number of road segments affected by the congestion, in which we see that there is a rapid increase from 08:03 AM to 08:09 AM. Figure 6.6(b) shows the number of different congested partitions, which means that there exist multiple independent congestions that can be visualized as different layers individually. For example, it shows two partitions at 08:03 AM as two independent congestions. Figure 6.6(c) shows CM as the level of congestion in the congested partitions and ACR as that in the overall network. The values of both CM and ACR can range from a minimum of 0 when there is no congestion at all to 1 when the network is fully congested. We see that in the peak hours the CM reaches even more than 0.7. Also there is a big gap between CM and ACR. An optimization of the traffic flow would bring the CM down and result into minimization of this gap.

Figure 5(d) shows the percentage of congested road segments in non-congested partitions, and that of non-congested road segments in congested partitions denoted as CRNP and NRCP, respectively. Observe that the percentages are under 23% for CRNP and 15% for NRCP, which indicates a good accuracy in terms of partitioning performance (more discussions based on this measure see [3]).

The SCATS traffic data in our framework is collected and preprocessed in every 3 minutes interval. The computations take 92 seconds for the initial partitioning, approximately 1 second for each incremental update of partitions, and less than a second for identifying the congested partitions. Thus it takes approximately 2 seconds to visualize the updated congested partitions at each successive 3-minute interval. It shows the applicability of our system in the real scenario, where looking into the trend of congestion evolution in real time, the traffic management authorities can prioritize their attention on these congested partitions and alleviate any unexpected circumstances (e.g., accident). Also the commuters can have a good idea about the congestion spread and connectivity, and plan their travel accordingly.

6.3 Traffic Diffusion and Influence Estimation

Traffic congestion remains a big challenge in the 21st century due to the rapid growth of population and their mobility demand within the urban areas [3]. Congestion often starts in few confined places within the network and propagates through various connected road segments. The propagation of congestion is due to the diffusion (or movement) of traffic from one road segment to another where the amount of traffic on any particular road segment is *influenced by* and *influences* that of others. The level of influence one road segment can have on others in terms of congestion depends on their spatial and temporal attributes and is not the same across the network. Clearly, high influential road segments play a more important role in the building up of congestion and will need to be better managed in order to alleviate or delay the congestion onset.

In this application, we propose our novel algorithm named `RoadRank` that computes the influence scores (called *roadrank scores*) of each road segment in an urban road network. Figure 6.7 shows the work flow of the complete method. To deal with the dynamic nature of traffic on a road network, it also updates the scores incrementally with time, by feeding in the latest traffic measures at each timestamp. The method starts with constructing a directed graph called *influence graph*. It is used to compute the traffic diffusion from one road segment to another, based on the collected traffic measures. Finally the *roadrank* scores are iteratively computed using probabilistic diffusion theory for the starting point of time. At each successive timestamps, they are iteratively updated based on the new traffic measures. We also show preliminary experimental results on real SCATS traffic data of Melbourne.

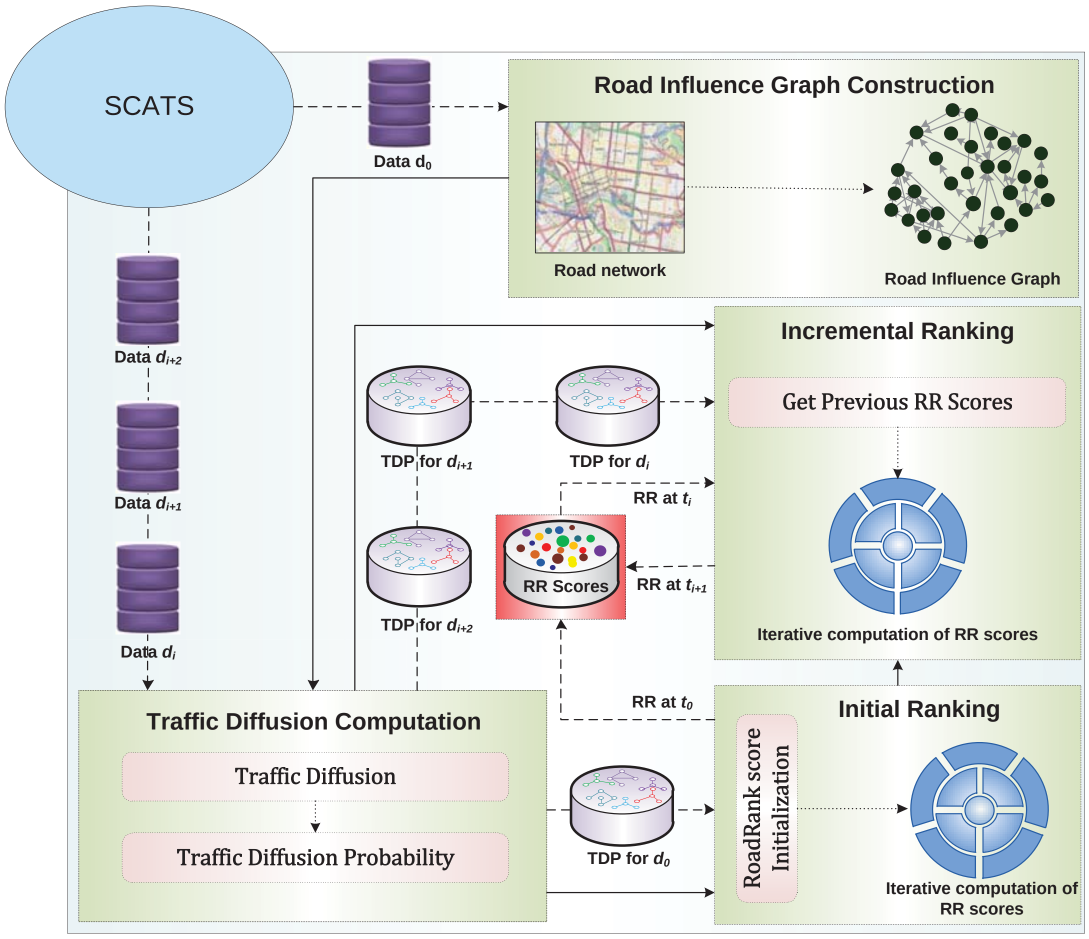
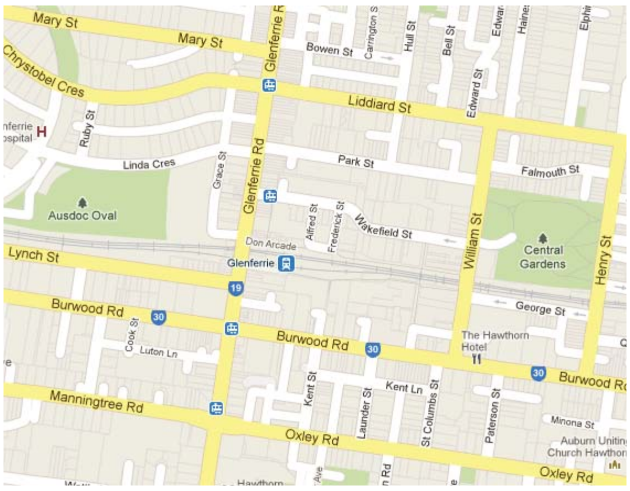


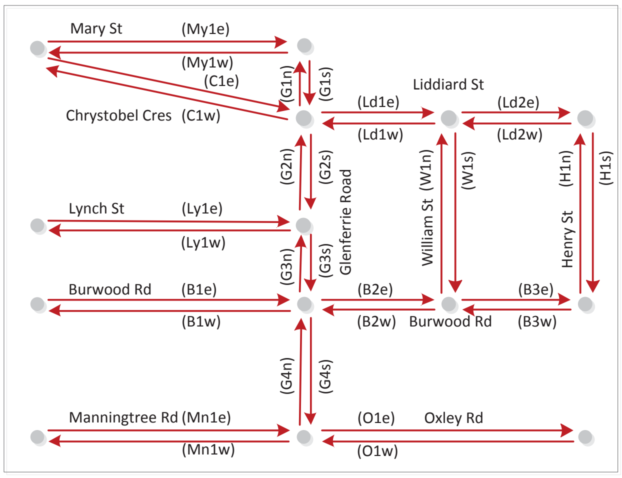
FIGURE 6.7: Proposed method for ranking influential road segments

6.3.1 Problem definition

Let us suppose we have a given urban road network $\mathcal{N}$ having all recorded historical traffic data $\mathcal{D}$ corresponding to each of the road segments in the network. Let $\mathcal{T} = \langle t_0, t_1, \ldots, t_{l-1}, t_l \rangle$ be the timestamps of the recorded data, where t_0 is the very first record, and t_l is the latest record that keeps on updating with time. Thus the dataset $\mathcal{D} = \langle d_0, d_1, \ldots, d_{l-1}, d_l \rangle$ becomes an incrementally updating vector, where each d_i is the traffic data recorded at time t_i. The problem of estimating the influence of road segments is to incrementally compute an *influence score* for each road segment r_i in $\mathcal{N}$ corresponding to each d_i (at t_i) and rank them based on this measure. The influence score (also called roadrank score to be defined in Section 6.3.2.3) of a road segment r_i gives a



(a) Actual road map



(b) Road network

FIGURE 6.8: An example of a road network

measure of how much the traffic on r_i influences that on the global network $\mathcal{N}$ because of the traffic diffusion going on via its linked road segments.

