Popular Content Distribution in Public Transportation Using Artificial Intelligence Techniques

by

KAIS EL-MURTADI SULEIMAN

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Abdulmotaleb El Saddik Professor
Supervisor	Otman Basir Professor
Internal Member	Pin-Han Ho Professor
Internal Member	Mohamed Oussama Damen Professor
Internal-external Member	Hamid Tizhoosh Professor

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Outdoor wireless networks suffer nowadays from an increasing data traffic demand which comes at the time where almost no vacant frequency spectrum has been left. A vast majority of this demand comes from popular content generated by video streaming and social media sites. In the future, other sources will generate even more demand with emerging applications such as virtual reality, connected cars and environmental sensing. While a significant progress has been made to address this network saturation in indoor environments, current outdoor solutions, based on fixed network deployments, are expensive to build and maintain. They tend to be immobile and therefore are inflexible in coping with the dynamics of outdoor data demand. On the other hand, Vehicular Ad-hoc NETworks (VANETs) are in nature more scalable, dynamic, flexible, and therefore more promising in terms of addressing such demand. This is especially feasible if we take advantage of public transportation vehicles and stops. These vehicles and stops are often owned and operated by the same administrative entity which overcomes the routing selfishness issue. Moreover, the mobility patterns of these vehicles are highly predictable given their regular schedules; their locations are publicly-sharable and their location distribution is uniform throughout space and time. Given these factors, a system that utilizes public transportation vehicles and stops to build a reliable, scalable and dynamic VANET for wireless network offloading in outdoor environments is proposed. This is done by exploiting the predictability demonstrated by such vehicles using an Artificial-Intelligence (AI) based system for wireless network offloading via popular content distribution. The AI techniques used are the Upper Popularity Bound (UPB) collaborative and group-based recommender based on multi-armed bandits for content recommendation and bayesian optimization based on batch-based Random Forest (RF) regression for content routing. They are used after analyzing the mobility data of public transportation vehicles and stops. This analysis includes both preprocessing and processing the data in order to select the optimal set of stops and clustering vehicles and stops based on cumulative contact duration thresholds. The final system has shown the promising networking potential of public transportation. It incorporates a recommender that has shown a versatile performance under different consumer interest and network capacity scenarios. It has also demonstrated a superior performance using a bayesian optimization technique that offloads as high as 95% of the wireless network load in an interference and collision free manner.

Acknowledgments

All praises to God, I thank Him for all the blessings He gave and all the blessings He delayed but not denied. Without His blessings, I would not have reached this stage of knowledge.

Next, I would like to thank Prof. Otman Basir for his profound supervision during the course of my studies. My mere thanks would not suffice the long hours of discussions, continuous guidance and invaluable feedback.

I would like also to thank all the members of my dissertation committee for their valuable remarks and insightful recommendations.

Moreover, I acknowledge with a great appreciation the continuous funding provided by the Libyan Ministry of Higher Education and the support I received from Prof. Basir.

And last, but not the least, I would like to express my sincere gratitude to my dear family. My parents: Abdelrazeg and Mabruka; my siblings: Waiel, Mohannad and Heba; my wife: Salma and my son: Abdelrazeg. I dedicate this work to you and pray to God to bless you all.

Kais El-murtadi Suleiman

Waterloo, Canada

August, 2019

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

Δcd	Length of the contact duration within one time step
ϵ	Probability of randomness
σ_i^g	Standard deviation of the consumer group interests distributions
μ_ω	Policy weights mean within the data batch used by the batch-based random forest regression model used throughout bayesian optimization
ω	Vector of policy weights given to the node features
ω_{max}	Maximum policy weight bound used by the batch-based random forest regression model throughout bayesian optimization
ω_{min}	Minimum policy weight bound used by the batch-based random forest regression model throughout bayesian optimization
σ_ω	Policy weights standard deviation within the data batch used by the batch-based random forest regression model used throughout bayesian optimization
$\mathbf{A}$	Matrix of data segment exchange actions
$\mathbf{CD}$	Set of contact duration matrices
$\mathbf{cd}$	Counters vector used to measure continuous contact durations
$\mathbf{cd}_{min}$	Schedule vector of minimum cumulative contact duration thresholds used in the successive biggest node clusterings
$\mathbf{CO}_{bl}$	Blocks coordinates matrix
$\mathbf{co}_{s_i}$	Vector of coordinates of refined stop (i)
$\mathbf{CO}_{sc_i}$	Coordinates matrix of all members of stops cluster (i)
$\mathbf{C}$	Connectivities matrix
$\mathbf{C}_b$	Connectivities matrix of the biggest node clusters

$\mathbf{c}_p$	Vector of instantaneous connectivities summation within period (p)
$\mathbf{DATA}_{bnn}$	Data matrix of policy weights and their corresponding number of data exchanges as evaluated throughout bayesian optimization using bayesian neural network regression
$\mathbf{DATA}_{brf}$	Data matrix of policy weights and their corresponding number of data exchanges as evaluated throughout bayesian optimization using batch-based random forest regression
$\mathbf{DATA}_{gp}$	Data matrix of policy weights and their corresponding number of data exchanges as evaluated throughout bayesian optimization using gaussian processes regression
$\mathbf{DATA}_{reg}$	Initial regression data matrix for bayesian optimization
$\mathbf{DATA}_{rf}$	Data matrix of policy weights and their corresponding number of data exchanges as evaluated throughout bayesian optimization using random forest regression
$\mathbf{DATA}_{rnd}$	Data matrix of random policy weights and their corresponding number of data exchanges as evaluated by the regression model used throughout bayesian optimization
$\mathbf{data}_{rnd}$	Vector of policy weights and their corresponding number of data exchanges of the current best random point as evaluated by the regression model used throughout bayesian optimization
$\mathbf{d}$	Vector of distances
$\mathbf{d}_{ne}^{tx}$	Vector of distances between the transmitting node (id^{tx}) and its neighbors
$\mathbf{d}_{sc_i}$	Vector of distances between the coordinates mean of stops cluster(i) and its cluster members
$\mathbf{F}^c$	Collected features matrix
$\mathbf{f}_i^c$	Collected feature column vector (i)
$\mathbf{F}^m$	Modified features matrix
$\mathbf{f}_i^m$	Modified feature column vector (i)
$\mathbf{F}_{bl}^m$	Modified features matrix for the same block
$\mathbf{F}^s$	Stops features matrix
$\mathbf{f}_i^s$	Stops feature column vector (i)
$\mathbf{F}^t$	Trips features matrix

$\mathbf{f}_i^t$	Trips feature column vector (i)
$\mathbf{F}^{bl}$	Same block features matrix
$\mathbf{f}_i^{bl}$	Same block feature column vector (i)
$\mathbf{F}_{st}^{bl}$	Same block features matrix with same arrival times
$\mathbf{F}^{cs}$	Converted stops features matrix
$\mathbf{f}_i^{cs}$	Coverted Stops feature column vector (i)
$\mathbf{F}^{n}$	Nodes features matrix
$\mathbf{f}_i^{n}$	Nodes feature column vector (i)
$\mathbf{F}^{sh}$	Shapes features matrix
$\mathbf{f}_i^{sh}$	Shapes feature column vector (i)
$\mathbf{F}^{st}$	Stop times features matrix
$\mathbf{f}_i^{st}$	Stop times feature column vector (i)
$\mathbf{id}_s^n$	Vector of node segment indices
$\mathbf{id}_s^{tx}$	Vector of indices of segments available at the transmitting node
$\mathbf{id}_{\mathbf{INT}_{true}}$	Vector of the normally-distributed true consumer positive-interest indices
$\mathbf{id}_{bl}$	Vector of vehicle block IDs
$\mathbf{id}_{cms}^{ne}$	Vector of common missing segment indices of neighboring nodes
$\mathbf{id}_{ds}$	Vector of the distributed service indices
$\mathbf{id}_{ene}^{tx}$	Vector of transmitting node neighbor indices being eliminated to control the transmitting node broadcasting range
$\mathbf{id}_{mps}$	Vector of the most popular service indices
$\mathbf{id}_{ms}^{ne}$	Vector of missing segment indices of a neighboring node
$\mathbf{ID}_{nc}$	Set of final member index matrices for all node clusters
$\mathbf{id}_{nc}$	Vector of node cluster indices
$\mathbf{ID}_{nc}^{b}$	Matrix of member IDs of the biggest node clusters
$\mathbf{ID}_{nc}^{nxt}$	Set of next iteration member index matrices for all node clusters
$\mathbf{ID}_{nc}^{prvs}$	Set of previous iteration member index matrices for all node clusters

$\mathbf{id}^{n}_{nene}$	Vector of indices of neighboring nodes around direct node neighbors
$\mathbf{id}^{n}_{ne}$	Vector of node neighbor indices
$\mathbf{id}^{tx}_{ne}$	Vector of neighboring node indices of the transmitting node
$\mathbf{id}_{os}$	Vector of optimal stop IDs
$\mathbf{id}^{hve}_{rows}$	Vector of all Collected features matrix row indices with high velocities
$\mathbf{id}_{rs}$	Vector of refined stop IDs
$\mathbf{id}_{sc_i}$	Vector of all member indices of stops cluster (i)
$\mathbf{id}_{sc}$	Vector of stop cluster IDs
$\mathbf{id}_{tr}$	Vector of all trip IDs
$\mathbf{id}^{bl}_{tr}$	Vector of trip IDs within the same block
$\mathbf{id}_{ts}$	Vector of segment indices being targeted for transmission
$\mathbf{id}^{ord}_{tx}$	Vector of node indices in order of transmission
$\mathbf{INT}_{avail}$	Available consumer interests matrix
$\mathbf{INT}_{rcmnd}$	Consumer interests matrix after making recommendations
$\mathbf{INT}_{true}$	True consumer interests matrix
$\mathbf{J}$	Jaccard similarities matrix
$\mathbf{LATS}$	Latitudes matrix
$\mathbf{LATS}_{bl}$	Synthetic block latitudes matrix
$\mathbf{LATS}_{tr}$	Synthetic trip latitudes matrix
$\mathbf{LONS}$	Longitudes matrix
$\mathbf{LONS}_{bl}$	Synthetic block longitudes matrix
$\mathbf{LONS}_{tr}$	Synthetic trip longitudes matrix
$\mathbf{Mdl}_{btr}$	Batch-based tree regression model used in bayesian optimization
$\mathbf{Mdl}_{gp}$	Gaussian processes regression model used in bayesian optimization
$\mathbf{Mdl}_{lat}$	Model of synthetic latitudes generated using linear regression
$\mathbf{Mdl}_{lin}$	Linear regression model used in bayesian optimization

$\mathbf{Mdl}_{lon}$	Model of synthetic longitudes generated using linear regression
$\mathbf{Mdl}_{nn}$	Neural network regression model used in bayesian optimization
$\mathbf{mdl}_{result}$	Vector of (latitude,longitude) pair generated
$\mathbf{Mdl}_{tr}$	Tree regression model used in bayesian optimization
$\mathbf{n}_{as}$	Vector of the different number of all node data segments
$\mathbf{n}_{ns}$	Vector of the number of segments for each node
$\mathbf{pop}_{segs}$	Vector of data segment popularities
$\mathbf{pop}_{services}^{avail}$	Vector of service popularities according to available consumer interests
$\mathbf{pop}_{services}^{rcmnd}$	Vector of service popularities after making recommendations
$\mathbf{pop}_{stops}$	Vector of refined stop popularities
$\mathbf{row}_{extra}$	Extra row generated
$\mathbf{SA}$	Matrix of segment allocations
$\mathbf{sizes}_{nc}$	Vector of node cluster sizes
$\mathbf{STAT}^{n}$	Node statuses matrix
$\mathbf{stat}_{i}^{n}$	Node status column vector (i)
$\mathbf{TR}_{gap}^{lats}$	Trajectory of latitudes for the gap between same-block trips
$\mathbf{TR}_{gap}^{lons}$	Trajectory of longitudes for the gap between same-block trips
$\mathbf{TR}_{sh}$	Shape trajectory matrix
$\mathbf{TR}_{tr}$	Trip trajectory matrix
$\mathbf{t}^{end}$	Period ending time indices vector
$\mathbf{t}^{start}$	Period starting time indices vector
$\mathbf{ut}$	Vector of node utilities given their feature values and the current policy weights
$\mathbf{u}_{avail}$	Vector of service upper popularity bounds according to available consumer interests
$\mathbf{u}_{rcmnd}$	Vector of service upper popularity bounds according to consumer interests after making recommendations

argfind	A function that finds the indices of matrix rows/columns such that they satisfy a certain condition
cluster	A function that clusters data using hierarchical clustering given certain conditions
distance	A function that gives the distance between two position pairs of (longitude,latitude) coordinates
fitgp	A function that fits a gaussian processes regression model to a set of data
fitlinear	A function that fits a linear regression model to a set of data
fitnn	A function that builds a neural network model with a certain number of layers and neurons in each layer
fittree	A function that fits a tree regression model to a set of data
length	A function that gives the longest dimension of a matrix
normrnd	A function that generates a predetermined number of normally-distributed random numbers according to a given mean and standard deviation
predict	A function that predicts the output and its standard deviation given a regression model and input data
randi	A function that generates a random integer number within a certain interval
repmat	A function that repeats matrix elements according to a given set of dimensions
sort	A function that sorts matrix rows in accordance to a specific column or set of columns and that is either in an ascending or a descending order
train	A function that trains a neural network model with a certain set of data
unique	A function that gives the unique rows of a matrix and their indices
c	Biggest node clusters index
cap_{net}	Network capacity in terms of the number of services that can be supported
cd_{min}	Minimum cumulative contact duration threshold
d	Distance