6.3.2 RoadRank Algorithm

6.3.2.1 Road Influence Graph Construction

The method starts with constructing a *road influence graph* $\mathcal{G}^{inf}$ from the road network $\mathcal{N}$, having the road segments as nodes and the directed edges between them represent the

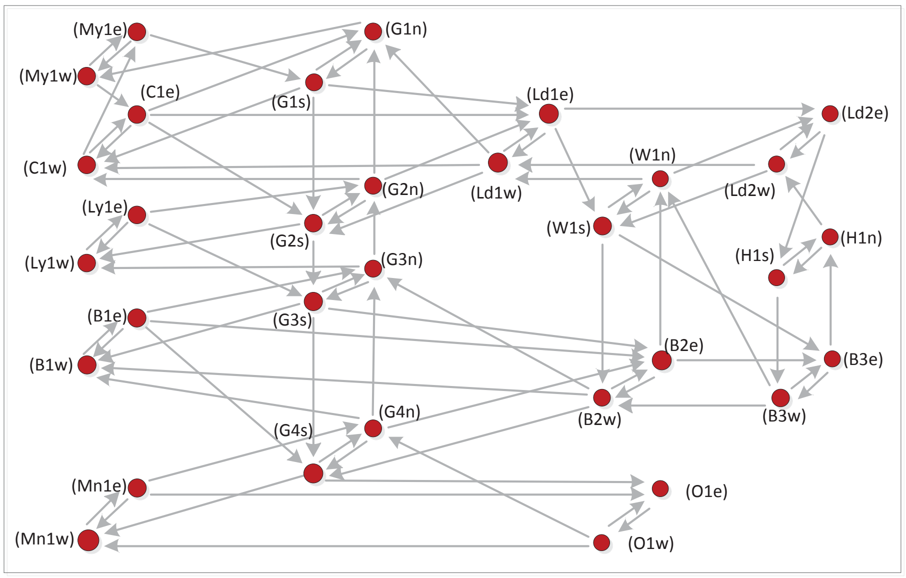


FIGURE 6.9: Road Influence graph

influences relationship.

Definition 6.10. (**Road Influence Graph**) Given a road network $\mathcal{N} = (\mathcal{I}, \mathcal{R})$, the corresponding road influence graph $\mathcal{G}^{inf} = (\mathcal{V}, \mathcal{E})$ is constructed by adding each road segment $r_i \in \mathcal{R}$ as a node $v_i \in \mathcal{V}$, and establishing a directed link $e_i \in \mathcal{E}$ from v_j to v_k if these two conditions hold- i) there exist at least one intersection point ι_l that is a common intersection for the roads r_j and r_k, and ii) the direction of traffic flow leads the movement from r_j to r_k. The directed link e_i shows that the traffic on road segment r_j (or node v_j) influences the traffic on road segment r_k (or node v_k). ■

We assume that the traffic rule allows a moving vehicle to take all the turns including left-, right-, and U-turns at intersection points on all the roads in the network (though in real there exist restrictions). In Figure 6.8, we have shown an example of road networks, where only the yellow major roads from Figure 6.8(a) are considered[3] to show the equivalent road network. The influence graph is constructed from this road network as shown in Figure 6.9. A traversal from one node to another in the influence graph gives the path of the traffic flow as well as the traffic diffusion process. One such example is $(My1e) \xrightarrow{inf} (G1s) \xrightarrow{inf} (G2s) \xrightarrow{inf} (G3s) \xrightarrow{inf} (B2e) \xrightarrow{inf} (B3e) \xrightarrow{inf} (B3w)$.

[3]To keep the example simple (all the roads including small ones are considered in real)

6.3.2.2 Traffic Diffusion Computation

Information diffusion is a widely studied research area in social media, where the information propagates from one source to another. In road traffic networks, there is a physical movement of the traffic from one road segment to another. We model this movement as a traffic diffusion process, in which the traffic diffuses among differently concentrated road segments through their connectivities. The directed links in the influence graph constructed in Section 6.3.2.1 show the direction and order of movement of this traffic. To find the exact amount of traffic that diffuse from one road segment to another in a directed link, we compute a measure based on their real time traffic measures.

Let $\boldsymbol{Vo} = \langle \boldsymbol{vo_1, vo_2, \ldots, vo_{n_r}} \rangle$ and $\boldsymbol{DS} = \langle \boldsymbol{ds_1, ds_2, \ldots, ds_{n_r}} \rangle$ be the vectors having the traffic volume and degree of saturation measures of all the road segments $\boldsymbol{r_i} \in \boldsymbol{\mathcal{R}}$. We compute two similarity matrices $\boldsymbol{S^{vo}}$ and $\boldsymbol{S^{ds}}$ for the $\boldsymbol{Vo}$ and $\boldsymbol{DS}$ vectors respectively such that $\boldsymbol{S^{vo}_{ij} = GSim(vo_i, vo_j)}$ and $\boldsymbol{S^{ds}_{ij} = GSim(ds_i, ds_j)}$. Equation 6.6 defines the Gaussian similarity measure $\boldsymbol{GSim(x_i, x_j)}$, where $\boldsymbol{\sigma^2 = \frac{1}{n} \times \sum_{i=1}^{n} (x_i - \mu)^2}$ is the variance of all values in the vector.

$$\boldsymbol{GSim(x_i, x_j) = \exp\left(\frac{-(x_i - x_j)^2}{2 \times \sigma^2}\right)} \tag{6.6}$$

Definition 6.11. (**Traffic Diffusion**) The *traffic diffusion* $\boldsymbol{td(r_i \rightarrow r_j)}$ is defined as the actual amount of traffic diffused from $\boldsymbol{r_i}$ to $\boldsymbol{r_j}$. We compute this measure by using Equation 6.7, where $\boldsymbol{\alpha_{ij}}$ is a balance factor called *traffic condition factor*, whose value is based on the real traffic conditions. ■

$$\boldsymbol{td(r_i \rightarrow r_j)} = \begin{cases} \boldsymbol{\alpha_{ij} \times S^{vo}_{ij} + (1 - \alpha_{ij}) \times S^{ds}_{ij},} & \text{if } \boldsymbol{(r_i \rightarrow r_j) \in \mathcal{E}} \\ \\ \boldsymbol{0,} \text{ otherwise} \end{cases} \tag{6.7}$$

The *traffic condition factor* $\boldsymbol{\alpha_{ij} \in [0, 1]}$ is a measure that quantifies the actual traffic condition on the roads. For example, let $\boldsymbol{r_i}$ and $\boldsymbol{r_j}$ be a narrow and a wide road segment respectively. When a certain number of vehicles move from $\boldsymbol{r_i}$ to $\boldsymbol{r_j}$, the $\boldsymbol{ds_i}$ will be greater than $\boldsymbol{ds_j}$. Thus even though all the vehicles from $\boldsymbol{r_i}$ move on to $\boldsymbol{r_j}$, their similarity in the $\boldsymbol{DS}$ measure becomes low, because of the difference in their width. The traffic condition factor captures this aspect of road segments to balance the $\boldsymbol{Vo}$ and $\boldsymbol{DS}$ similarity measures and aid in accurately computing the traffic diffusion. Setting this value to 1 makes Definition 6.11 behave as volume diffusion, where as 0 makes it behave as degree of saturation diffusion, and setting an appropriate value[4] in between them makes it behave as the traffic congestion diffusion.

Definition 6.12. (**Traffic Diffusion Probability**) The *traffic diffusion probability* $\boldsymbol{tdp(r_i \rightarrow r_j)}$ is defined as probability of the traffic that diffuse from $\boldsymbol{r_i}$ to $\boldsymbol{r_j}$. We compute this measure by using Equation 6.8, where $\boldsymbol{r_i \xrightarrow{inf} r_k}$ means $\boldsymbol{r_i}$ *influences* $\boldsymbol{r_j}$ in $\boldsymbol{\mathcal{G}^{inf}}$ (in other words, $\boldsymbol{r_i \rightarrow r_k \in \mathcal{E}}$). ■

$$\boldsymbol{tdp(r_i \rightarrow r_j)} = \frac{\boldsymbol{td(r_i \rightarrow r_j)}}{\sum\limits_{\forall k: r_i \xrightarrow{inf} r_k} \boldsymbol{td(r_i \rightarrow r_k)}} \tag{6.8}$$

6.3.2.3 Ranking

PageRank

The PageRank is one of the most popular ranking algorithms [101]. It computes a ranking of the webpages to estimate their importance to Web navigators. It initializes each of the pages with a small value as their page rank score ($\mathbf{PR}(\boldsymbol{p_i})$), and iteratively uses the linkages ($\boldsymbol{L}$) among them to compute their new page rank score ($\mathbf{PR}(\boldsymbol{p_j})$) using equation 6.9, where $\boldsymbol{d} \in [\mathbf{0}, \mathbf{1}]$ is the damping factor typically set to 0.85 [101], $\mathbf{prob}(\boldsymbol{p_j|p_i}) = \frac{\mathbf{1}}{\boldsymbol{out-degree(p_i)}}$ is the transition probability from webpage $\boldsymbol{p_i}$ to webpage $\boldsymbol{p_j}$, and $\boldsymbol{l_{ij}} \in \boldsymbol{L}$ is the hyperlink from page $\boldsymbol{p_i}$ to $\boldsymbol{p_j}$.

[4]It can be done based on experimental results and inputs from the experienced traffic management people.

$$\mathbf{PR}(\boldsymbol{p_j}) = (\mathbf{1} - \boldsymbol{d}) + \boldsymbol{d} \times \sum_{\forall p_i : l_{ij} \in L} \mathbf{prob}(\boldsymbol{p_j}|\boldsymbol{p_i}) \times \mathbf{PR}(\boldsymbol{p_i}) \tag{6.9}$$

In the road traffic network scenario, the top-k influential road segments can be identified by ranking and selecting the top-k of them. Moreover, as the road networks are dynamic in nature because of the frequently changing traffic, the ranking is time-sensitive, and needs to be updated with time based on the real-time traffic measures. Therefore we start with computing the ranking at timestamp $\boldsymbol{t_0}$, and keep on updating the ranking at each new timestamps $\boldsymbol{t_1, t_2, \dots t_l}$ to always have the ranking obtained from the latest collected traffic measures.

Ranking at $\boldsymbol{t_0}$: Our ranking algorithm borrows some concepts of the PageRank algorithm [101]. The traffic on road networks consists of vehicles moving on its road segments. In the proposed `RoadRank` algorithm, we consider that the traffic moving from one road segment $\boldsymbol{r_i}$ to another road segment $\boldsymbol{r_j}$ as the recommendation made by $\boldsymbol{r_i}$ to $\boldsymbol{r_j}$. Over a period of time, also new vehicles start and existing vehicles stop randomly on different road segments. Let $\boldsymbol{vpn}$ be the probability that a new random vehicle starts on any road segment in the given road network $\boldsymbol{\mathcal{N}}$ at timestamp $\boldsymbol{t_r}$, such that $(\mathbf{1} - \boldsymbol{vpn})$ gives the probability that the vehicle was already there in $\boldsymbol{\mathcal{N}}$. For the new random vehicle that started in $\boldsymbol{\mathcal{N}}$, let $\boldsymbol{vp(r_j)}$ be the probability that the vehicle started on $\boldsymbol{r_j}$, such that $(\mathbf{1} - \boldsymbol{vp(r_j)})$ gives the probability that the vehicle started on any other road segment $\boldsymbol{r_k} \in \{\boldsymbol{\mathcal{V}} - \boldsymbol{r_j}\}$. Thus $\boldsymbol{vpn} \times \boldsymbol{vp(r_j)}$ gives the probability that a new random vehicle started on $\boldsymbol{r_j}$, and $(\mathbf{1} - \boldsymbol{vpn} \times \boldsymbol{vp(r_j)})$ gives the probability that the vehicles currently on $\boldsymbol{r_j}$ were there on $\boldsymbol{\mathcal{N}}$ since before. We compute the roadrank score of a road segment $\boldsymbol{r_j}$ by considering and aggregating these two scenarios using Equation 6.10, where $\boldsymbol{RR(x)}$ denotes the updated score of $\boldsymbol{x}$ and $\boldsymbol{RR'(x)}$ denotes the score of $\boldsymbol{x}$ in the previous iteration. For the new traffic, it simply counts the probability $\boldsymbol{vpn} \times \boldsymbol{vp(r_j)}$, and for the existing traffic it counts the summation of all roadrank scores diffused from the neighboring road segments, multiplied by the probability $(\mathbf{1} - \boldsymbol{vpn} \times \boldsymbol{vp(r_j)})$. The diffused roadrank scores are computed as multiplication of the previous roadrank scores ($\boldsymbol{RR'(r_i)}$) of the roads influencing $\boldsymbol{r_j}$ with their traffic diffusion probability.

$$\begin{aligned} \boldsymbol{RR(r_j)} = & \quad \boldsymbol{f}(\text{new traffic}) + \boldsymbol{f}(\text{existing traffic}) \\ = & \quad \boldsymbol{vpn \times vp(r_j) + (1 - vpn \times vp(r_j))} \\ & \times \sum_{\forall i : r_i \xrightarrow{inf} r_j} \big(\boldsymbol{tdp(r_i \to r_j) \times RR'(r_i)}\big) \end{aligned} \tag{6.10}$$

To start with, the roadrank scores $\boldsymbol{RR(r_i)}$ for all the road segments are set to 1. Equation 6.10 is then used to iteratively update the $\boldsymbol{RR(r_j)}$ scores from the previously available $\boldsymbol{RR'(r_i)}$ scores. Each iteration updates $\boldsymbol{RR(r_j)}$ for all $\boldsymbol{r_j \in \mathcal{R}}$ using $\boldsymbol{RR'(r_i)}$, and the iterations are repeated until convergence is achieved.

Incremental ranking at each new timestamp $\boldsymbol{t_l}$: If we closely monitor the traffic on road networks in very short intervals of time, we will notice that the change in traffic in successive timestamps are very smooth. Therefore we can expect that in these short intervals of time there will be no big change also in the roadrank scores. Let $\boldsymbol{RR^0(r_j), RR^1(r_j), \ldots, RR^l(r_j)}$ denote the roadrank scores at timestamps $\boldsymbol{t_0, t_1, \ldots, t_l}$ respectively, based on the traffic datasets $\boldsymbol{d_0, d_1, \ldots, d_l}$ respectively. Similarly $\boldsymbol{vpn^l}$, $\boldsymbol{vp^l(r_j)}$, and $\boldsymbol{tdp^l(r_i \to r_j)}$ denote the values at timestamp $\boldsymbol{t_l}$ based on the dataset $\boldsymbol{d_l}$. To compute the score at the latest timestamp $\boldsymbol{t_l}$, the roadrank scores of all $\boldsymbol{r_i}$ are initialized as those computed from the previous timestamp $\boldsymbol{t_{l-1}}$, i.e., $\boldsymbol{RR'^l(r_i) = RR^{l-1}(r_i)}$. Equation 6.11 is then used to iteratively compute latest roadrank scores. The iterations are continued until it reaches the convergence.

$$\begin{aligned} \boldsymbol{RR^l(r_j)} = & \quad \boldsymbol{vpn^l \times vp^l(r_j) + (1 - vpn^l \times vp^l(r_j))} \\ & \times \sum_{\forall i : r_i \xrightarrow{inf} r_j} \Big(\boldsymbol{tdp^l(r_i \to r_j) \times RR'^l(r_i)}\Big) \end{aligned} \tag{6.11}$$

Thus at each new timestamp, instead of recomputing the roadrank scores by initializing them to 1, we incrementally update them. It significantly improves the performance by requiring comparatively fewer iterations and making the task much faster. In this way, the algorithm is able to maintain the roadrank scores with the real time data.

6.3.3 Experimental Results

To show the performance of `RoadRank`, we conducted experiments on real traffic data collected from the urban Melbourne road network. The considered Melbourne sub-network consists of 1444 road segments and 581 intersection points, where the traffic measures are logged by 493 SCATS sites. In the dataset, many road segments have their traffic measures missing because of faulty SCATS sensors. We regain those missing data by applying a data repairing technique. The traffic measures are recorded for each signal cycle that differs for the different SCATS sites. To make them consistent, we made slots of 5 minutes each and aggregate the traffic measures during that time slot for all the road segments.