er_{min}	Minimum ratio between the number of exchanged data segments and the number of data segments distributed initially
id^{tx}	Index of the transmitting node
id_c^f	First group consumer index
id_c^l	Last group consumer index
id_s^f	First group service index
id_s^l	Last group service index
id_{cs}	Index of the chosen data segment for transmission
n_c	Number of consumers
n_g	Number of consumer geographical groups
n_i	Number of iterations
n_n	Number of all nodes
n_n^b	Number of nodes in the biggest nodes cluster
n_p	Number of time periods
n_s	Number of services
n_T	Total number of time steps
n_{as}^{max}	Maximum number of all distributable data segments given er_{min}
$n_{h_{max}}$	Maximum number of loop-free hops that a signal can traverse between two nodes within the same cluster given a cumulative contact duration threshold
n_{os}	Number of optimal stop nodes
n_{rnd}	Number of random samples generated throughout bayesian optimization
$n_{T_{day}}$	Total number of time steps in a day
$n_{T_{gap}}$	Number of time steps constituting the time gap between same-block trips
$n_{T_{period}}$	Number of time steps within a period
$n_{T_{rcmnd}}$	Number of recommendation time intervals
r_{br}	Maximum broadcasting range of a node

r_{earth}	Earth's radius
$ratio_{ui}$	Ratio of unknown consumer interests
sa_l	Index of the last data segment allocation
t^{end}	Ending time of the period under consideration
t^{start}	Starting time of the period under consideration
t^{bl}_{end}	Block end time
t^{bl}_{start}	Block start time
T_{travel}	Travel time
T_{wait}	Waiting time
ve_{max}	Maximum vehicle velocity
AI	Artificial Intelligence
BNN	Bayesian Neural Network
DSRC	Dedicated Short-Range Communications
FANET	Flying Ad-hoc NETwork
GP	Gaussain Processes
GTFS	General Transit Feed Specification
LTE	Long Term Evolution
MDP	Markov Decision Process
RAN	Radio Access Network
RF	Random Forest
RSU	Road Side Unit
UCB	Upper Confidence Bound
UPB	Upper Popularity Bound
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
V2X	Vehicle to Everything
VANET	Vehicular Ad-hoc NETwork

Chapter 1

Introduction

1.1 Motivation

Wireless networks provide services to consumers in a convenient and ubiquitous way. They distribute private content (e. g. emails) and public content (e. g. emergency announcements and social media posts). In fact, it has been found that 98% of Chinese internet consumers use wireless networks according to a recent report made by the China Internet Network Information Center [1]. This accounts for almost 800 million users up from 100 million users in 2008 which shows the convenience and ubiquity of these networks.

However, wireless networks suffer from traffic overload and a saturating frequency spectrum. For example, the US needs more than 350 MHz of additional frequency spectrum as of 2015 to meet the data demand of wireless networks in 2019 according to Bazelon *et. al.* [2]. This demand is expected to grow even further which means that more spectrum will be needed. In fact, wireless networks have generated about 10 exabytes/month of data traffic in 2017 and are expected to generate as high as 77 exabytes/month in 2022 according to the 2019 Visual Networking Index by Cisco [3]. This network overload is mainly driven by the traffic demand generated by applications generating mostly popular content including social media and video streaming sites. According to the 2019 Mobile Internet Phenomena report by Sandvine [4], Facebook properties (i.e. Facebook, Instagram, WhatsApp and Messenger) account for 20% of worldwide wireless network traffic while Youtube accounts alone for 35% of this traffic. In fact, it has been reported in the 2019 Visual Networking Index by Cisco [3] that 59% of wireless networks traffic is for videos as of 2017 and that this number will reach 79% of wireless networks traffic in 2022. This popular content demand is expected to grow even further in the near future with data-intensive applications such as virtual reality, connected cars and environmental sensing. For example, the wireless network data traffic generated by augmented reality and virtual reality will grow from 22.1 petabytes/month in 2017 to 254.4 petabytes/month in 2022 according to the 2019 Visual Networking Index by Cisco [3].

While this wireless network demand can be addressed indoors using technologies such as fixed Wi-Fi hotspots or small cells, it is still challenging when it comes to outdoor environments. This is mainly due to the fact that outdoor traffic demand comes from sources such as vehicles or consumers mobilizing all the time. A vast majority of those consumers mobilize in public transportation vehicles and stops while remaining outdoor consumers might be busy walking or driving. Therefore, a promising approach is to build a reliable VANET that matches this demand using public transportation vehicles and stops. Compared to regular vehicles, public transportation vehicles have by nature the following distinctive attributes:

- they have highly predictable mobility patterns given their fixed schedules,
- their real time locations can be shared in public without privacy concerns,
- their location distribution is uniform throughout city space and time while converging to consumer locations throughout the day, and
- they do not suffer from the selfish routing issue given that they are often administered by the same governmental entity which can enforce routing cooperation.

With these advantages in mind, a popular content distribution system is proposed to offload wireless network traffic by utilizing public transportation vehicles and stops. Having a reliable VANET implemented using public transportation vehicles and stops can form the backbone for other vehicle categories to communicate through including regular vehicles. This can lead eventually to a wide scale deployment of VANETs and break the network effect barrier that has prevented them so far from being deployed in a large scale.

The proposed system design is based on using AI-based techniques due to the high predictability nature of public transportation vehicles and stops. These techniques include the UPB collaborative and group-based recommender based on multi-armed bandits for content recommendation and bayesian optimization based on batch-based RF regression for content routing. These techniques are implemented after extensively preprocessing and processing the realistic mobility data of public transportation vehicles and stops. Such system design approach enables the deployment of VANETs in a gradual, reliable and scalable manner while utilizing the existing Vehicle-to-everything (V2X) technologies such as the Dedicated Short Range Communications (DSRC) technology.

On one hand, consumers will gain access to popular content after the proposed system makes the proper recommendations and routing while paying a fee in exchange for this service, receiving paid-for advertisements and/or receiving discounts from wireless network operators. Wireless network operators, on the other hand, can have their network expenses partially funded through targeted advertisements sent with the popular content. Moreover, they will see their operating costs decline due to the significant network offloading made by the proposed system. In fact, this system has allowed for a reliable, scalable and gradual VANET deployment while offloading as high as 95% of the outdoor wireless network load.

The market for applications which can utilize the proposed system is expanding quickly with great benefits to the economy. This is mainly due to the fact that the proposed system is in fact a 5G-enabling technology which allows for cheap, gradual and scalable network deployment. According to the IHS Markit modeling presented in [5], 5G will enable $US12.3 trillion of global economic activity by 2035 and will support 22 million jobs.

Compared to the proposed system, most other competing network technologies do not offer significant offloading to outdoor wireless networks. For example, the high cost of deploying and maintaining fixed outdoor Wi-Fi hotspots and small cells restrict their scalability and therefore their offloading potential. As for the mobile hotspots, they are built inside vehicles but still have their backhaul connected directly to the wireless network. While these mobile hotspots might lead to some minor offloading due to traffic aggregation but this is, by no means, significant. In general, these competing technologies fail to exploit the much cheaper VANET deployment cost using public transportation vehicles. They even fail to exploit the predictability demonstrated by these vehicles using a data-based approach. They are often distributed and based on domain knowledge assumptions which might not fit all scenarios.

1.2 Research Objectives

The main objectives of this thesis are as follows:

- Survey previous content distribution attempts to understand their shortcomings and define areas of improvement,
- Design the proposed popular content distribution system,
- Analyze the networking potential of public transportation vehicles and stops using a novel data preprocessing and processing approach. The data used represents the realistic transportation service offered by the Grand River Transit authority throughout the Region of Waterloo, ON, Canada,
- Develop recommendation and routing schemes which take advantage of public transportation vehicles and stops,
- Conduct experiments to study the performance of the proposed content recommender to demonstrate its versatility under different consumer interest and network capacity scenarios, and
- Test the proposed content routing mechanism to show the significant improvement offered in terms of offloading outdoor wireless networks.

1.3 Thesis Organization

Chapter 1 presents the motivations of this work as well as the research objectives and thesis organization. Chapter 2 provides background material, introduces popular content distribution systems and the different techniques used in building these systems. It presents a survey of previous mobility analysis studies in relation to VANETs. It also presents a comprehensive survey of previous content distribution systems, starting with recommendation systems and then content routing protocols providing multicast and broadcast services. In Chapter 3, the problem formulation of content distribution as well as the overall structure of the proposed system are introduced. Chapter 3 also discusses the details and assumptions of the case study adopted which represents the Grand River Transit bus service offered throughout the Region of Waterloo, Ontario, Canada. Chapter 4 presents the proposed system operations concerning the mobility data analysis of public transportation vehicles and stops. Chapter 5 introduces the experimental study conducted to design the content recommender. Chapter 6 presents the proposed system operations concerning content routing. Chapter 7 concludes the thesis and outlines future research directions.

Chapter 2

Background and Literature Review

2.1 Overview

Popular content includes any popular and publishable material generated by any application or site ranging from social media sites such as Facebook and Twitter to non-real-time video streaming sites such as Youtube and Netflix. It does also include data generated outdoor by vehicles related to applications such as infotainment and environmental sensing. Offloading such content from outdoor wireless networks would have a significant impact on the overall network performance. This can be done via content distribution using public transportation vehicles and stops given their predictable mobility patterns, their publicly-shared real-time locations, their uniform location distribution throughout city space and time and the possibility of enforcing cooperation between them given the single administration authority they usually have.

Operations made by these content distribution systems can be classified into: content recommendation and content routing. Content recommendation operations are responsible for choosing the proper content based on consumer service interests whereas content routing operations are responsible for routing services content to consumers.

The recommender making the recommendation operations starts with some knowledge about consumer interests and then gradually and based on its interactions with the consumers, interests are updated towards the true consumer interests. These interactions can take several forms depending on the exploration-exploitation trade-off made by the recommender. They can also make assumptions about consumer interests in order to filter service recommendations. This can be based on consumer-consumer interest-profile similarities or based on content-content feature similarities. Moreover, they can divide consumers into groups based on their locations such that more personalized interest recommendations are made.

After making the recommendations, popular content is routed to consumers. This can take several forms using different routing actions. An intelligent routing mechanism should adapt between these routing actions so that content is routed quickly and reliably. This adaptation can either be based on domain knowledge about VANET routing or be AI-based such that adaptations are more intelligent and flexible in terms of addressing different routing scenarios.

In the next sections, more details about these two main content distribution classes of operations are introduced, namely: content recommendation and content routing. Previous content recommendation and routing studies are also discussed by briefly describing them and highlighting their key distinctive benefits and drawbacks. However and since the proposed content distribution system is driven by data analysis as it will be seen in Chapter 4, previous mobility data analysis studies of VANETs are reviewed before surveying previous content recommendation and routing studies.

2.2 Content Recommendation

Content recommendation has three main elements to it; the first is concerned with AI-based consumer interactions, the second is with the way filtering is made in terms of determining unknown service interests based on either consumer-consumer interest similarities or content-content feature similarities and the third is concerned with the possibility of identifying location-based consumer groups which can allow for more personalized recommendations. In what follows, each one of these elements is introduced.

2.2.1 Artificial Intelligence-based Consumer Interactions

The problem of recommending the truly most popular services to consumers in the smallest number of interactions is similar to the well-known AI-problem of multi-armed bandits. This problem has the following commonly used search techniques: Greedy search, ϵ-greedy search, Decaying ϵ-greedy search and Upper Confidence Bound (UCB) search. These search techniques are introduced as follows.

Greedy Search

In Greedy search, the most popular services based on current consumer interest profiles are recommended at each time step. This greedy approach of interaction does not account for the fact that service popularities based on current consumer interest profiles might be misleading and lacking sufficient knowledge about the true consumer interest profiles. This search approach is in essence exploiting current knowledge about consumer interest profiles while not exploring the unknown interest search space.

ϵ-greedy Search

In ϵ-greedy search, the disadvantage of greedy search is overcome by exploring unknown service popularities with a constant probability of ϵ. This can be done by recommending less popular services randomly with an equal probability while acting greedily with the remaining probability of $(1 - \epsilon)$ at each time step. This approach of interaction does guarantee that unknown service popularities will eventually be explored but it does also ignore the fact that the search space of unknown service popularities shrinks over time; having a fixed exploration probability of ϵ and a fixed exploitation probability of $(1 - \epsilon)$ does lead to this.

Decaying ϵ-greedy Search

In Decaying ϵ-greedy search, the shortcomings of ϵ-greedy search is overcome by exploring unknown service popularity search space with a decaying ϵ as time steps progress. This approach of interaction does account for the fact that unknown interests shrink over time. However, the optimal schedule used for ϵ-decays is an open research question with many possibilities. A commonly used schedule is to have $\epsilon = \frac{1}{t}$ where t is the interaction time index; this schedule allows for a decaying ϵ that reaches eventually 0 as t approaches ∞.

Upper Confidence Bound Search

In UCB search, the UCBs of service popularities are computed and then the service with the highest UCB is chosen greedily at each time step. These bounds are set in a way that guarantees the service with the highest popularity to be chosen eventually. If the UCB are set to include the number of all remaining unknown consumer interests, then this approach of interaction guarantees that the most popular service is eventually chosen. The main idea here is to be "optimistic in the face of uncertainty".

2.2.2 Content Filtering

Assumptions are made when filtering service recommendations to consumers at each time step. There are two ways to make these assumptions; it is by either assuming that consumers with similar interest profiles like the same unknown-interest services or assuming that services with similar content features are liked by the same consumers.

Consumer-Consumer Interest Similarities

There are several ways to measure consumer-consumer interest similarities. One way is to measure Jaccard similarities given that consumer interests are using a binary representation with the value 1 denoting a liked service and 0, otherwise. Unknown services, however, can be given the value NaN in order to ensure that computations of Jaccard similarities are not distorted. The unknown interests of the current consumer are decided collaboratively based on the known interests of a consumer that has the

highest Jaccard similarity to the current consumer. Therefore, this kind of filtering is commonly known as "Collaborative filtering".

Content-Content Feature Similarities

After deciding which features to use when representing a certain content, distances between content representations are measured using conventional clustering techniques. Contents within the same cluster are therefore similar and should be liked by a consumer if s/he likes one of them and vice versa, otherwise. This shows clearly that deciding which features to use would have a significant impact on deciding which contents are similar and which are not. Therefore, using content-content similarities is controversial and might not be suitable in many scenarios.

2.2.3 Consumer Grouping

Identifying consumer location-groups can make popular content recommendations more personalized for consumers. This is due to the fact that consumer interest profiles may differ depending on which location the consumer is at. Exploiting such location-based differences between consumer group interests can make recommendations more personalized. In fact, it is guaranteed that it will not make things worse. Such location-based interest differences can be imagined in scenarios where consumers are distributed between different work environments (e.g. students in a university campus vs. employees in a downtown business area).

2.3 Content Routing

After making recommendations to consumers, content has to be routed quickly and reliably. As mentioned before, there are several routing actions which can be taken using a certain routing policy or strategy. The decision on how to adapt between these routing actions can be made either using a hand-crafted algorithm based on our domain knowledge about VANET routing or it can be made automatically using an AI-based agent that is capable of exploring and exploiting. In what follows, the different VANET routing actions are introduced followed by the two main routing adaptation mechanisms, namely: domain knowledge-based mechanism and AI-based mechanism.

2.3.1 Routing Actions

Routing actions are the different actions which can be taken to route content data after being recommended. These routing actions are explained in what follows.

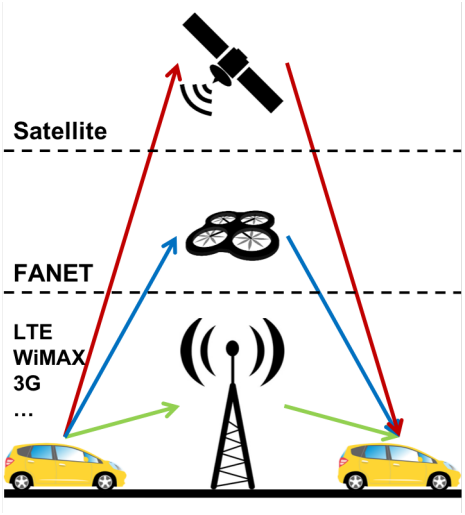


Figure 2.1: Multi-tier routing

Multi-tier Routing

Multi-tier routing takes place between different network tiers with different radio access technologies (see Figure 2.1). Ready infrastructure, such as cellular networks, can be utilized for this purpose especially in sparse to no Vehicle-to-Vehicle (V2V) communication conditions. Recent technologies, such as Long-Term-Evolution (LTE), can provide very low latency communication between vehicles enabling even hard-delay safety services. Other special tiers include satellites and Flying Ad-hoc NETworks (FANETs).

Delay-tolerant Routing

Delay-tolerant routing overcomes big voids by storing, carrying and forwarding data packets if no other vehicle or infrastructure is encountered directly (see Figure 2.2). This routing action overcomes network disconnections frequently encountered in sparse VANET deployments. This can also be done by storing, carrying and forwarding data packets to mules, stationary vehicles or stationary infrastructure until destination is reached. In fact, it has been found in [6] that even a disconnection time as short as 30 seconds can dramatically affect routing protocols based only on routing through direct paths. Therefore, enabling delay-tolerant routing might be a necessity rather than being an option.

Direct Routing

Direct routing might be the most intuitive V2V routing action. It can be done by acting greedily and routing data packets through the shortest path (see Figure 2.3). This path, under high vehicle deployment densities, can be found and utilized directly to route packets. However, this is not usually the case which makes this routing action unreliable alone in the other much more frequent VANET deployment scenarios.

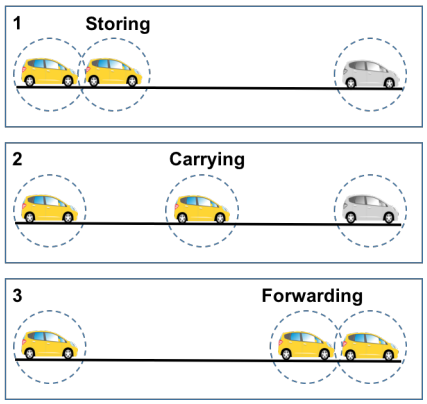


Figure 2.2: Delay-tolerant routing

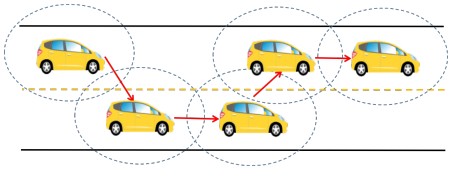


Figure 2.3: Direct routing

Cluster-based Routing

Cluster-based routing is made to enhance stability in high density VANET deployments. This is done by clustering vehicles around cluster heads with special mobility characteristics and/or special interface capabilities. This routing action utilizes bandwidth more efficiently by enabling cluster members to communicate directly within the same cluster and via cluster heads between different clusters. On the other hand, cluster heads are allowed to communicate directly between themselves as shown in Figure 2.4.

Cross-layer Optimized Routing

Cross-layer optimized routing is an advanced routing action that takes into consideration information coming from other network layers. This information is

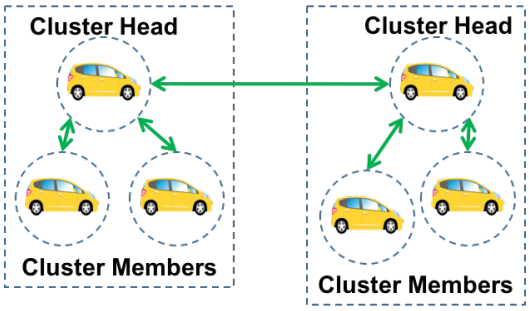


Figure 2.4: Cluster-based routing

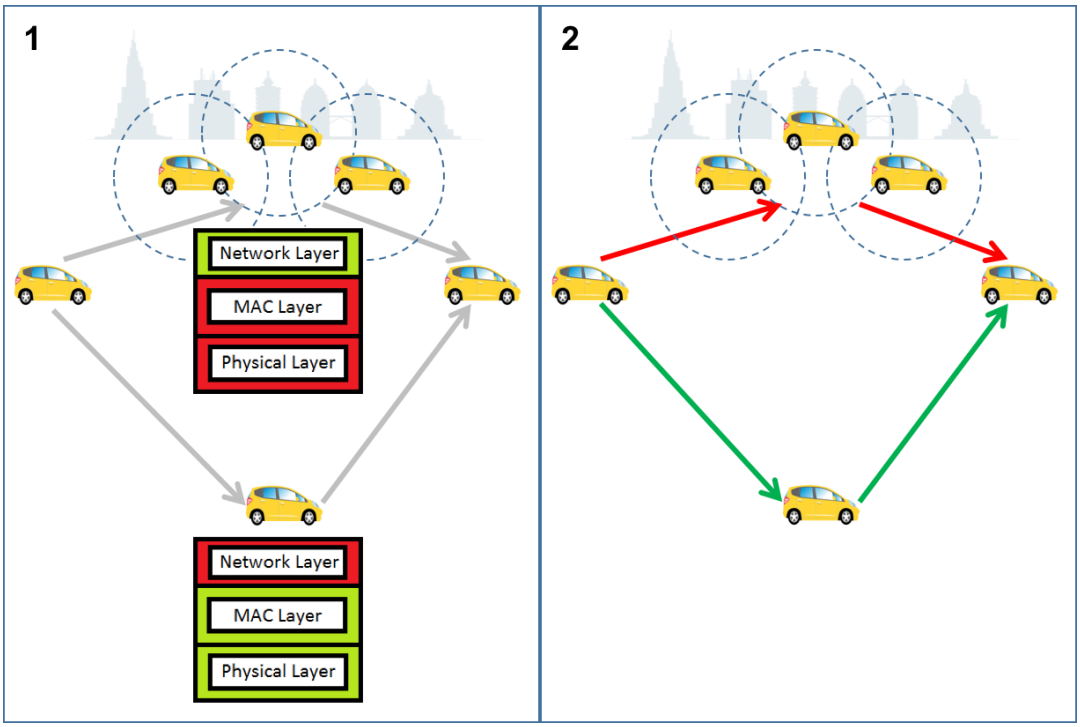


Figure 2.5: Cross-layer optimized routing

exchanged in order to forward data packets to the best route as seen by the different layers. For example in Figure 2.5, the source vehicle is making a decision between routing traffic through the upper path or the lower. The upper path is shorter but goes through a busier area where a high contention level is expected. This is illustrated in the figure by the green colored network layer and the red colored MAC and physical layers. On the other hand, the lower path is longer but experiences a much less contention level as indicated by the red colored network layer and the green colored MAC and physical layers. As shown, the final decision made using cross-layer optimized routing is to route traffic through the lower path.

Terminated Routing

Terminated routing is done by stopping the forwarding of packets after their Time-To-Live (TTL) period expires (see Figure 2.6) or after a certain number of hops is passed. This action can prevent packets form reaching destinations of no relevance or propagating through an excessive number of hops. Destinations of no relevance include, for instance, vehicles outside an accident area whereas propagation through an excessive number of hops can result from routing loops.

Expedited Routing

Expedited routing is done by anticipating vehicle demanded packets and routing them beforehand while current network conditions still permit such routing. For example, packets can be routed expeditiously to a vehicle heading to a tunnel before it reaches

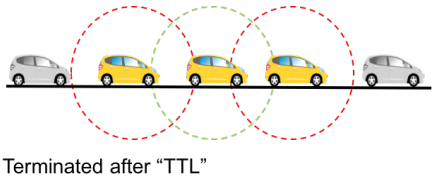


Figure 2.6: Terminated routing

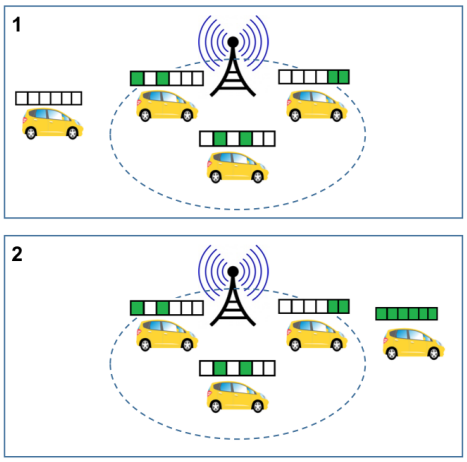


Figure 2.7: Expedited routing

the tunnel and suffer from the bad coverage inside. Another example is when chunks of popular content are distributed expeditiously between vehicles in the same area in order to allow incoming vehicles to find and collect this content (see Figure 2.7).

Splitted Routing

Splitted routing is done by forwarding data packets through different paths. For example, traffic is splitted into x and $(1 - x)$ data portions between the two paths shown in Figure 2.8. This action might be necessary considering that high-velocity vehicles might never have the chance to route all packets through a single path. In fact, it has been found in [7] that there are 40 seconds of effective communication time between vehicles crossing at 20 km/h, 15 seconds between vehicles crossing at 40 km/h and only 11 seconds between vehicles crossing at 60 km/h. These short communication windows can result in goodputs as low as 80 KB at 60 km/h. A source high-velocity vehicle might overcome this by dividing its packets between different passing vehicles heading towards the same destination vehicle using splitted routing.

Infrastructure antennas might also not have enough time to transmit packets to a destination vehicle under its coverage. Given a vehicle with a velocity of 80 km/h and an infrastructure antenna with 500 meters coverage area radius, a 1 minute period of transmission might be the only available duration for this antenna to communicate with this vehicle. Therefore, the infrastructure might also split packets between vehicles surrounding or heading towards the destination vehicle (see Figure 2.7).

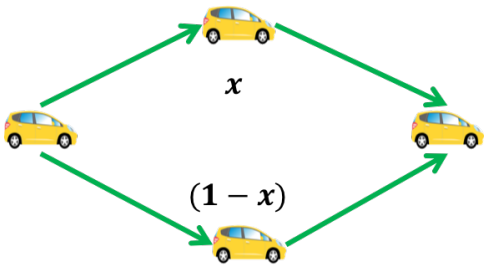


Figure 2.8: Splitted routing

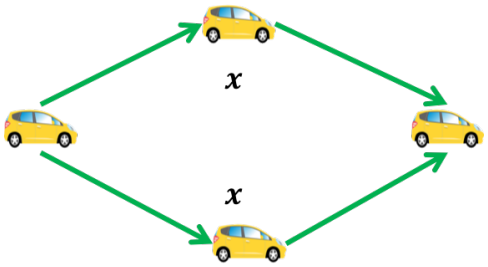


Figure 2.9: Redundant routing

Redundant Routing

Redundant routing is done by sending redundant copies of the same data packet using the same path or through different paths in an effort to maximize packet chances of reaching destination. This routing action might be necessary under harsh channel conditions. For example in Figure 2.9, the source vehicle is routing redundant copies of message (x) through two different paths to enhance reliability.