Table 6.1: Top-5 influential road segments

Rank	**Road segment**	**RR Score**
	03-02-2012 08:05 AM	
1.	Hoddle St (Victoria Parade to Elizabeth St)	02.57095
2.	Hoddle St (Elizabeth St to Albert St)	02.00645
3.	Mills St (Canterbury Rd to Danks St)	01.89356
4.	Heidelberg Rd (Hoddle St to The Esplanade)	01.83253
5.	Heidelberg Rd (The Esplanade to Hoddle St)	01.81797
	03-02-2012 10:05 AM	
1.	Hoddle St (Victoria Parade to Elizabeth St)	03.49890
2.	Hoddle St (Elizabeth St to Albert St)	02.50019
3.	Heidelberg Rd (Hoddle St to The Esplanade)	02.38343
4.	Heidelberg Rd (The Esplanade to Hoddle St)	02.35427
5.	Hoddle St (Elizabeth St to Victoria Parade)	02.31045

Table 6.1 shows the 5 top-ranking influential road segments in the considered network on 03-12-2012 (Monday) at 08:05 AM and 10:05 AM. These results are obtained by setting the traffic condition factor $\boldsymbol{\alpha_{ij}}$ to 0.5, and getting the values of $\boldsymbol{vpn}$ and $\boldsymbol{vp(r_j)}$ from the dataset. A few road segments of Hoddle St and Heidelberg Rd are there in the list which are well known as a congestion hotspot for both the network operators and the commuters during the aforementioned time period. The ranking varies greatly for different values of $\boldsymbol{\alpha}$. We obtained the rankings at $\boldsymbol{\alpha = 0.00, 0.25, 0.50, 0.75, 1.00}$,

and compared the top-100 of them using the Kendal's Tau measure. The measures we obtained are KT$(\boldsymbol{RR_{\alpha=0}}, \boldsymbol{RR_{\alpha=0.25}}) = \mathbf{2409}$, KT$(\boldsymbol{RR_{\alpha=0}}, \boldsymbol{RR_{\alpha=0.5}}) = \mathbf{4516}$, KT$(\boldsymbol{RR_{\alpha=0}}, \boldsymbol{RR_{\alpha=0.75}}) = \mathbf{6428}$, and KT$(\boldsymbol{RR_{\alpha=0}}, \boldsymbol{RR_{\alpha=1}}) = \mathbf{7590}$. These values show the difference in behavior of Vo diffusion and DS diffusion. A good balance between these two diffusion can be obtained through $\boldsymbol{\alpha_{ij}}$ by incorporating the real traffic conditions to improve the ranking accuracy.

The ranking also keeps on changing with time based on the new traffic measures. Comparing the previous ranking to that obtained at 10:05 AM, we found that while few road segments in the top order remain same, those in the lower order change greatly with KT$(\boldsymbol{RR_{08:05AM}}, \boldsymbol{RR_{10:05AM}}) = \mathbf{3794}$ for the top 100 segments. Also some small road segments sometimes become influential for a short period of time but quickly goes down in the next time point as the congestion scenario changes. The convergences for $\boldsymbol{t_0} =$ 08:00 AM and 10:05 AM are achieved[5] in 922 (taking 36 seconds) and 1193 (taking 42 seconds) iterations respectively, which take significantly lesser iterations and time in incrementally computing the ranking at subsequent timestamps $\boldsymbol{t_l}$. Normally a traffic signal takes one to three minutes to complete a cycle. With this much interval of available time for processing the data, our algorithm is efficient enough to perform the computations in real time.

6.4 Summary

In this chapter, we presented two important applications and performed an experimental study using real traffic data. Through our analytical study, we showed the importance of these applications in understanding the propagation of congestion patterns. The first application monitors the congested partitions in dynamic urban road networks. It is able to capture both the spatial and temporal change in the congestion pattern over a period of time. The novelty of this method lies in utilizing the SCATS data for identifying the congested partitions by means of graph transformation, graph partitioning and incremental update based on different system parameters. We found that the traffic congestion in Melbourne can change rapidly in as short as 3 minutes interval during peak hours. Some of the interesting patterns we found include the anti-clockwise congestion formation

[5]Performed on a Core i5 computer with 8GB RAM

on Monash Freeway and Toorak Rd, and then rapidly forming bigger congestion in the east-west direction through the Eastern Fwy in less than 10 minutes. The framework can be a useful tool to assist traffic network operators via tracking congestion patterns and their behavior in real-time. In the second application, we developed a probabilistic traffic diffusion model to identify the most influential road segments in an urban road network. We presented `RoadRank` as a novel algorithm to compute the influence scores (or roadrank scores) of each road segment, which are updated incrementally with time to deal with the dynamic nature of traffic. The method starts with constructing a directed *influence graph* that is used to compute the traffic diffusion from one road segment to another at each timestamp. The *roadrank* scores are iteratively computed using probabilistic diffusion theory. We conducted experiments on historical traffic data collected from the 493 SCATS sites in Melbourne (Australia), and presented insights to the real traffic congestion scenario.

Chapter 7

Conclusion and Future Work

In this chapter, we summarize the major contributions made in this thesis and propose some interesting future research directions that can be explored further.

7.1 Summary of this Thesis

We proposed and implemented a series of techniques to partition the urban road traffic networks based on the traffic load, and capture the spatiotemporal evolution of congestion through the obtained differently congested partitions. We also presented two important applications to demonstrate the usefulness of our theoretical contributions and showed interesting experimental results on real traffic data. We now summarize the research works done in this thesis.

- Firstly, we developed a scalable method for traffic-based spatial partitioning of large urban road traffic networks (see Chapter 3). We investigated the structure of real road networks and developed an equivalent graph representation upon which the technical methods can be applied. The proposed method transforms the large road graph into a supergraph by clustering the road segments in a bottom-up manner. We developed a spectral theory based novel graph cut (referred as $\boldsymbol{\alpha}$-Cut) to partition the supergraph (or any graph). In our experimental results, we compared the performance of our $\boldsymbol{\alpha}$-Cut and the complete road network partitioning algorithm with existing existing methods. Our results show that the proposed method outperforms the normalized cut based existing method in all the performance evaluation metrics for small road networks and provides good results for much larger networks where other methods may face serious problems of time and space complexities.

- Secondly, we developed a robust framework for spatial partitioning of large urban road traffic networks using density peak graphs (see Chapter 4). It includes the properties of both density-based and spectral-based clustering methods, and provides an option to select a trade-off between efficiency and accuracy. It consists of a two-stage algorithm (referred as FaDSPa) embedded in the framework. It starts with transforming the large road graph into a well-structured and condensed density peak graph (DPG) using the concepts of density based clustering. Thereafter our spectral theory based $\boldsymbol{\alpha}$-Cut is applied on the DPG for partitioning, and the different sub-networks or partitions are obtained. Experimental results show that for large networks where efficiency is an important concern, we can opt for the settings to fasten its execution. This come at the cost of compromising accuracy up to some extent.

- Thirdly, we presented a comprehensive framework to track and capture the spatiotemporal evolution of road network partitions (see Chapter 5). The evolution is tracked by incrementally updating the partitions in an efficient manner, instead of repartitioning the network after each update. The functionalities are embedded in two different layers, including physical layer and logical layer. The physical layer maintains a large number of small-sized road network building blocks, and performs low-level computations to incrementally update them, whereas the logical layer performs high-level computations in order to serve as an interface to query the physical layer about the congested partitions. To capture the historical information and keep them saved, we also present an in-memory index called Bin that compactly stores the historical sets of building blocks with no information loss and facilitates their efficient retrieval. Our experimental results demonstrate the effectiveness and efficiency of the framework.

- Fourthly, we presented two important applications to study the traffic congestion propagation patterns in the dynamic urban road networks (see Chapter 6). These are the temporal tracking of congested partitions, and the traffic diffusion and influence estimation. We investigated real traffic data and presents our application-specific experimental study. During our study we found some interesting insights about the formation and spreading of the congestion. In the first application, we found that the traffic congestion in Melbourne can change rapidly in as short as 3 minutes interval during peak hours. Some of the interesting patterns we found include the anti-clockwise congestion formation on Monash Freeway and Toorak Rd, and then rapidly forming bigger congestion in the east-west direction through the Eastern Fwy in less than 10 minutes. In the second application, we developed a probabilistic traffic diffusion model to identify the most influential road segments in an urban road network. We found that the Hoddle Steert, Mills Street, and Heidelberg Road are among the most influential roads in the Melbourne road network in the morning hours.

In a nutshell, the techniques proposed in this thesis provide a way to analyze and understand the sptiotemporal evolution of traffic congestion in dynamic urban road networks. The traffic congestion is marked by the characteristics of being *spatially diverse* and *temporally dynamic*. By partitioning the road network, we capture the spatially diverse property of congestion in the form of differently congested partitions. By incrementally updating those partitions, we capture its temporally dynamic nature in the form of the change in structure and location of those partitions. We have greatly taken care of the efficiency of our methods so that the tracking of the evolution can be performed in real time. This thesis aims at contributing to the area of traffic data analysis for an intelligent traffic (and/or congestion) management in order to aid the smart transportation services. We hope that our research contributes towards improving the understanding of spatiotemporal evolution of traffic congestion.

7.2 Future Work

In this section, we point out several other interesting research directions that might be explored in the future.

7.2.1 Extension of the Thesis

In this section, we point out several future works that can be attempted to improve and /or extend the works presented in this thesis.

- Traffic congestions in peak hours is a big problem in urban road networks. The road network partitions discussed in Chapter 3 and Chapter 4 are the sets of connected differently congested segments of a road network at a point of time, which are determined by the level of traffic in them. In the road network partitions, it is often found that there are some road segments whose traffic density do not match at all with the partition in which they belong. Their connected neighboring road segments have measures similar to that of the partitions. Because connectivity plays an important role in partitioning, such non-matching road segments still become part of that partition. One possible future work could be developing some method to consider them as outliers and treating them accordingly.
- We developed two methods to partition the road traffic networks. The first one uses only spectral clustering, whereas the second one uses both spectral and density based clusytring. The second one is more efficient than the first, but still the execution time is long. Improving this execution time further while maintaining a good accuracy may not be so easy, but is definitely a problem to extend this work further.
- In Chapter 5 we incrementally update the partitions to track the spatiotemoporal evolution. While doing the incremental update, our method checks each and every node that lie on the boundary to compute its stability measure, and check each unstable node to reallocate to the most suitable partition. One good thing we do is that this measure is computed only for the boundary nodes, not all the nodes in the network, but still computing the suitable partition for each unstable node one by one is a tedious task. Some mechanisms could be developed to do the changed or re-allocation in groups of nodes or subgraphs. Also the merge and split operations on the partitions could be further developed to improve the efficiency.
- We have also presented an in-memory index to store the historical building blocks in Chapter 5. One thing we noticed is that after long times of execution, when there is

lots of historical information stored and the index has become very big size, the tree accumulates many unwanted duplicate nodes. It makes the index inefficient. This can be handled by developing a method to regularly restructure the tree by filter the unwanted duplicate nodes. Also the historical information can be squeezed at egular intervals to maintain them sparsely, as generally we would not want the information of long time back in fine granularity. As it is not possible to store all the historical data in primary memory, development of an index for secondary memory storage is also a problem worth considering.

- Some previous works have suggested to use partitions for route guidance, but do not consider the real time traffic level to suggest the fastest route. An important future work would be to investigate computation of fastest route from a source to a destination by making use of the maintained differently congested partitions (from Chapter 5).
- Our method for identifying influential road segments in a road network discussed in Chapter 6 uses the diffused traffic. There has been some study on identifying hot routes previously. These routes are generally targeted for marketing different commercial products, or for other purposes they are considered important. It would be interesting to extend our method further to identify the influential routes, and compare how similar are the influential route with the hot routes.

7.2.2 Other Unexplored Areas

There are many other unexplored areas related to advanced and intelligent traffic data analysis, which could be very useful for the traffic management authorities and the daily commuters. Some of the important areas are discussed below, and can be considered for further research.

- **Exploiting Socio-Spatial Data for Advanced Traffic Management:** With the growing use of social media on mobile smart-phones and increasing pervasiveness of geo-positioning technologies, location-based social networks (LBSN), like Foursquare, are getting popular these days [72]. An LBSN is a social networking website coupled with location-based services. It allows a user to check-in at venues using a mobile phone and share it with others in realtime, where the venue corresponds to a location on the geographic map. Foursquare is the most popular LBSN today, which has over 30M users, and receives millions of check-ins per day [71]. Recently, the traditional social networking sites, Facebook and Twitter, have also added the check-in service functionality. Through these sources, huge amounts of socio-spatial data are getting accumulated each day. It would be very interesting to design methodologies to discover hidden facts and traffic patterns from these data for advanced traffic management. The challenges with this problem are the large size and semi-structured nature of the data. To overcome these challenges, efficient data indexing structures and efficient algorithms need to be designed. This task consist of two subtasks, which are as follows.

- **Unusual geographic event detection through socio-spatial data:** Transport departments always try to maintain a smooth traffic flow on roads. However, if an irregularity in traffic flow, like crowd, occurs somewhere, they need to get informed about this in real-time. The term *unusual socio-spatial event* refers to any abnormal crowd behavior at a specific geographic location. In [77], Lee and Sumiya studied the problem of unusual event detection, but their method did not consider efficiency as a major concern. Solving this problem so as to monitor the LBSN in realtime is very important, as the monitoring has to be performed continuously. Developing methodologies to continuously monitor an LBSN to investigate the crowd regularities and automatically detect *unusual socio-spatial events* would greatly benefit the transport authorities.

- **Socio-spatial-traffic keyword queries:** In different location-based services, information about locations is retrieved by passing spatial queries that consist of the latitude and longitude coordinates, e.g. retrieving specific portion of a geographic map. An advanced way to retrieve these information is to pass spatial keyword queries, that consist of textual keywords in addition to geographic coordinates [46, 140]. The keywords characterize locations and they are so common that people are generally familiar with them. The combination of the two query components make it a more meaningful query to retrieve an enriched information. For example, a query

"*Indian restaurant* nearest to me" gets the geographic coordinates from the device geo-positioning technology and the keywords as "Indian restaurant". With the rapid growth in accumulation of spatial textual data through social media, it has become much easier to answer the spatial keyword queries. Due to its increasing demand, it has become a widely accepted problem and various models and techniques are being proposed these days. The top-k spatial keyword queries [49–51] ranks the $\boldsymbol{k}$ best matching spatial objects in terms of both spatial proximity to the query location and textual relevance to the query keywords are returned. So far there has been no work on *socio-spatial-traffic* queries to the best of our knowledge. These queries combine the information of traffic load on the route along with the social and spatial information. As the keyword queries are very commonly used in our daily lives, developing techniques to solve traffic based queries would give a lot more information to the users.

- **Detecting and Predicting Repeatable Congestion Related Events:** Traffic congestion is very much related to unusual events happening in an urban area. These events can be as big as a football match in the Etihad Stadium (Melbourne), or as small as a rush to the train station to board a specific train. Many times these events go unnoticed and congestions keep on repeating with unknown patterns and reasons. It is a very important research area to identify the patterns of such repeatable events that happened in the past, and predict any such events going to happen in the near future.

Bibliography

[1] Ken Henry. To build or not to build: Infrastructure challenges in the years ahead and the role of the government. In *Address to the Conf. on The Economics of Infrastructure in a Globalised World: Issues, Lessons and Future Challenges*, 2010.

[2] BTRE. Estimating urban traffic and congestion cost trends for australian cities, 2007. Working paper 71, BTRE, Canberra ACT.

[3] Tarique Anwar, Chengfei Liu, Hai L. Vu, and Christopher Leckie. Spatial partitioning of large urban road networks. In *Proc. of the EDBT*, pages 343–354, 2014.

[4] Tarique Anwar, Chengfei Liu, Hai L. Vu, and Christopher Leckie. Partitioning road networks using density peak graphs: Efficiency vs. accuracy. *Inf. Syst.*, 64: 22–40, 2017. doi: 10.1016/j.is.2016.09.006. URL `http://dx.doi.org/10.1016/j.is.2016.09.006`.

[5] Tarique Anwar, Chengfei Liu, Hai L. Vu, and Md. Saiful Islam. Tracking the evolution of congestion in dynamic urban road networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 2323–2328, 2016. doi: 10.1145/2983323.2983688. URL `http://doi.acm.org/10.1145/2983323.2983688`.

[6] Tarique Anwar, Hai L. Vu, Chengfei Liu, and Serge P. Hoogendoorn. Temporal tracking of congested partitions in dynamic urban road networks. In *Proc. of the TRB Annual Meeting*, 2016.