2.3.2 Routing Adaptation

Domain Knowledge-based

There have been many attempts to craft routing protocols for content routing in VANETs based on domain knowledge. Such protocols are often based on predefined hyper parameters which control when and where to switch between the different routing actions. Their main drawback is their inability to fit all scenarios due to their inflexibility.

Artificial Intelligence-based

Fewer routing protocols based on this adaptation mechanism have been encountered so far. Most protocols encountered follow the domain knowledge-based adaptation. However, using AI-based routing adaptation agents can itself follow different approaches. A common approach is to use reinforcement learning after building an explicit environment model of states and actions. This can be based on our knowledge or based on the agent's experience after interacting with the

environment. After building such a model, there are different mathematical techniques to solve the resulting Markov Decision Process (MDP). One of the most widely used techniques is dynamic programming. The main drawback of this approach of learning over the state-action space is its limited scalability. This is especially true in situations where we have a continuous state-action space which is the case in many interesting applications of reinforcement learning.

Bayesian optimization, on the other hand, can overcome the scalability issue of reinforcement learning by dealing with the environment as a "black-box" without the need to build a model. There is also no need to find the derivative of the policy using bayesian optimization which acts directly over the policy search space. This is done by following the bayesian approach assuming some prior and updating it based on environment interactions towards a posterior. This can be done by first interacting with the environment using random policy weights for a certain number of iterations. After that, a regression model over the resulting data is employed in order to pick up the next policy weights according to a certain acquisition function. Such weights can be chosen by sampling the resulting regression model multiple times and then choosing the weights according to the acquisition function. The resulting weights are used to interact with the environment again and the resulting reward is used to update the regression model in the next iteration. This method continues until the total reward converges to the optimal set of policy weights and reward.

However, there are many variables in bayesian optimization. One of them is the acquisition function to use. One of the most well-known acquisition functions is the UCB acquisition function which is based on choosing greedily the maximum UCB given the current data. These UCBs can be computed by summing the means and standard deviations of the resulting policy rewards.

Another bayesian optimization variable is the number of data points used when making regressions. One possibility is to choose all the data which can slow-down the bayesian optimization. Another more efficient approach is to choose batches of regression data where the search space is most "interesting".

Choosing the regression model to use in bayesian optimization iterations is itself another important decision to make. The most common regression model used is Gaussian Processes (GP) but other regression models can also be used including: Random Forest (RF) and Bayesian Neural Network (BNN). The best regression model to use is application-specific and there is no "one size fits all" model.

2.4 Mobility Analysis Studies

The mobility data analysis study discussed in Chapter 4, represents the first detailed study of its kind. It highlights the distinctive and promising networking potential of public transportation vehicles and stops. It also presents a detailed data analysis study using a realistic case study dataset and a novel data preprocessing and processing approach. In what follows, a survey of previous mobility analysis studies is introduced to highlight the novelty of the data analysis study conducted in Chapter 4.

Starting with the work of S. Uppoor *et. al.* in [8], [9] and [10]. This work presents a data analysis study of urban mobility data representing regular vehicles throughout the city of Cologne, Germany. The synthetic dataset used has been prepared for studies involving ad-hoc network protocols evaluation. Authors claim that their dataset captures both microscopic and macroscopic mobility aspects of vehicular movement. They show that other datasets lack such scale and realism which results in overestimating VANET protocols.

Using the same synthetic dataset, S. Uppoor *et. al.* in [11] present probability laws characterizing vehicle mobility between Radio Access Network (RAN) cells. Several networking aspects are evaluated including cell inter-arrival times, cell-residence times and vehicular contact times.

H. Zhu *et. al.* in both [12] and [13] present a large scale data analysis study using taxi mobility data throughout the city of Shanghai, China. Their study reveals that inter-contact times between vehicles follow an exponential-like tail distribution.

In [14], K. Zhao *et. al.* use three large scale taxi datasets from the cities of Rome, San Francisco and Beijing. They propose a quad-tree technique to divide these city areas into regions based on the number of taxi visits. These regions are then associated with one of four functions: residential, work, entertainment or other. Efficient delay-tolerant networking solutions can be developed based on these functional regions.

None of the aforementioned studies offers a mobility data analysis of public transportation vehicles and stops for networking purposes like the one presented in Chapter 4. In [8], [9] and [10], the main focus is on generating a large scale synthetic dataset with a high degree of realism for regular vehicles. The comparison of VANET protocol performances under this dataset and other less-realistic datasets is only used to demonstrate the importance of such realism in avoiding overoptimistic protocol evaluations.

The focus in [11] is only on characterizing regular vehicle movements within and between RAN cells. The focus of both works presented in [12] and [13] is only on revealing the distribution of inter-contact times between taxi vehicles in urban environments. And finally, the focus in [14] is only about identifying functional regions in order to be utilized by delay-tolerant networking solutions for regular vehicles.

2.5 Content Recommendation Studies

MH. Park *et. al.* in [15] propose a restaurant recommendation system for a group of mobile consumers. This system makes recommendations to consumers after integrating their preferences using the Analytic Hierarchy Process (AHP). The preference profile of an individual consumer is modeled using bayesian networks given the uncertainty usually associated with it.

A multimedia recommendation system for a group of consumers in vehicular networks is proposed by Y. Zhiwen *et. al.* in [16]. The system creates a common consumer profile based on minimizing the total distance between the different consumer profiles within the group. The goal is to use this common profile in order

to recommend multimedia content that is of interest to most consumers.

Y. Ge *et. al.* propose in [17] a recommendation system for mobile consumers driving taxis. This system recommends pick up points and/or parking spots throughout the city for these taxi drivers. It does also recommend a set of energy efficient routes for their trips. These recommendations are based on clustering of historic data of taxi drivers with high returns. The objective here is to maximize the revenue per energy use of these taxis.

T. Ruotsalo *et. al.* in [18] propose a context-aware recommender for on-site tourists. The system is based on retrieving content to tourists using a clustering technique based on their interest profiles and physical locations. Another travel recommendation system for tourists is proposed by M. Kenteris *et. al* in [19]. This system utilizes an on-site wireless sensor network that collects tourist ratings from points of attraction. These ratings are then given a different weight compared to web ratings made by off-site tourists. The notion here is to use all of these ratings to make travel recommendations to new tourists using collaborative filtering. A third travel recommendation system is proposed by WS. Yang *et. al.* in [20]. The system allows tourists to get travel recommendations from nearby tourists using peer-to-peer communication. This should be favorable given the saving of roaming costs incurred compared to the cost of using wireless communication. Relying on peer-to-peer communication should result in travel recommendations which are not distorted by ratings coming from off-site tourists who might have different interests.

T. Li *et. al.* in [21] propose a recommender system that utilizes vehicles for marketing purposes. Vehicles are chosen based on their mobility patterns within regions of high benefits to marketers. They are also chosen based on the city area they cover throughout their movement which can constitute a future benefit for marketers. In shopping environments, WS. Yang *et. al.* propose in [22] a location-aware recommender system for consumers. The system matches the consumer interests to the on-site vendor offers and promotions. The consumer interests and preferences are complemented with their position and history in order to make location-aware recommendations. Several attributes of shopping environments are also considered in the recommendations.

All of the above studies propose recommendation systems for popular content ranging from restaurant, multimedia and taxi recommendations to travel and marketing recommendations. Many of them rely on distance-based clustering techniques to make recommendations. Compared to my proposed content recommender, the choice of features used throughout these studies is usually controversial. All of these studies lack the intelligent search mechanisms needed to interact with the consumers efficiently and no study has proposed a recommender that exploits the consumer groups in order to make more personalized recommendations.

2.6 Content Routing Studies

There are two protocol categories serving content routing; they provide either multicast services which route content to a specific group of consumers or broadcast services which route content to all neighboring consumers. In the next subsections, previous studies related to each category are presented.

2.6.1 Multicast Services

This subsection presents routing protocols supporting multicast services. They are compared in light of the routing actions implemented. To the best of my knowledge, no Vehicle-to-Infrastructure (V2I) or V2X based routing protocols supporting multicast services have been proposed. Therefore, only V2V based protocols are presented in what follows.

V2V-based Routing

Protocols proposed by L. Briesemeister *et. al.* in [23] and M. Guo *et. al.* in [24] support delay-tolerant routing and terminated routing. The Role-based protocol supporting multicast service is proposed by L. Briesemeister *et. al.* in [23]. Using this protocol, a broken vehicle multicast warning messages to neighboring vehicles in a highway scenario including approaching opposite-direction and same-direction vehicles. Message multicasting is terminated after passing a maximum number of hops and vehicles incapable of braking are not within the multicast group. In [24], the V3 video streaming architecture is proposed by M. Guo *et. al.* Vehicles multicast traffic inquiries to other vehicles in a targeted destination area. This triggers onboard cameras to video stream destination area traffic conditions. Until suitable nodes are available, multicast requests can wait for a predetermined waiting time at intermediate nodes before reaching destination area.

The Inter Vehicles Geocast (IVG) protocol is proposed by A. Bachir *et. al.* in [25]. This protocol divides highway vehicles into vehicles within and outside "risk areas". Risk areas include same-direction vehicles behind the broken vehicle and opposite-direction vehicles in front of the broken vehicle. Risk area vehicles rebroadcast broken vehicle alarm messages after a deferring time that is inversely proportional to their distance from the broken vehicle.

L. Briesemeister *et. al.* in [26] and M. Kihl *et. al.* in [27] propose protocols supporting cross-layer optimized routing and terminated routing. L. Briesemeister *et. al.* propose in [26] a Warning dissemination protocol for highway vehicles within the Zone of Relevance (ZOR). This includes approaching vehicles from both directions in the case of undivided roads and only same direction back vehicles in the case of divided roads. Vehicles capable of braking are only warned. Within ZOR, warning messages are resent after a waiting time that is inversely proportional to the distance separating a vehicle from source vehicle while being terminated after a maximum number of hops is reached. In [27], the RObust VEhicular Routing (ROVER) protocol is proposed by M. Kihl *et. al.* This protocol

	Protocol(s)			
	[23, 24]	[25]	[26, 27]	[28]
Multi-tier Routing				
Delay-tolerant Routing	✓			
Direct Routing	✓	✓	✓	✓
Cluster-based Routing				
Cross-layer Optimized Routing		✓	✓	
Terminated Routing	✓		✓	✓
Expedited Routing				
Splitted Routing				
Redundant Routing				

Table 2.1: Multicast Services - V2V-based protocols

builds on demand multicasting trees within ZOR. These trees enable vehicles to meet QoS requirements of different applications which are capable of defining which neighboring vehicles should be within ZOR.

T. Kosch *et. al.* discuss in [28] information dissemination in VANETs. They propose a geo-casting technique based on Multicasting messages to areas where the "Rate of Interest" is higher than a threshold. The Rate of Interest represents the percentage of vehicles interested in a certain message at a certain area.

Table 2.1 summarizes the above V2V based protocols supporting multicast services in terms of the routing actions adopted.

2.6.2 Broadcast Services

This subsection presents routing protocols supporting broadcast services. They are compared in light of the routing actions implemented. V2I, V2V and V2X based routing protocols providing these services are all covered in what follows.

V2I-based Routing

G. Korkmaz *et. al.* in [29] and [30] propose protocols supporting multi-tier routing and cross-layer optimized routing. The cross-layer Controlled Vehicular Internet Access (CVIA) protocol is proposed in [29]. This protocol switches neighboring segments between active and inactive phases in order to avoid collisions resulting from neighboring segments transmitting simultaneously. In [30], the same authors enhance CVIA by taking QoS requirements into consideration (CVIA-QoS). They propose scheduling and admission control techniques to prioritize packets and meet these QoS requirements.

V2V-based Routing

The Distributed Vehicular broad-CAST (DV-CAST) protocol is proposed by O. K. Tonguz *et. al.* in [31]. Each vehicle has a "Region Of Interest" that includes

a maximum number of one-hop neighboring vehicles. These neighboring vehicles are classified into same direction front vehicles, same direction back vehicles and opposite direction vehicles. Based on the classification, vehicles decide whether to broadcast in the connected state or to store, carry and forward in the disconnected state. The broadcasting technique adopts suppression using the well-known weighted p-persistence, slotted 1-persistence or slotted p-persistence technique. In all cases, packets are discarded after packet timer expires.

Protocols proposed in [32] to [34] support cross-layer optimized routing and terminated routing. A Cross-layer optimized routing protocol supporting broadcast service is proposed by S. Eichler *et. al.* in [32]. It relies on the application layer informing the MAC layer which packets should be given a higher priority. The application layer prioritizes packets based on their age, time of the day, type, neighboring vehicle directions, velocity, distance and number of connected vehicles. The MAC layer sets different priorities by adjusting contention windows, channel access timers and persistence factors. L. Briesemeister *et. al.* propose in [33] a routing scheme in which neighboring vehicles rebroadcast after a waiting time that gets shorter as the distance from source vehicle gets longer. A maximum number of hops is set. In [34], the Fair Data Dissemination (FairDD) protocol is proposed by R. S. Schwartz *et. al.* Vehicles broadcast Hello messages conveying mobility and data context information. Mobility context spans: direction, velocity, destination and mobility history. Whereas, data context spans: message age, message geographic region and message priority. Considering the resulting utilities, data messages are prioritized using Nash bargaining game before waiting for a random period and then broadcasting. Low priority data messages can be suppressed to avoid network congestions.