[7] Tarique Anwar, Hai L. Vu, Chengfei Liu, and Serge P. Hoogendoorn. Temporal tracking of congested partitions in dynamic urban road networks. *Transportation Research Record: Journal of the Transportation Research Board*, 2595:8897, 2016.

[8] Tarique Anwar, Chengfei Liu, Hai L. Vu, and Md. Saiful Islam. Roadrank: Traffic diffusion and influence estimation in dynamic urban road networks. In *Proc. of the CIKM*, pages 1671–1674, 2015.

[9] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Symposium on Math, Statistics, and Probability*, pages 281–297, 1967.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[11] Kriegel Hans-Peter, Kroger Peer, Sander Jorg, and Zimek Arthur. Density–based clustering. *WIREs Data Mining Knowl Discov*, 1(3):231–240, 2011.

[12] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the KDD*, pages 226–231, 1996.

[13] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.

[14] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. Scan: A structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 824–833, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7. doi: 10.1145/1281192.1281280. URL `http://doi.acm.org/10.1145/1281192.1281280`.

[15] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Scan++: Efficient algorithm for finding clusters, hubs and outliers on large-scale graphs. *Proc. VLDB Endow.*, 8(11):1178–1189, July 2015. ISSN 2150-8097. doi: 10.14778/2809974.2809980. URL `http://dx.doi.org/10.14778/2809974.2809980`.

[16] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, August 2000. ISSN 0162-8828.

[17] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graph. In *Proc. of the SDM*, 2005.

[18] C.H.Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and H.D. Simon. A min-max cult algorithm for graph partitioning and data clustering. In *Proc. of the ICDM*, pages 107–114, 2001.

[19] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *CoRR*, abs/1311.3144, 2013. URL `http://arxiv.org/abs/1311.3144`.

[20] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *Proc. VLDB Endow.*, 2(1):718–729, August 2009.

[21] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

[22] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partition. In *Proceedings of the 19th Design Automation Conference*, pages 175–181, 1982.

[23] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, December 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z. URL `http://dx.doi.org/10.1007/s11222-007-9033-z`.

[24] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, 2001.

[25] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: Spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 551–556, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014118. URL `http://doi.acm.org/10.1145/1014052.1014118`.

[26] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(11):1101–1113, November 1993. ISSN 0162-8828. doi: 10.1109/34.244673. URL `http://dx.doi.org/10.1109/34.244673`.

[27] Mohammed J. Zaki and Wagner Meira Jr. *Data Mining and Analysis: Fundamental Concepts and Algorithms*, pages 421–435. Cambridge University Press, 2014.

[28] Mijung Kim and K. Selçuk Candan. Sbv-cut: Vertex-cut based graph partitioning using structural balance vertices. *Data Knowl. Eng.*, 72:285–303, February 2012.

[29] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. Rankclus: Integrating clustering with ranking for heterogeneous information network analysis. In *Proc. of the EDBT*, pages 565–576, 2009.

[30] Jure Leskovec, Kevin J. Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *Proc. of the WWW*, pages 631–640, 2010.

[31] Hanan Samet. *Spatial data structures*, pages 361–385. ACM Press and Addison-Wesley, 1995.

[32] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, 1984. ISBN 0-89791-128-8.

[33] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 507–518, 1987. ISBN 0-934613-46-X.

[34] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD '90, pages 322–331, 1990. ISBN 0-89791-365-5.

[35] Mark Overmars Mark de Berg, Marc van Kreveld and Otfried Schwarzkopf. *Quadtrees*, page 291306. Springer-Verlag, 2000.

[36] Nikos Mamoulis. Morgan & Claypool Publishers, 2011.

[37] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Inf.*, 4(1):1–9, March 1974. ISSN 0001-5903. doi: 10.1007/BF00288933. URL `http://dx.doi.org/10.1007/BF00288933`.

[38] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 802–813. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL `http://dl.acm.org/citation.cfm?id=1315451.1315520`.

[39] João B. Rocha-Junior, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Efficient processing of top-k spatial preference queries. *Proc. VLDB Endow.*, 4(2): 93–104, November 2010. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1921071.1921076`.

[40] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, pages 71–79, New York, NY, USA, 1995. ACM. ISBN 0-89791-731-6. doi: 10.1145/223784.223794. URL `http://doi.acm.org/10.1145/223784.223794`.

[41] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 201–212, New York, NY, USA, 2000. ACM. ISBN 1-58113-217-4. doi: 10.1145/342009.335415. URL `http://doi.acm.org/10.1145/342009.335415`.

[42] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. Finding fastest paths on a road network with speed patterns. In *Proc. of the ICDE*, 2006.

[43] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. On computing top-t most influential spatial sites. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 946–957. VLDB Endowment, 2005. ISBN 1-59593-154-6. URL `http://dl.acm.org/citation.cfm?id=1083592.1083701`.

[44] Yang Du, Donghui Zhang, and Tian Xia. The optimal-location query. In *Proceedings of the 9th international conference on Advances in Spatial and Temporal Databases*, SSTD'05, pages 163–180, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-28127-4, 978-3-540-28127-6. doi: 10.1007/11535331_10. URL `http://dx.doi.org/10.1007/11535331_10`.

[45] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. Efficient query processing in geographic web search engines. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 277–288, New York, NY, USA, 2006. ACM. ISBN 1-59593-434-0. doi: 10.1145/1142473.1142505. URL `http://doi.acm.org/10.1145/1142473.1142505`.

[46] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 656–665, 2008. ISBN 978-1-4244-1836-7.

[47] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Proceedings of the 12th international conference on Advances in spatial and temporal databases*, SSTD'11, pages 205–222, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22921-3. URL `http://dl.acm.org/citation.cfm?id=2035253.2035270`.

[48] João B. Rocha-Junior and Kjetil Nørvåg. Top-k spatial keyword queries on road networks. In *Proc. of the 15th EDBT*, pages 168–179, 2012.

[49] Dingming Wu, Man Lung Yiu, Christian S. Jensen, and Gao Cong. Efficient continuously moving top-k spatial keyword query processing. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 541–552, 2011. ISBN 978-1-4244-8959-6.

[50] Dongxiang Zhang, Kian-Lee Tan, and Anthony K. H. Tung. Scalable top-k spatial keyword search. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 359–370, 2013.

[51] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. pages 901–912, 2013. ISSN 1063-6382.

[52] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB Endow.*, 2(1):337–348, August 2009. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1687627.1687666`.

[53] Nikos Mamoulis, Huiping Cao, George Kollios, Marios Hadjieleftheriou, Yufei Tao, and David W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 236–245, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014080. URL `http://doi.acm.org/10.1145/1014052.1014080`.

[54] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 330–339, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7. doi: 10.1145/1281192.1281230. URL `http://doi.acm.org/10.1145/1281192.1281230`.

[55] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 593–604, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-686-8. doi: 10.1145/1247480.1247546. URL `http://doi.acm.org/10.1145/1247480.1247546`.

[56] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proc. VLDB Endow.*, 1(1):1081–1094, August 2008. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1453856.1453972`.

[57] Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez. Traffic density-based discovery of hot routes in road networks. In *Proc. of the SSTD*, pages 441–459, 2007.

[58] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. Discovering popular routes from trajectories. In *Proc. of the ICDE*, pages 900–911, 2011.

[59] Hector Gonzalez, Jiawei Han, Xiaolei Li, Margaret Myslinska, and John Paul Sondag. Adaptive fastest path computation on a road network: a traffic mining approach. In *Proc. of the VLDB*, pages 794–805, 2007.

[60] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 791–800, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526816. URL `http://doi.acm.org/10.1145/1526709.1526816`.

[61] Yuxuan Ji and Nikolas Geroliminis. On the spatial partitioning of urban transportation networks. *Transportation Research Part B: Methodological*, 46(10):1639–1656, 2012.

[62] Dimitris Sacharidis, Kostas Patroumpas, Manolis Terrovitis, Verena Kantere, Michalis Potamias, Kyriakos Mouratidis, and Timos Sellis. On-line discovery of hot motion paths. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 392–403, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-926-5. doi: 10.1145/1353343.1353392. URL `http://doi.acm.org/10.1145/1353343.1353392`.

[63] Nicolas Lefebvre and Michael Balmer. Fast shortest path computation in time-dependent traffic networks, 2007. Working paper 439.

[64] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *Proceedings of the Fifth International Conference on Information and Knowledge Management*, CIKM '96, pages 261–268, New York, NY, USA, 1996. ACM. ISBN 0-89791-873-8. doi: 10.1145/238355.238550. URL `http://doi.acm.org/10.1145/238355.238550`.