In [35], M. T. Sun *et. al.* propose the Vector-based TRAck DEtection (V-TRADE) protocol and the History-enhanced V-TRADE (HV-TRADE) protocol. With these protocols, vehicles classify their neighbors according to their movement direction and location. Then, border vehicles rebroadcast messages while accounting for message ID and remaining TTL. This may result in erroneous neighbor vehicle classifications especially in unconventional roads like roads with big curves. Therefore, authors enhance V-TRADE protocol by comparing history rectangles of different nodes (HV-TRADE). Both protocols utilize bandwidth more efficiently compared to flooding while offering a small sacrifice in terms of reachability.

The Location Based Broadcasting (LBB) protocol is proposed by X. Qing *et. al.* in [36]. Vehicles send redundant message copies within the useful message lifetime. Specific redundant messages are chosen by flipping an unfair coin while considering that large number of repetitive packets can lead to excessive collisions and low number of repetitive packets can lead to transmission failures.

Protocols proposed in [37] to [40] support cross-layer optimized routing. In [37], A. Nasri *et. al.* propose a cross-layer optimized routing protocol supporting broadcast service in which vehicles defer packet rebroadcasting according to their distance from the source node. In [38], the MultiHop Vehicular Broadcast (MHVB) protocol is proposed by T. Osafune *et. al.* Initially, vehicles broadcast after some waiting time that is inversely proportional to the distance from source vehicle.

Then, a "Backfire Algorithm" is proposed to suppress unnecessary packets and retransmit through farther nodes. A traffic congestion detection algorithm based on vehicles sensing is also proposed. Broadcast waiting times are adjusted based on this algorithm findings. In [39], M. N. Mariyasagayam *et. al.* propose the Enhanced MHVB protocol which adopts sectoral backfire. In sectoral backfire, an angle is added to control the backfire area. In addition, dynamic scheduling is adopted so that earlier retransmissions take place at vehicles located more than 200 meters away from source vehicle. In [40], the Cross Layer Broadcast Protocol (CLBP) is proposed by B. Yuanguo *et. al.* Vehicles with a larger distance, better channel conditions and a smaller relative velocity difference are chosen as relays. A novel composite relaying metric is used for this purpose incorporating vehicle relative velocity, distance from source vehicle, communication channel signal to noise ratio and packet error rate.

The BROADCOMM protocol is proposed by M. Durresi *et. al.* in [41]. It divides highway vehicles into virtual cells. Each virtual cell has a cell reflector based on its: proximity to the cell center, velocity and direction of movement. Cell members communicate with neighboring cell members via these cell reflectors. Cell reflectors prioritize cell member messages and update cell members with their mutual locations. The update interval is based on how much mobility is exhibited by the vehicles.

V2X-based Routing

Protocols proposed in both [42] and [43] support cross-layer optimized routing. The Urban Multi-hop Broadcasting (UMB) protocol is proposed by G. Korkmaz *et. al.* in [42]. This protocol has two phases: directional and intersection broadcast. In the directional broadcast phase, the farthest node is selected without a prior knowledge of locations or IDs. In the intersection broadcast phase, repeaters rebroadcast into road segments. This way, UMB addresses broadcast storms and hidden terminal problems. In [43], the Game based Routing algorithm for Congestion Control of Multimedia transmission (GRCCM) is proposed by D. Di *et. al.* It classifies vehicles into gateway vehicles or high quality nodes and non-gateway vehicles or low quality nodes. It disallows vehicles from excessively choosing gateway vehicles and causing congestions. Vehicles minimize their cost, represented by the product of load and latency, by finding out the Nash high and low quality nodes selection probabilities.

In [44], W. Saad *et. al.* propose a Coalition formation game between Road Side Units (RSUs). This enables RSUs to coordinate sending diversified data classes to passing vehicles. Each RSU seeks maximum utility by increasing revenue acquired from passing vehicles and decreasing coordination cost. This leads to changing coalitions in response to environmental changes until Nash-stable partitions are reached.

Protocols proposed in both [45] and [46] support cross-layer optimized routing, expedited routing and splitted routing. In [45], B. Shrestha *et. al.* propose a solution for RSUs to maximize their utility by prioritizing vehicle packets using a simplistic heuristic technique. Vehicles exchange missing packets afterwards in a fair bargaining game. Exchanging missing packets starts with the vehicle that has

	Protocol(s)										
	[29], [30]	[31]	[32]-[34]	[35]	[36]	[37]-[40]	[41]	[42], [43]	[44]	[45], [46]	[47]
V2I	✓							✓	✓	✓	✓
V2V		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Multi-tier Routing	✓							✓	✓	✓	✓
Delay-tolerant Routing		✓									
Direct Routing		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cluster-based Routing							✓				
Cross-layer Optimized Routing	✓	✓	✓		✓	✓	✓	✓		✓	
Terminated Routing		✓	✓	✓	✓						
Expedited Routing									✓	✓	✓
Splitted Routing									✓	✓	✓
Redundant Routing					✓						

Table 2.2: Broadcast Services - V2I, V2V and V2X-based protocols

the best communication channel and continues until all vehicle-pairs have the same packets or the communication channel becomes bad. In [46], a mechanism for RSU Popular Content Distribution (PCD) is proposed by T. Wang *et. al.* Initially, RSUs spread all packets between vehicles. Then, vehicles exploit cognitive radio for V2V communication. Each vehicle decides periodically what to receive and to which other vehicle packets are forwarded until reaching "best response". This results in all vehicles forming a Nash directed graph. To guarantee this graph convergence and avoid cycles, recent vehicles chosen are stored.

T. Wang *et. al.* propose in [47] a coalition formation game for the RSU popular content distribution problem. RSUs send different data packets to passing vehicles which in turn exchange remaining packets after forming coalitions. Authors claim that Nash-stable coalitions are reached using their approach.

Table 2.2 summarizes the routing protocols supporting broadcast services and their routing actions.

2.7 Summary

This chapter has presented background material about content recommendation and routing. It does so by discussing recommendation search approaches used to interact with the consumers based on multi-armed bandits. Assumptions made to filter content recommendations have also been introduced in addition to the rule of consumers grouping in making recommendations more personalized. The different routing actions have been explained in addition to the two main adaptation mechanisms based on domain knowledge and AI. Previous mobility analysis studies, content recommendation and routing studies have all been surveyed by the end of the chapter.

It has been found that the focus of previous mobility analysis studies has not been on evaluating the networking potential of public transportation vehicles and stops. It has also been found that previous popular content recommenders are not collaborative-based and lack the intelligent search mechanisms and/or the location-based clustering needed to interact efficiently with the consumers. In addition, it has been found that most routing protocols are designed based on domain knowledge. This applies for protocols supporting multicast and broadcast services. Criteria used to make switching between the different routing actions in these protocols are usually based on the vehicle direction, velocity and acceleration. Compared to AI-based protocols, domain knowledge based protocols are inflexible and might be suitable for only a specific set of scenarios. This is due to the fact that certain heuristics are used which are quite specific for their corresponding scenarios. Overall, the routing protocols surveyed support a limited number of routing actions which adds to their inability to support different networking scenarios.

Chapter 3

Proposed Content Distribution System

The purpose of this chapter is to explain and formulate the problem of popular content distribution. This is followed by giving an overview of the proposed content distribution system. This overview introduces the overall system structure followed by explaining briefly the main operations involved and the relationships between them. Further details about each operation are left for Chapters 4 to 6. After this overview, details about the case study used throughout experimentation are given in addition to the major assumptions made.

3.1 Problem Formulation

Assuming that the popular content of n_s services is distributed periodically to n_c consumers using a public transportation system where the total number of periods is n_p. These n_c consumers are distributed throughout the day across the vehicle and stop nodes constituting the public transportation system. They enter and leave the system randomly while changing their locations frequently. Their service interests are expressed using the matrix $\mathbf{INT}_{avail}$ of available interests with the size $(n_c \times n_s)$ where $\mathbf{INT}_{avail}[i,j]$ is the interest of consumer i in service j such that:

$$\mathbf{INT}_{avail}[i,j] = \begin{cases} 1 & if\ consumer\ i\ is \\ & interested\ in\ service\ j \\ 0 & if\ consumer\ i\ is\ not \\ & interested\ in\ service\ j \\ NaN & if\ the\ interest\ of \\ & consumer\ i\ in\ service\ j \\ & is\ curently\ unknown \end{cases}$$

Each vehicle node in the public transportation system makes a block of successive trips using certain routes throughout the day before leaving the system. Several stops

are made throughout these trips while waiting for a certain duration of time before departing each stop node.

The first objective to achieve here is to build a content distribution system that serves all n_c consumers by utilizing a total budget of n_{os} optimal stop nodes only and all system vehicles. These stops should be chosen optimally such that we end up with a content distribution system that has a capital cost corresponding to the budget of n_{os} stops. The IDs of these optimal stops, as defined by the vector $\mathbf{id}_{os}$, should also be determined.

The second objective to achieve is to cluster all system nodes given their maximum broadcasting range of r_{br} and the member IDs of all resulting clusters, as defined by the matrices $\mathbf{ID}_{nc}\{1 : n_p\}$ at the different n_p periods, should also be determined. This clustering is essential to allow for gradual system deployment.

At each period p, popular content is distributed in the following three phases:

- Initial V2I content distribution to provide the first version of the popular content segments,
- Direct V2V segment exchanges to replicate the first version of the popular content segments across all system nodes, and
- Final V2I content distribution to provide any missing segments to the system nodes.

Therefore, the third objective to achieve here is to estimate the maximum number of segments n_{as}^{max} that can be distributed to the system nodes at the beginning of each content distribution period p using V2I communication.

The fourth objective to achieve is to find the matrix $\mathbf{A}$ which includes a table of exchange actions which should be taken by the system nodes during the direct V2V segment exchanges where the columns of this matrix $\mathbf{A}$ are defined as follows:

- $\mathbf{A}[*, 1]$ which represents the time indices at which the exchange actions are executed such that no collision occurs with neighboring transmitting nodes,
- $\mathbf{A}[*, 2]$ which represents the indices of the transmitting nodes taking the exchange actions,
- $\mathbf{A}[*, 3]$ which represents the broadcasting ranges of the transmitting nodes such that no interference occurs with neighboring transmitting nodes,
- $\mathbf{A}[*, 4]$ which represents the indices of the chosen segments for transmission, and
- $\mathbf{A}[*, 4+1 : 4+n_n^b]$ which indicates the neighboring nodes which are receiving the chosen data segments from the transmitting nodes given that n_n^b is the number of nodes comprising the biggest nodes cluster under consideration. This indication is made by setting the value of $\mathbf{A}[i, 4 + j] = 1$ if the neighboring node j is the one receiving the chosen data segment and $\mathbf{A}[i, 4 + j] = 0$, otherwise.

The objective here is to maximize the offloading of outdoor wireless networks by maximizing the number of segment exchange actions as defined by $length(\mathbf{A})$ where the function ($length$) gives the length of matrix $\mathbf{A}$. These actions should allow the V2V segment exchanges to be collision-free as well as interference-free.

At each period p, the services are drawn from the predetermined set of n_s services. These services should be prioritized based on recommendations driven from the available interests matrix $\mathbf{INT}_{avail}$. Therefore, the fifth objective to achieve here is to make these recommendations such that they allow the system to have a better estimate of the most popular services which should be distributed to the consumers by achieving the following:

- exploring the consumer interests in the minimum number of interactions, and
- exploiting this knowledge about consumer interests to distribute the maximum number of truly-popular services.

3.2 System Overview

A novel content distribution system is proposed with the two main classes of operations, namely: content recommendation and routing. These operations are proceeded with a set of mobility analysis related operations. Figure 3.1 shows an overview of the proposed system.

As it can be seen, there are four databases in the proposed content distribution system which are: the consumer interests database, the services content database, the mobility features database and the offline results database. In its core, the proposed system has the content distribution procedure which interacts with the databases periodically throughout the day where the number of periods is given by n_p. This procedure receives data from both the services content database and the mobility features database. It does also go through the offline operations in order to update the offline results database which is subsequently used throughout the online operations. As for the consumer interests database, the proposed procedure receives data as well as updates this database while operating.

The consumer interests database stores the matrix $\mathbf{INT}_{avail}$. This matrix is called the available interests matrix because as the time goes and as more content is distributed, the feedback of consumers will be received either explicitly or implicitly and the number of consumer interests which become available will change. Notice also that multiple different versions of this matrix are stored for the different n_p periods to take into account the fact that consumer behavior, in terms of what services are liked or not, might change over the day periods.

The services content represents the content of the different n_s services. The choice of contents and the way they are discarded or sorted at the consumer equipments are all beyond the scope of this work here and therefore are left for future research.

The mobility features database stores the different aspects of vehicle node mobilities and others related to stop nodes. These features are assumed to be

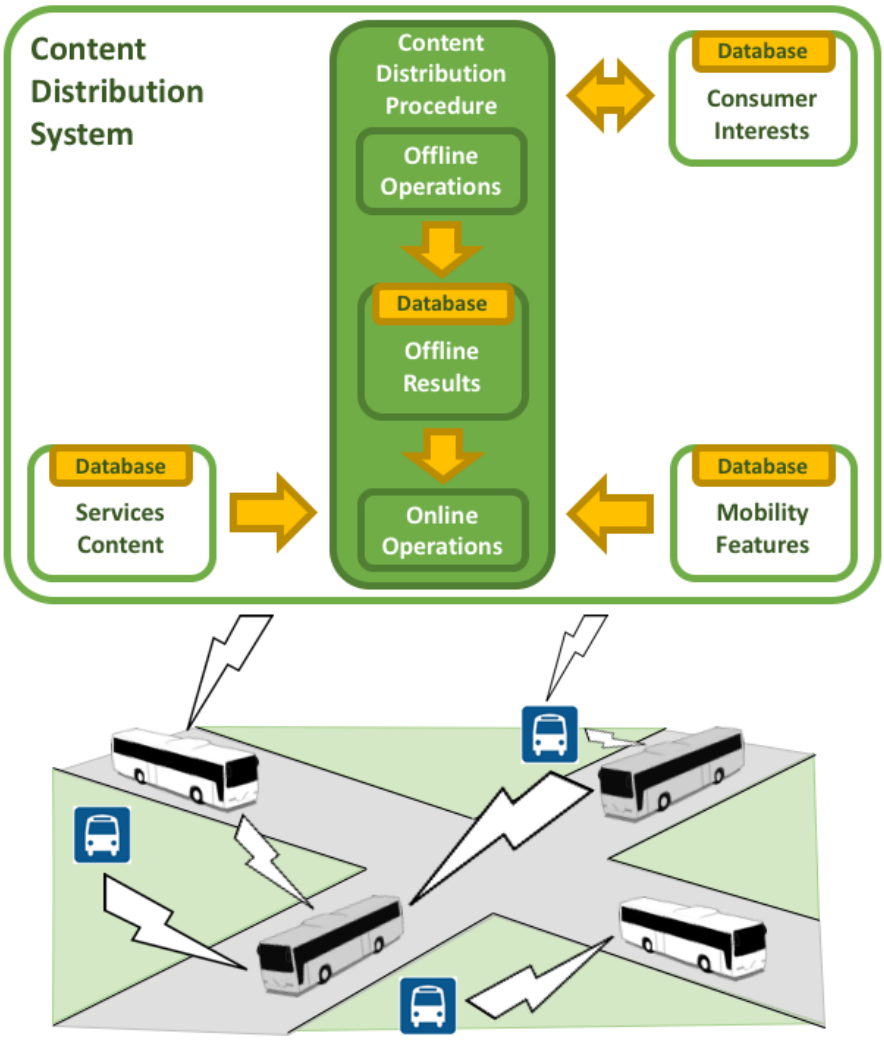


Figure 3.1: Proposed content distribution system

expressed according to the popular General Transit Feed Specification (GTFS) using the following matrices:

- the "Trips" features matrix $\mathbf{F}^t$ which represents aspects related to trips such as the IDs of the routes being followed throughout the trips, the IDs of the transportation services offered, the trip IDs, the directions taken by the vehicles throughout the trips, the IDs of the trip blocks made by the vehicles and the IDs of the map shapes followed by the trip routes where each shape is expressed using several points,

- the "Stop Times" features matrix $\mathbf{F}^{st}$ which represents aspects related to the times at which stops are made throughout the trips. These aspects include the trip IDs, the times at which the vehicle nodes arrive at the stops, the times at which the vehicle nodes depart the stops, the stop IDs and the sequence numbers given to the successive stops made throughout the trips,

- the "Stops" features matrix $\mathbf{F}^s$ which represents aspects related to the stop nodes such as the stop IDs, the stop codes, the stop names, the latitudes of the stop locations and the longitudes of the stop locations, and

- the "Shapes" features matrix $\mathbf{F}^{sh}$ which represents aspects related to the map shapes being followed by the trip routes. These aspects include the shape IDs, the latitudes given to the shape points, the longitudes given to the shape points and the shape point sequence numbers.

The core of the proposed content distribution system is its procedure which is divided between the offline operations shown in Algorithm 3.1 and the online operation shown in Algorithm 3.2.

The offline operations part includes all operations which can be made offline as long as the system has not started running yet. These operations are valid as long as the nodes commit to their mobility features data. The first offline operation made is collecting the mobility feature matrices $\mathbf{F}^t$, $\mathbf{F}^{st}$ and $\mathbf{F}^s$ into one matrix called the "Collected" features matrix $\mathbf{F}^c$. This matrix can be further adjusted to include only those trips offering a certain transportation service type (e.g. workday services only). This matrix is then sorted to make sure that trips of the same block are sorted properly according to their arrival and departure times. Following this, the matrix $\mathbf{F}^c$ is cleaned by first computing vehicle velocities at different trip stop times. Some of these velocities will have NaN values due to the fact that stop arrival and departure times are often measured in minutes only which leads, from the perspective of mobility traces, to having vehicle nodes depart certain stops and arrive at others at the same minute. While this might be acceptable from the perspective of stop schedules, it is considered a data error that should be cleaned from the perspective of mobility traces. Therefore, the matrix $\mathbf{F}^c$ is cleaned from these NaN velocity values by first correcting stop times, recomputing velocities afterwards and then removing any resulting velocities which are higher than a maximum velocity threshold ve_{max}.

To prepare the matrix $\mathbf{F}^c$ for further processing, it needs to be modified by merging the arrival and departure time features into the same time feature. The resulting

"Modified" features matrix $\mathbf{F}^m$ is then synthesized to generate trip trajectories with high time granularity. During this synthesis, trip trajectories are matched to their corresponding maps given the map shapes matrix $\mathbf{F}^{sh}$ while being merged into their corresponding block trajectories with no gaps in between.

Given the synthesized $\mathbf{F}^m$ matrix, the offline operations shown in Algorithm 3.1 proceed with optimizing the stop node selections. These stops are first refined by clustering them given the broadcasting range r_{br} and then choosing only the medoids of the resulting clusters. Given the budget of n_{os} optimal stop nodes, these medoids are further optimized by choosing only those stop node medoids which are the most popular or alternatively have the highest number of times in which vehicle nodes pass by them throughout the day given the maximum broadcasting range r_{br}.

Before being able to cluster the nodes, the connectivities matrix $\mathbf{C}$ of size $(n_T \times n_n \times n_n)$ is computed where n_T is the total number of time steps in which the proposed system is running and n_n is the total number of system nodes including both vehicle and stop nodes. These connectivities are defined as $\mathbf{C}[t, i, j] = 1$ whenever node i and node j are within the broadcasting range r_{br} of each other at time step t and $\mathbf{C}[t, i, j] = 0$, otherwise. Notice that $\mathbf{C}[t, i, j] = \mathbf{C}[t, j, i]$ due to the symmetry in distance measurements.

Given the connectivities matrix $\mathbf{C}$, system nodes are clustered within the same period p into n_g groups meeting cumulative contact duration thresholds as specified by the vector $\mathbf{cd}_{min}$. These thresholds represent the minimum total cumulative contact durations of the different groups such that $\mathbf{cd}_{min}[g]$ represents the minimum cumulative contact duration at which any group g node is in contact with at least another node within the same group g throughout period p. Notice that group g is chosen to be the biggest nodes cluster that satisfies $\mathbf{cd}_{min}[g]$ in order to maximize the offloading potential of the proposed content distribution system by reaching as many nodes and therefore as many consumers as possible.

Another important aspect specified throughout clustering is the maximum number of loop-free hops $n_{h_{max}}$ in which a content segment traverses between any two nodes within the same cluster.

Given $\mathbf{cd}_{min}$ and $n_{h_{max}}$, clustering is made while storing the resulting member IDs of the different node clusters at the different periods in the matrices $\mathbf{ID}_{nc}\{1 : n_p\}$. Notice that the thresholds given by the vector $\mathbf{cd}_{min}$ are chosen in a way that guarantees that all system nodes are eventually chosen to be part of a node cluster. In addition, $n_{h_{max}}$ controls the maximum delay that content segments experience while being distributed within the clusters.

In summary, the outputs of the aforementioned offline operations concerning mobility analysis (to be used as inputs for the upcoming operations) are:

- the matrix of synthesized latitude trajectories of all nodes **LATS**,
- the matrix of synthesized longitude trajectories of all nodes **LONS**,
- the vector of optimal stop node IDs $\mathbf{id}_{os}$, and
- the matrices of cluster member IDs at the different periods $\mathbf{ID}_{nc}\{1 : n_p\}$.

Algorithm 3.1 continues with the offline operations in relation to content routing. Given $\mathbf{ID}_{nc}$, these operations start by estimating the maximum number of segments n_{as}^{max} that can be distributed to the nodes of each cluster at the beginning of the content distribution period. This distribution is made using V2I communication throughout online operations. The number n_{as}^{max} is determined given a predetermined ratio er_{min} between the number of content segments exchanged via V2V communication and the number of segments distributed initially using V2I communication. The V2V communication takes part after the initial V2I direct content distribution at the beginning of the content distribution period throughout online operations.

Given n_{as}^{max}, Algorithm 3.1 proceeds with dividing and allocating the n_{as}^{max} segments between the nodes of each cluster. The fraction of segments given to each node is proportional to its cumulative connectivities summation given the connectivities matrix $\mathbf{C}$. The intuition here is to give more segments to nodes with a higher chance of meeting other nodes as indicated by their connectivities summation. The allocation can however take any form since all segments are treated equally.

With segments allocated to the nodes in each cluster, Algorithm 3.1 finds the weights vector ω of the optimal V2V segment exchanges policy using bayesian optimization. This vector ω comprises the weights given to the node features measuring the V2V communication potential of system nodes. These features are chosen based on my domain knowledge of the field of vehicular networking and their weighted summation gives nodes utilities which are used to prioritize the V2V segment exchanges of the different nodes.

The bayesian optimization can be made using different regression techniques such as: GP, RF, BNN and batch-based RF. The final outcome of this optimization process is the matrix $\mathbf{A}$ which includes a table of exchange actions to be taken by the cluster nodes according to the optimal policy found. These actions allow for the optimal V2V segment exchanges to take place after being stored at the different cluster nodes as indicated by Algorithm 3.1.

Notice that these actions allow the V2V segment exchanges to be collision-free as well as interference-free. This is done by dividing the time between the nodes and controlling their broadcasting ranges as specified by the resulting matrix $\mathbf{A}$ using any power control scheme. Moreover, transmitted content segments are chosen based on their scarcity among the cluster nodes neighboring the transmitting node. This shows clearly that my routing mechanism is adapting between the different routing actions. It uses multi-tier routing using V2I communication at the initial and final content distributions at each period p. This is done while utilizing expedited and splitted routings by dividing the popular content between the system nodes. It does also use delay-tolerant and direct routings throughout the V2V segment exchanges. This is done while utilizing cluster-based routing by clustering system nodes, cross-layer optimized routing by making the exchanges interference-free as well as collision-free and terminated routing by terminating the exchange of segments by the time they are not missing anymore at the system nodes.

Algorithm 3.1 Content distribution procedure (Offline operations)

1. Collect mobility features data;
2. Sort mobility features data;
3. Clean mobility features data;
4. Synthesize mobility features data;
5. Optimize stop node selections;
6. Compute node connectivities;
7. Cluster nodes;
8. Estimate the maximum number of data segments;
9. Divid and allocate data segments to the nodes;
10. Optimize segment V2V exchange tables;
11. Store the optimal segment V2V exchange tables at the nodes;

In summary, the offline operations concerning content routing design achieve the following:

- estimating the maximum number of data segments n_{as}^{max} to distribute initially between the nodes using V2I communication, and
- finding the optimal policy (defined by the vector ω) that maximizes the number of segments exchanged using V2V communication between the nodes.

Further details about the offline operations related to node mobility analysis as shown in Lines 1 to 7 of Algorithm 3.1 are given in Chapter 4 whereas details related to content routing design as shown in Lines 8 to 10 are given in Chapter 6.

After finishing all the offline operations specified in Algorithm 3.1, all results are stored in the offline results database shown in Figure 3.1. These results are used throughout the online operations.

Algorithm 3.2 shows online operations made while the proposed system is running throughout periods 1 to n_p for node clusters 1 to n_g. Figure 3.2, on the other hand, shows the sequence diagram of these operations for cluster g at period p.

For period p and group g, Algorithm 3.2 starts by inquiring the offline results database about the IDs of group g nodes. These IDs have already been determined using the offline nodes clustering operation. Knowing these IDs, the IDs of consumers located at any of the corresponding nodes during most of period p are detected. This detection can be made by directly asking the consumers for their IDs whenever they join the content distribution system.

Given the consumer IDs, interests can be acquired from the consumer interests database. These interests are defined, as mentioned previously, by the matrix $\mathbf{INT}_{avail}$ at any period p. Notice that some assumptions have to be made at $p = 1$ about these interests in order to proceed with the online operations. The nature of these assumptions is beyond the scope of the thesis work presented here and is left for future research.

With the most recent $\mathbf{INT}_{avail}$, services can be prioritized for group g consumers at period p. These services are drawn from a predetermined set of services with a total number of n_s services. The prioritization is made based on recommendations driven from the available interests matrix $\mathbf{INT}_{avail}$. The recommendations allow the system to have a better estimate of the most popular services which should be routed to the consumers of group g. A UPB collaborative and group-based recommender is designed for this purpose and experimented under different consumer interest and network scenarios as will be shown in Chapter 5.

Algorithm 3.2 proceeds by inquiring the offline results database about the maximum number of data segments n_{as}^{max} to be distributed between group g nodes at period p using V2I communication. Given the services content data, these segments are then matched to the prioritized service content segments while making a fixed data rate assumption. This assumption can be made based on the average V2V data rate experienced by cluster g nodes throughout period p. Notice that the number of service content segments as well as their data sizes and priorities control the number of services which the proposed content distribution system would be able to support. Detailed investigation about this matching are beyond the scope of the thesis work presented here and are left for future research.

After matching the services content to the segments, the offline results database is inquired about the segment allocations of the different nodes. Given these allocations, matched segments are routed to their nodes using V2I communication.