[65] Yu-Li Chou, H. Edwin Romeijn, and Robert L. Smith. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems.

INFORMS J. on Computing, 10(2):163–179, February 1998. ISSN 1526-5528. doi: 10.1287/ijoc.10.2.163. URL `http://dx.doi.org/10.1287/ijoc.10.2.163`.

[66] Camil Demetrescu, Stefano Emiliozzi, and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 369–378, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. ISBN 0-89871-558-X. URL `http://dl.acm.org/citation.cfm?id=982792.982845`.

[67] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *Proceedings of the 13th Annual European Conference on Algorithms*, ESA'05, pages 568–579, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29118-0, 978-3-540-29118-3. doi: 10.1007/11561071_51. URL `http://dx.doi.org/10.1007/11561071_51`.

[68] Ugur Demiryurek, Farnoush Banaei-Kashani, Cyrus Shahabi, and Anand Ranganathan. Online computation of fastest path in time-dependent spatial networks. In *Proceedings of the 12th International Conference on Advances in Spatial and Temporal Databases*, SSTD'11, pages 92–111, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22921-3. URL `http://dl.acm.org/citation.cfm?id=2035253.2035263`.

[69] Abdul Majida, Ling Chena, Hamid Turab Mirzaa, Ibrar Hussaina, and Gencai Chen. A system for mining interesting tourist locations and travel sequences from public geo-tagged photos. *Data & Knowledge Engineering*, 95:66–86, January 2015.

[70] Xin Cao, Gao Cong, and Christian S. Jensen. Mining significant semantic locations from gps data. *Proc. VLDB Endow.*, 3(1-2):1009–1020, September 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920968. URL `http://dx.doi.org/10.14778/1920841.1920968`.

[71] Nikos Armenatzoglou, Stavros Papadopoulos, and Dimitris Papadias. A general framework for geo-social query processing. *Proc. VLDB Endow.*, 6(10):913–924, August 2013. ISSN 2150-8097.

[72] De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. On socio-spatial group query for location-based social networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 949–957, 2012.

[73] Weimo Liu, Weiwei Sun, Chunan Chen, Yan Huang, Yinan Jing, and Kunjie Chen. Circle of friend query in geo-social networks. In *Proceedings of the 17th international conference on Database Systems for Advanced Applications - Volume Part II*, DASFAA'12, pages 126–137, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29034-3. doi: 10.1007/978-3-642-29035-0_9. URL `http://dx.doi.org/10.1007/978-3-642-29035-0_9`.

[74] Chi-Yin Chow, Jie Bao, and Mohamed F. Mokbel. Towards location-based social networking services. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, LBSN '10, pages 31–38, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0434-4. doi: 10.1145/1867699.1867706. URL `http://doi.acm.org/10.1145/1867699.1867706`.

[75] Jie Bao, Yu Zheng, and Mohamed F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 199–208, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1691-0. doi: 10.1145/2424321.2424348. URL `http://doi.acm.org/10.1145/2424321.2424348`.

[76] Justin J. Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F. Mokbel. Lars: A location-aware recommender system. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 450–461, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4747-3. doi: 10.1109/ICDE.2012.54. URL `http://dx.doi.org/10.1109/ICDE.2012.54`.

[77] Ryong Lee and Kazutoshi Sumiya. Measuring geographical regularities of crowd behaviors for twitter-based geo-social event detection. In *Proc. of the ACM SIGSPATIAL Int'l Workshop on LBSN*, pages 1–10, 2010.

[78] Ryong Lee, Shoko Wakamiya, and Kazutoshi Sumiya. Discovery of unusual regional social activities using geo-tagged microblogs. *World Wide Web*, 14(4):321–349, July 2011. ISSN 1386-145X. doi: 10.1007/s11280-011-0120-x. URL `http://dx.doi.org/10.1007/s11280-011-0120-x`.

[79] Nan Li and Guanling Chen. Analysis of a location-based social network. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04*, CSE '09, pages 263–270, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3823-5. doi: 10.1109/CSE.2009.98. URL `http://dx.doi.org/10.1109/CSE.2009.98`.

[80] Chao Zhang, Lidan Shou, Ke Chen, Gang Chen, and Yijun Bei. Evaluating geo-social influence in location-based social networks. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 1442–1451, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1156-4. doi: 10.1145/2396761.2398450. URL `http://doi.acm.org/10.1145/2396761.2398450`.

[81] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, VLDB '94, pages 144–155, 1994.

[82] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[83] J. W. Han, M. Kamber, and A. K. H. Tung. *Spatial clustering methods in data mining: A survey*, pages 188–217. Taylor & Francis, 2001.

[84] Chih-Chieh Hung, Wen-Chih Peng, and Wang-Chien Lee. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *The VLDB Journal*, 24(2):169–192, April 2015. ISSN 1066-8888.

[85] Ling-Yin Wei and Wen-Chih Peng. An incremental algorithm for clustering spatial data streams: exploring temporal locality. *Knowl. Inf. Syst.*, 37(2):453–483, 2013. doi: 10.1007/s10115-013-0636-8. URL `http://dx.doi.org/10.1007/s10115-013-0636-8`.

[86] Dapeng Li, Joerg Sander, Mario A. Nascimento, and Dae-Won Kwon. Discovering spatial co-clustering patterns in traffic collision data. In *Proc. of the ACM SIGSPATIAL Int'l Workshop on Computational Transportation Science*, IWCTS '13, pages 55:55–55:60, 2013.

[87] Hamideh Etemadniaa, Khaled Abdelghanya, and Ahmed Hassan. A network partitioning methodology for distributed traffic management applications. *Transportmetrica A: Transport Science*, 10(6):518–532, 2014.

[88] Jiping Wang, Kai Zheng, Hoyoung Jeung, Haozhou Wang, Bolong Zheng, and Xiaofang Zhou. Cost-efficient spatial network partitioning for distance-based query processing. In *Proc. of the MDM*, pages 13–22, 2014.

[89] Zhengdao Xu and Hans-Arno Jacobsen. Processing proximity relations in road networks. In *Proc. of the ACM SIGMOD*, SIGMOD '10, pages 243–254, 2010.

[90] Georgios Kellaris and Kyriakos Mouratidis. Shortest path computation on air indexes. *Proc. VLDB Endow.*, 3(1-2):747–757, September 2010.

[91] Bowu Zhang, Kai Xing, Xiuzhen Cheng, Liusheng Huang, and Rongfang Bie. Traffic clustering and online traffic prediction in vehicle networks: A social influence perspective. In *Proc. of the IEEE INFOCOM*, pages 495–503, 2012.

[92] Da Yan, James Cheng, Wilfred Ng, and Steven Liu. Finding distance-preserving subgraphs in large road networks. In *Proc. of IEEE ICDE*, pages 625–636, 2013.

[93] Anand Meka and Ambuj K. Singh. Distributed spatial clustering in sensor networks. In *Proc. of the EDBT*, pages 980–1000, 2006.

[94] Min-Soo Kim and Jiawei Han. A particle-and-density based evolutionary clustering method for dynamic networks. *Proc. VLDB Endow.*, 2(1):622–633, August 2009. ISSN 2150-8097. doi: 10.14778/1687627.1687698. URL `http://dx.doi.org/10.14778/1687627.1687698`.

[95] Yuxuan Ji, Jun Luo, and Nikolas Geroliminis. Empirical observations of congestion propagation and dynamic partitioning with probe data for large-scale systems. *Transportation Research Record*, 2422:1–11, 2014.

[96] Pei Lee, Laks V. S. Lakshmanan, and Evangelos E. Milios. Incremental cluster evolution tracking from highly dynamic network data. In *Proc. ICDE*, pages 3–14, 2012.

[97] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *VLDB*, 5(6):574–585, 2012.

[98] Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanta Tirthapura. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *VLDB J.*, 23(2):175–199, 2014. doi: 10.1007/s00778-013-0340-z. URL `http://dx.doi.org/10.1007/s00778-013-0340-z`.

[99] Manoj K. Agarwal, Krithi Ramamritham, and Manish Bhide. Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments. *VLDB*, 5(10):980–991, 2012.

[100] Derek Greene, Donal Doyle, and Padraig Cunningham. Tracking the evolution of communities in dynamic social networks. In *Proc. of the ASONAM*, pages 176–183, 2010.

[101] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7), April 1998.

[102] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46 (5), September 1999.

[103] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966. URL `http://EconPapers.repec.org/RePEc:spr:psycho:v:31:y:1966:i:4:p:581-603`.