With content segments divided and allocated to the system nodes and given the optimal V2V exchange tables stored at these nodes, segment exchanges start according to the optimal V2V segments exchange policy found previously throughout the offline operations.

After the V2V segment exchanges finish, the remaining segments are sent directly using V2I communication before the end of the content distribution period p. The final outcome is having all the system nodes with the number n_{as}^{max} of data segments available at each node. Consumers will give their feedback about the content segments being recommended and routed using the proposed system throughout period p. This feedback can be either explicit or implicit. Either way, the matrix $\mathbf{INT}_{avail}$ of consumer interests within group g at period p is updated according to this feedback in order to have better recommendations in the next content distribution period. Therefore, these recommendations:

- explore the consumer interests in the minimum number of interactions, and
- exploit this knowledge about consumer interests to distribute the maximum number of truly-popular services.

Algorithm 3.2 Content distribution procedure (Online operations)

1. **for** $p = 1 : n_p$ **do**
2. **for** $g = 1 : n_g$ **do**
3. Get cluster g node IDs at period p;
4. Get consumer IDs at cluster g nodes during most of period p;
5. Get the interests of cluster g consumers at period p;
6. Recommend services to cluster g consumers at period p;
7. Get the maximum number of data segments ...
8. for cluster g nodes at period p;
9. Get services content data at period p;
10. Match recommended services content ...
11. to cluster g segments at period p;
12. Get cluster g node segment allocations at period p;
13. Distribute the matched and allocated data segments ...
14. to cluster g nodes at period p using V2I communication;
15. Wait for the V2V segment exchange tables stored ...
16. at cluster g nodes to be executed during period p;
17. Distribute missing segments to cluster g nodes ...
18. at period p using V2I communication;
19. Update the interests of cluster g consumers at period p ...
20. according to their services feedback;
21. **end for**
22. **end for**

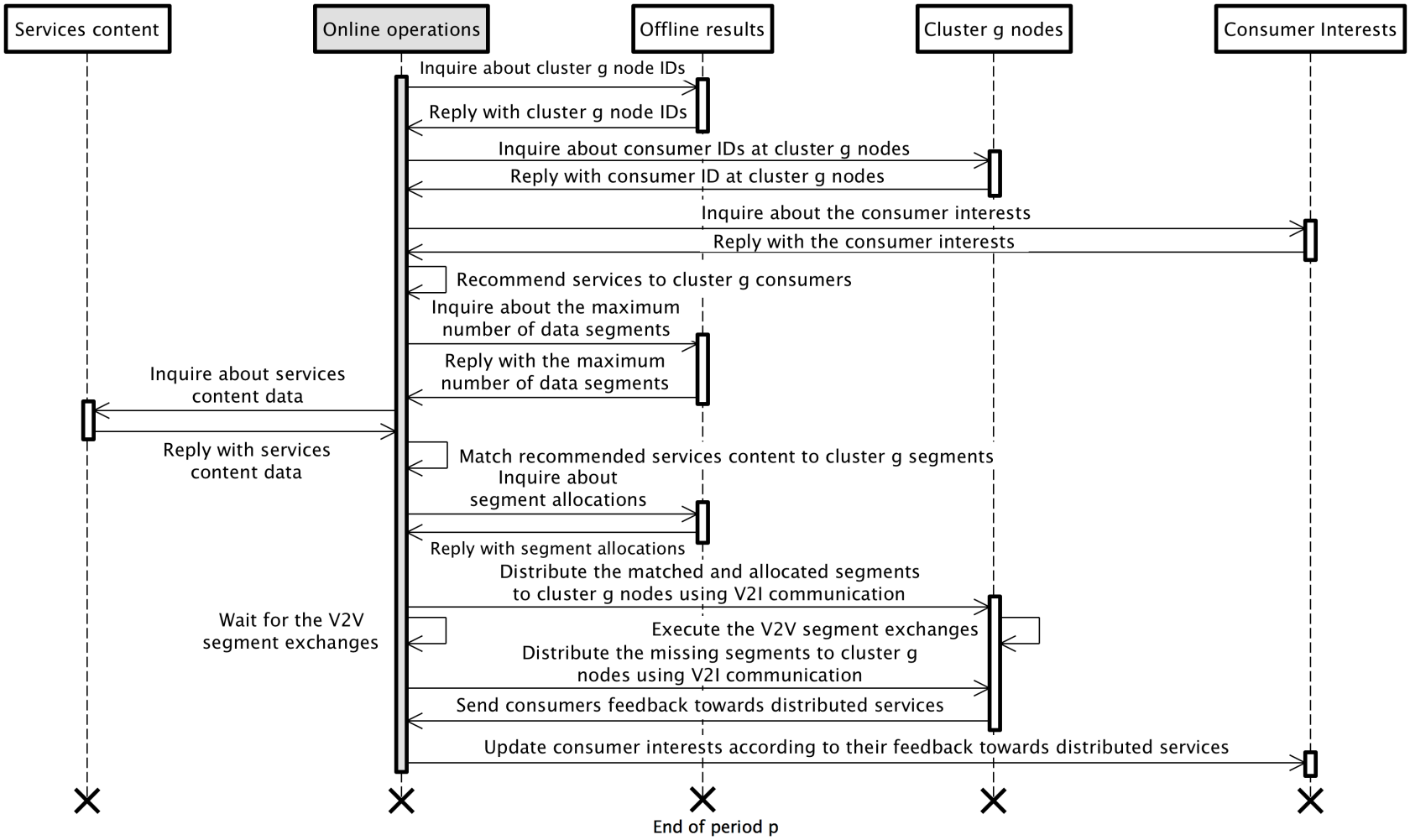


Figure 3.2: Content distribution procedure (Online operations)

3.3 Case Study Assumptions

The mobility data of the public transportation service offered throughout the Region of Waterloo, Ontario, Canada is used as the case study. This region has an estimated area of $1,046\ km^2$ [48] and a transportation service called the Grand River Transit service. This service uses a fleet of buses as the mode of transportation. The period in which this case study service takes place is between February 10^{th}, 2017 and April 25^{th}, 2017. Throughout the design of the proposed system, details about the different operations are given followed by experiments on this case study.

At the beginning, the "Collected" features matrix $\mathbf{F}^c$ is generated using the following case study matrices:

- $\mathbf{F}^t$ representing vehicle trips with a total file size of ~ 500 KB and $6,969$ trips,
- $\mathbf{F}^{st}$ representing vehicle stop times with a total file size of ~ 9.5 MB and $252,622$ stop times, and
- $\mathbf{F}^s$ representing vehicle stops with a total file size of ~ 160 KB and $2,522$ stops.

The datasets of all of these matrices are available at the "Region of Waterloo" website at [49]. Notice, however, that the main focus in the adopted case study is on workdays and therefore these matrices are filtered for these days only.

In the offline operations, the maximum velocity ve_{max} is assumed to be 115 km/h when detecting data errors. During data synthesis, the map shapes matrix $\mathbf{F}^{sh}$, downloaded also from [49], is used with a total file size of ~ 2.5 MB and $75,322$ shape points. Trip trajectories are generated every 10 seconds and therefore the total number of time steps throughout the day $n_{T_{day}}$ is given by $24\,hours \times 60\,mins \times 60\,secs/10$. However and according to the case study, the transportation service runs after midnight until 3:00 AM of the next day which means that the total number of time steps n_T is given by $27\,hours \times 60\,mins \times 60\,secs/10$.

When deciding on the number of optimal stop nodes n_{os}, the total system nodes budget n_n is assumed to be 500 nodes including both vehicles and stops. According to the data, the total number of vehicles is 253 buses and therefore $n_{os} = 500 - 253 = 247$ stop nodes. This is less than 10% of the total number of stop nodes provided by the Grand River Transit service which are needed to be part of the proposed content distribution system.

The broadcasting range r_{br} used throughout all of the offline operations is assumed to be 300 meters. Notice that this assumption is quite conservative given that commonly used V2V communication technologies (e.g. DSRC) support broadcasting ranges as far as 1000 meters nowadays.

When clustering the nodes, the focus of the adopted case study has been on the period between 4:00 PM and 6:00 PM which represents the peak in terms of the total number of active trips. The clustering has been made under the assumption that $cd_{min} = 20$ minutes for the first biggest cluster whereas $n_{h_{max}}$ has been set to 20 hops.

To decide on the maximum number of distributable data segments n_{as}^{max}, the total number of data segments has been varied according to the vector $\mathbf{n}_{as} = [50\ 100\ ...\ 2000]$ which comprises the different numbers of data segments experimented. Meanwhile, the ratio er_{min} has been set such that at least 90% of the content data is being either distributed initially using V2I communication or distributed afterwards using V2V communication by the end of the content distribution period. Notice that the remaining 10% is left for the final V2I communication used to send any remaining missing content segments.

When optimizing the weights vector ω parametrizing the V2V segment exchange policy, the weight given to each feature $\omega[i]$ is assumed to belong to the set $\{-10, -9, ..., 0, ..., 9, 10\} \forall i$. This restriction is made in an attempt to make the implemented bayesian optimization technique faster.

Throughout bayesian optimization, some initial batch data is generated in which the total number of points has been set to 100 points. On the other hand, the total number of bayesian optimization iterations n_i has been set to 900 iterations and the total number of random samples n_{rnd} generated in each one of these iterations has been set to 1000 samples.

In the online operations, each period p is assumed to last for 2 hours. Therefore, a total of 11 periods are used given the following starting and ending time indices:

$$\mathbf{t}^{start} = [t_1^{start}\ \ t_2^{start}\ \ ...\ \ t_{n_p}^{start}]^{1 \times n_p} = [5\ \ 7\ \ ...\ \ 25]\, hrs \times 60\, mins \times 60\, secs/10$$

$$\mathbf{t}^{end} = [t_1^{end}\ \ t_2^{end}\ \ ...\ \ t_{n_p}^{end}]^{1 \times n_p} = [7\ \ 9\ \ ...\ \ 27]\, hrs \times 60\, mins \times 60\, secs/10$$

where:

- $\mathbf{t}^{start}$ is the vector of starting time indices for these periods, and
- $\mathbf{t}^{end}$ is the vector of ending time indices for these periods.

Chapter 4

Mobility Analysis

4.1 Overview

In this chapter, the offline operations which are related to analyzing the mobility data of public transportation nodes are discussed. These operations have already been indicated in Lines 1 to 7 of Algorithm 3.1 discussed previously in Chapter 3. Some of them are for preprocessing the mobility data and others for processing it. The preprocessing operations include: data collection, data sorting, data cleaning and data synthesis. The processing operations include: stop nodes selection optimization, connectivities computation and nodes clustering. An evaluation of the networking potential of nodes is made before going into the details of the clustering technique implemented. This is done by instant-clustering the more challenging and dynamic nodes (i.e. vehicle nodes) under different minimum degrees of connectivity and again under different broadcasting ranges. After that, continuous contact durations are computed for all the system nodes under consideration. Throughout the chapter, the purpose and mathematical formulations of all operations are explained in details while leaving their algorithms to Appendix A at the end of the thesis. Moreover, the case study of the Grand River Transit bus service offered throughout the Region of Waterloo, Ontario, Canada is used with the assumptions explained previously in Chapter 3.

4.2 Data Preprocessing

4.2.1 Data Collection

At the beginning, the feature matrices $\mathbf{F}^t$,$\mathbf{F}^{st}$ and $\mathbf{F}^s$ are defined according to the popular GTFS specification as follows where the function $(length)$ outputs the number of rows of a given matrix:

- $\mathbf{F}^t$ is the "Trips" features matrix set to $[\mathbf{f}_1^t \ \mathbf{f}_2^t \ ... \ \mathbf{f}_7^t]^{length(\mathbf{F}^t)\times 7}$ such as:
  - $\mathbf{f}_1^t$ is the feature vector representing the IDs of the different routes followed by the vehicles throughout their trips,

  - $\mathbf{f}_2^t$ is the feature vector representing the IDs of transportation services where 0 indicates workdays, 1 indicates Saturdays and 2 indicates Sundays,
  - $\mathbf{f}_3^t$ is the feature vector representing the IDs given to vehicle trips,
  - $\mathbf{f}_4^t$ is the feature vector representing the headsigns given to vehicle trips,
  - $\mathbf{f}_5^t$ is the feature vector representing the IDs given to trip directions where 0 indicates one trip direction and 1 indicates the opposing direction,
  - $\mathbf{f}_6^t$ is the feature vector representing the IDs given to each block of successive trips conducted by the same vehicle. Therefore, each block ID corresponds to one or more trip IDs, and
  - $\mathbf{f}_7^t$ is the feature vector representing the IDs given to the map shapes followed by the trip routes. Notice that shapes are expressed using a set of points.

- $\mathbf{F}^{st}$ is the "Stop Times" features matrix set to $[\mathbf{f}_1^{st}\ \mathbf{f}_2^{st}\ ...\ \mathbf{f}_5^{st}]^{length(\mathbf{F}^{st})\times 5}$ such as:
  - $\mathbf{f}_1^{st}$ is the feature vector representing the IDs given to vehicle trips,
  - $\mathbf{f}_2^{st}$ is the feature vector representing the arrival times of vehicles at the different stops throughout their trips,
  - $\mathbf{f}_3^{st}$ is the feature vector representing the departure times of vehicles from the different stops throughout their trips,
  - $\mathbf{f}_4^{st}$ is the feature vector representing the IDs given to vehicle stops, and
  - $\mathbf{f}_5^{st}$ is the feature vector representing the sequence numbers given to stops indicating the order in which they are visited by the vehicles throughout their trips.