[104] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[105] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, page 215, 1978.

[106] Daniel Gruhl, R. Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. In *Proc. WWW*, 2004.

[107] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Proc. ICALP*, 2005.

[108] Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proc. WWW*, 2009.

[109] Xiaodan Song, Yun Chi, Koji Hino, and Belle Tseng. Identifying opinion leaders in the blogosphere. In *Proc. CIKM*, 2007.

[110] Amit Goyal, Francesco Bonchi, and Laks V.S. Lakshmanan. Learning influence probabilities in social networks. In *Proc. WSDM*, 2010.

[111] Arlei Silva, Sara Guimarães, Wagner Meira, Jr., and Mohammed Zaki. Profilerank: Finding relevant content and influential users based on information diffusion. In *Proc. SNAKDD*, 2013.

[112] Jonathan Herzig, Yosi Mass, and Haggai Roitman. An author-reader influence model for detecting topic-based influencers in social media. In *Proc. HT*, 2014.

[113] Zuchao Wang, Min Lu, Xiaoru Yuan, Junping Zhang, and Huub van de Wetering. Visual traffic jam analysis based on trajectory data. *IEEE Trans. on Visualization and Computer Graphics*, 19(12):2159–2168, December 2013. ISSN 1077-2626.

[114] Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 344–353, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2521-9. doi: 10.1145/2525314.2525343. URL `http://doi.acm.org/10.1145/2525314.2525343`.

[115] JianCheng Long, ZiYou Gao, HuaLing Ren, and AiPing Lian. Urban traffic congestion propagation and bottleneck identification. *Science in China Series F: Information Sciences*, 51(7):948–964, 2008.

[116] Daqing Li, Bowen Fu, Yunpeng Wang, Guangquan Lu, Yehiel Berezin, H. Eugene Stanley, and Shlomo Havlin. Percolation transition in dynamical traffic network with evolving critical bottlenecks. *Proceedings of the National Academy of Sciences*, 112 (3):669–672, 2015.

[117] Hongsheng Qi, Meiqi Liu, Lihui Zhang, and Dianhai Wang. Tracing road network bottleneck by data driven approach. *PLoS One*, 11(5), 2016.

[118] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 140–149, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-1-4244-1836-7. doi: 10.1109/ICDE.2008.4497422. URL `http://dx.doi.org/10.1109/ICDE.2008.4497422`.

[119] Wei Liu, Yu Zheng, Sanjay Chawla, Jing Yuan, and Xie Xing. Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1010–1018, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020571. URL `http://doi.acm.org/10.1145/2020408.2020571`.

[120] Sanjay Chawla, Yu Zheng, and Jiafeng Hu. Inferring the root cause in road traffic anomalies. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*, ICDM '12, pages 141–150, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4905-7. doi: 10.1109/ICDM.2012.104. URL `http://dx.doi.org/10.1109/ICDM.2012.104`.

[121] Jinsong Lan, Cheng Long, Raymond Chi-Wing Wong, Youyang Chen, Yanjie Fu, Danhuai Guo, Shuguang Liu, Yong Ge, Yuanchun Zhou, and Jianhui Li. A new framework for traffic anomaly detection. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 875–883, 2014. doi: 10.1137/1.9781611973440.100. URL `http://dx.doi.org/10.1137/1.9781611973440.100`.

[122] R. Guimera, S. Mossa, A. Turtschi, and L.A.N. Amaral. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proceedings of the National Academy of Sciences*, 102(22):7794–7799, 2005.

[123] Hao Lei, Tao Xing, Jeffrey D. Taylor, and Xuesong Zhou. Monitoring travel time reliability from the cloud. *Transportation Research Record: Journal of the Transportation Research Board*, 2291:35–43, 2012.

[124] Mohammed Elhenawy and Hesham A. Rakha. Congestion prediction using adaptive boosting machine learning classifiers. In *Proc. of the 93rd Annual TRB Meeting*, 2014.

[125] Huijun Sun, Jianjun Wu, Dan Maa, and Jiancheng Long. Spatial distribution complexities of traffic congestion and bottlenecks in different network topologies. *Applied Mathematical Modelling*, 38(2):496–505, 2014.

[126] Zhao Zhou, Shu Lin, and Yugeng Xi. A dynamic network partition method for heterogenous urban traffic networks. In *Proc. of the IEEE ITSC*, pages 820–825, 2012.

[127] Takahiro Tsubota, Ashish Bhaskar, and Edward Chung. Brisbane macroscopic fundamental diagram: Empirical findings on network partitioning and incident detection. In *Proc. of the 93rd Annual TRB Meeting*, 2014.

[128] Betty Salzberg and Vassilis J. Tsotras. Comparison of access methods for time-evolving data. *ACM Comput. Surv.*, 31(2), 1999.

[129] Richard Snodgrass and Ilsoo Ahn. A taxonomy of time databases. In *Proc. of the ACM SIGMOD*, pages 236–246, 1985.

[130] Gultekin Ozsoyoglu and Richard Thomas Snodgrass. Temporal and real-time databases: A survey. *IEEE TKDE*, 7(4):513–532, 1995.

[131] Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In *ICDE*, pages 997–1008, 2013.

[132] Yunjae Jung, Haesun Park, Ding-Zhu Du, and Barry L. Drake. A decision criterion for the optimal number of clusters in hierarchical clustering. *J. of Global Optimization*, 25(1):91–111, January 2003. ISSN 0925-5001.

[133] Mohamed F. Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. Mntg: an extensible web-based traffic generator. In *Proc. of the SSTD*, pages 38–55, 2013.

[134] Jack J. Dongarra, Danny C. Sorensen, and Sven J. Hammarling. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics*, 27(1–2):215–227, September 1989.

[135] Donghui Yan, Ling Huang, and Michael I. Jordan. Fast approximate spectral clustering. In *Proc. of ACM SIGKDD*, pages 907–916, 2009.

[136] Mohamed F. Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. A demonstration of mntg– a web-based road network traffic generator. In *Proc. of the ICDE*, pages 1246–1249, 2014.

[137] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[138] Ngoc Nguyen and John Gaffney. Real-time traffic performance measures of adaptive traffic signal control systems. In *Proc. of the 22nd ARRB Conference*, 2006.

[139] Alexander Artikis, Matthias Weidlich, Francois Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, Avigdor Gal, Shie Mannor, Dimitrios Gunopulos, and Dermot Kinane. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *Proc. of the EDBT*, pages 712–723, 2014.

[140] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: an experimental evaluation. *Proc. VLDB Endow.*, 6(3):217–228, January 2013. ISSN 2150-8097.

Author's Publications

1. Tarique Anwar, Chengfei Liu, Hai L. Vu, and Christopher Leckie, Spatial Partitioning of Large Urban Road Networks, **Proceedings of the 17th International Conference on Extending Database Technology**, Athens, Greece, Pages 343-354, 24-28 Mar, 2014 (ERA Rank: A)

2. Tarique Anwar, Chengfei Liu, Hai L. Vu, and Md. Saiful Islam, RoadRank: Traffic Diffusion and Influence Estimation in Dynamic Urban Road Networks, **Proceedings of the 24th ACM International Conference on Information and Knowledge Management**, Melbourne, Australia, Pages 1671-1674, 19-23 Oct, 2015 (ERA Rank: A)

3. Tarique Anwar, Hai L. Vu, Chengfei Liu, and Serge Hoogendoorn, Temporal Tracking of Congested Partitions in Dynamic Urban Road Networks, **Proceedings of the Transport Research Board 95th Annual Meeting**, Washington, D.C., USA, 10-14 Jan, 2016 (ERA Rank: A)

4. Tarique Anwar, Hai L. Vu, Chengfei Liu, and Serge Hoogendoorn, Temporal Tracking of Congested Partitions in Dynamic Urban Road Networks, **Transportation Research Record: Journal of the Transportation Research Board**, Vol 2595, Pages 88-97, 2016 (ERA Rank: A)

5. Tarique Anwar, Chengfei Liu, Hai L. Vu, and Md. Saiful Islam, Tracking the Evolution of Congestion in Dynamic Urban Road Networks, **Proceedings of the 25th ACM International Conference on Information and Knowledge Management**, Indianapolis, USA, Pages 2323-2328, 24-28 Oct, 2016 (ERA Rank: A)

6. Tarique Anwar, Chengfei Liu, Hai L. Vu, and Christopher Leckie, Partitioning road networks using density peak graphs: Efficiency vs. accuracy, **Information Systems**, Vol 64, Pages 22-40, 2017 (ERA Rank: A*)