- and $\mathbf{F}^s$ is the "Stops" features matrix set to $[\mathbf{f}_1^s\ \mathbf{f}_2^s\ ...\ \mathbf{f}_6^s]^{length(\mathbf{F}^s)\times 6}$ such as:
  - $\mathbf{f}_1^s$ is the feature vector representing the IDs given to stops,
  - $\mathbf{f}_2^s$ is the feature vector representing the codes given to stops which are typically the same as the stop IDs,
  - $\mathbf{f}_3^s$ is the feature vector representing the names given to stops,
  - $\mathbf{f}_4^s$ is the feature vector representing stop descriptions,
  - $\mathbf{f}_5^s$ is the feature vector representing the latitudes of stop locations, and
  - $\mathbf{f}_6^s$ is the feature vector representing the longitudes of stop locations.

From these matrices, the "Collected" features matrix $\mathbf{F}^c$ is generated where $\mathbf{F}^c = [\mathbf{f}_1^c\ \mathbf{f}_2^c\ ...\ \mathbf{f}_{12}^c]^{length(\mathbf{F}^c)\times 12}$ such as:

- $\mathbf{f}_1^c$ is the feature vector representing the IDs given to trip blocks,
- $\mathbf{f}_2^c$ is the feature vector representing the IDs given to trip routes,
- $\mathbf{f}_3^c$ is the feature vector representing the IDs given to trip directions,

- $\mathbf{f}_4^c$ is the feature vector representing trip IDs,
- $\mathbf{f}_5^c$ is the feature vector representing the vehicle arrival times at different stops,
- $\mathbf{f}_6^c$ is the feature vector representing the vehicle departure times from different stops,
- $\mathbf{f}_7^c$ is the feature vector representing stop IDs,
- $\mathbf{f}_8^c$ is the feature vector representing the sequence numbers given to vehicle stops throughout trips,
- $\mathbf{f}_9^c$ is the feature vector representing the latitudes given to shape points,
- $\mathbf{f}_{10}^c$ is the feature vector representing the longitudes given to shape points,
- $\mathbf{f}_{11}^c$ is the feature vector representing shape IDs, and
- $\mathbf{f}_{12}^c$ is the feature vector representing service IDs.

In the adopted case study, workdays are only considered since they have more trips scheduled and therefore potentially serve more popular content consumers compared to Saturday and Sunday services. In order to filter these workdays only from the matrix $\mathbf{F}^c$, the following operation is made:

$$\mathbf{F}^c \leftarrow \mathbf{F}^c[\underset{i}{argfind}(\mathbf{f}_{12}^c[i] = 0), *]$$

where the function ($argfind$) finds any row index i that results in $\mathbf{f}_{12}^c[i] = 0$ which indicates a workday service. After this filtering, the feature column $\mathbf{f}_{12}^c$ is eliminated by setting it to ϕ since there is no need for it anymore (i.e. $\mathbf{f}_{12}^c = \phi$).

Notice that the matrices $\mathbf{F^t}$, $\mathbf{F}^{st}$ and $\mathbf{F}^s$ are all downloaded as datasets from the "Region of Waterloo" website found at [49] where their lengths are:

$$length(\mathbf{F}^t) = 6969,\ length(\mathbf{F}^{st}) = 252622 \text{ and } length(\mathbf{F}^s) = 2522$$

Therefore, the length of matrix $\mathbf{F}^c$ has been found to be:

$$length(\mathbf{F}^c) = length(\mathbf{F}^{st}) = 252622$$

Refer to Algorithm A.1 in Appendix A for further details about this data collection operation.

4.2.2 Data Sorting

Sorting the data presented by matrix $\mathbf{F}^c$ is critical for the upcoming analysis operations as well as for quick validations. Therefore, the rows of matrix $\mathbf{F}^c$ are sorted first according to the order of its block IDs followed by the order of arrival times for all the trips within each of these block IDs. This can be done as follows:

$$\mathbf{F}^c \leftarrow sort(\mathbf{F}^c, [1, 5])$$

where the function ($sort$) is used to sort the rows of its first matrix argument according to the order of columns indicated in its second argument where positive column numbers indicate an ascending order whereas negative numbers indicate a descending order.

This initial sorting results though in some rows of different same-block trips with the same arrival time to be sorted in an order that leads to a sudden and unreasonable change in trip directions. Therefore, these rows are resorted depending on whether this change of directions is at the beginning or the end of the trip.

These abrupt changes of directions occur whenever the block direction $\mathbf{f}_3^c[i]$ for any row index $i : i \in \{2, 3, ..., length(\mathbf{F}^c)\}$ does not equal the immediate previous row block direction $\mathbf{f}_3^c[i-1]$ and the immediate next row block direction $\mathbf{f}_3^c[i+1]$. If this happens, then the features matrix $\mathbf{F}^{bl}$ of the same block is extracted first as follows:

$$\mathbf{F}^{bl} = \mathbf{F}^c[\underset{j}{argfind}(\mathbf{f}_1^c[j] = \mathbf{f}_1^c[i]), *]$$

where the function ($argfind$) is used to find the indices of trip rows belonging to the same block ID $\mathbf{f}_1^c[i]$ such as i is the current $\mathbf{F}^c$ row index. After that, the features matrix $\mathbf{F}_{st}^{bl}$ of the same block and the same arrival time is extracted as follows:

$$\mathbf{F}_{st}^{bl} = \mathbf{F}^{bl}[\underset{j}{argfind}(\mathbf{f}_5^{bl}[j] = \mathbf{f}_5^c[i]), *]$$

where the function ($argfind$) is used to find the indices of same-block trip rows which have the same arrival time such as i is the current $\mathbf{F}^c$ row index. Notice that $\mathbf{f}_i^{bl}$ is the feature column vector i of $\mathbf{F}^{bl}$.

The resulting same-block and same-arrival-time features matrix $\mathbf{F}_{st}^{bl}$ is then sorted either in an ascending order or in a descending order with respect to the trip direction. This is depending on whether the abrupt change of directions is taking place at the beginning or at the end of the trip under consideration and depending on the current block direction $\mathbf{f}_3^c[i]$. The function ($sort$) is used for this purpose with its second argument indicating that the trip direction column is used to sort the data and the order of this sorting.

After resorting the data, the unique rows of the resulting $\mathbf{F}^c$ are found using the function ($unique$) as follows:

$$\mathbf{F}^c \leftarrow unique(\mathbf{F}^c)$$

Refer to Algorithms A.2 and A.3 in Appendix A for further details about this data sorting operation.

4.2.3 Data Cleaning

In order to clean the data, vehicle velocities are computed first to be used in detecting errors afterwards. Velocity computations are made whenever we are at the same block ID such that $\mathbf{f}_1^c[i] = \mathbf{f}_1^c[i-1]$ for any row with index $i : i \in \{2, 3, ..., length(\mathbf{F}^c)\}$. The distance d is computed first between the (latitude, longitude) coordinates of the current trip row $[\mathbf{f}_9^c[i]\ \mathbf{f}_{10}^c[i]]$ and the previous trip

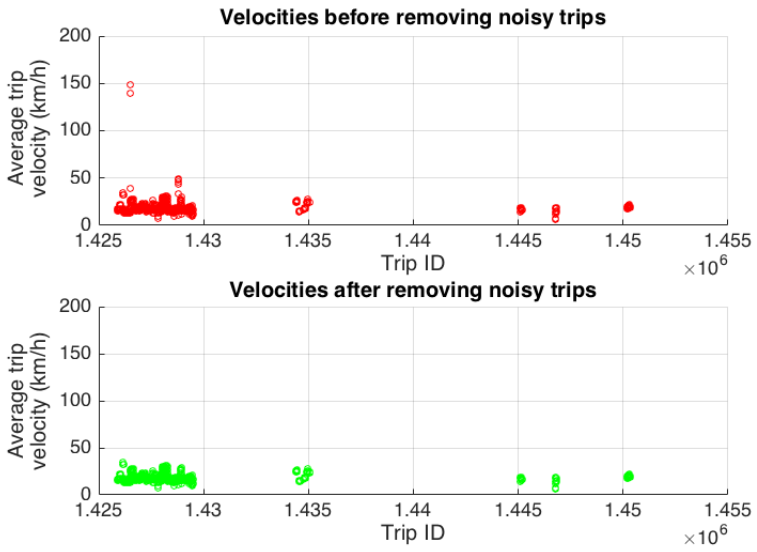


Figure 4.1: Velocities before and after removing noisy trips

row $[\mathbf{f}_9^c[i-1]\ \mathbf{f}_{10}^c[i-1]]$ using the function $(distance)$. Then, the current travel time T_{travel} is computed by subtracting the previous trip row departure time $\mathbf{f}_6^c[i-1]$ from the current trip row arrival time $\mathbf{f}_5^c[i]$. Based on these d and T_{travel} values, the current trip row velocity $\mathbf{f}_{12}^c[i]$ is computed using the formula $\mathbf{f}_{12}^c[i] = d/T_{travel}$.

Whenever $T_{travel} = 0$, this means that $\mathbf{f}_{12}^c[i]$ is NaN which indicates an error. In addition, some velocities might be found to be excessively high, based on this computation, which also indicates an error. Therefore, data cleaning starts by visually recognizing those trips with extreme velocities and removing them by manually setting the trip rows with their trip IDs to $\emptyset$. Notice that in the context of the analysis conducted here, the velocity of 115 km/h is set as the maximum velocity threshold ve_{max}. Figure 4.1 shows the trip velocities before and after removing these noisy and high velocity trips.

After that, $\mathbf{F}^c$ rows with NaN-velocity values are replaced by first identifying the vector of all block IDs $\mathbf{id}_{bl}$ using the formula:

$$\mathbf{id}_{bl} = unique(\mathbf{f}_1^c)$$

where the function $(unique)$ is used to find the unique column elements. Then, each features matrix of the same block $\mathbf{F}^{bl}$ is checked if $\exists k : \mathbf{f}_{12}^{bl}[k]\ is\ NaN$ in order to replace its NaN-velocity values. This is done by looping around trip rows within indices 2 and $(length(\mathbf{F}^{bl}) - 1)$ of the same block ID and replacing those NaN-velocity values whenever $(\mathbf{f}_{12}^{bl}[j]\ is\ NaN)$ and $(\mathbf{f}_{12}^{bl}[j+1]\ is\ not\ NaN)$ where j is the current trip row index of $\mathbf{F}^{bl}$. Each loop iteration is started with the computation of the current waiting time T_{wait} by subtracting the current row arrival time $\mathbf{f}_5^{bl}[j]$ from the current row departure time $\mathbf{f}_6^{bl}[j]$. After that, the current row velocity NaN-value is overcome by resetting the current row arrival time $\mathbf{f}_5^{bl}[j]$ to be in the middle of the period between the previous row departure time $\mathbf{f}_6^{bl}[j-1]$ and the next row arrival time $\mathbf{f}_5^{bl}[j+1]$. This is done using the formula:

$$\mathbf{f}_5^{bl}[j] = \mathbf{f}_6^{bl}[j-1] + (\mathbf{f}_5^{bl}[j+1] - \mathbf{f}_6^{bl}[j-1])/2$$

This leads to the recomputation of the current departure time $\mathbf{f}_6^{bl}[j]$ given T_{wait} as follows:

$$\mathbf{f}_6^{bl}[j] = min(\mathbf{f}_5^{bl}[j] + T_{wait}, \mathbf{f}_5^{bl}[j+1])$$

Such that the current departure time $\mathbf{f}_6^{bl}[j]$ does not exceed the next row arrival time $\mathbf{f}_5^{bl}[j+1]$. Then, and in order to recompute the current row velocity value $\mathbf{f}_{12}^{bl}[j]$ and replace its NaN-value, the distance d is computed first between the current row coordinates $[\mathbf{f}_9^{bl}[j] \;\; \mathbf{f}_{10}^{bl}[j]]$ and the previous row coordinates $[\mathbf{f}_9^{bl}[j-1] \;\; \mathbf{f}_{10}^{bl}[j-1]]$ using the formula:

$$d = distance([\mathbf{f}_9^{bl}[j] \;\; \mathbf{f}_{10}^{bl}[j]], [\mathbf{f}_9^{bl}[j-1] \;\; \mathbf{f}_{10}^{bl}[j-1]])$$

After that, T_{travel} is computed given the new value of $\mathbf{f}_5^{bl}[j]$ and using the formula:

$$T_{travel} = \mathbf{f}_5^{bl}[j] - \mathbf{f}_6^{bl}[j-1]$$

Based on these d and T_{travel} values, $\mathbf{f}_{12}^{bl}[j]$ is computed as follows: $\mathbf{f}_{12}^{bl}[j] = d/T_{travel}$. However, changing $\mathbf{f}_6^{bl}[j]$ means that $\mathbf{f}_{12}^{bl}[j+1]$ should be recomputed by first computing the distance d between the current row coordinates $[\mathbf{f}_9^{bl}[j] \;\; \mathbf{f}_{10}^{bl}[j]]$ and the next row coordinates $[\mathbf{f}_9^{bl}[j+1] \;\; \mathbf{f}_{10}^{bl}[j+1]]$ and also the travel time T_{travel} between the current row departure time and the next row arrival time using the formula:

$$T_{travel} = \mathbf{f}_5^{bl}[j+1] - \mathbf{f}_6^{bl}[j]$$

Based on these d and T_{travel} values, $\mathbf{f}_{12}^{bl}[j+1]$ is computed as follows: $\mathbf{f}_{12}^{bl}[j+1] = d/T_{travel}$ if $T_{travel} \neq 0$. If this $T_{travel} = 0$, then going through the loop, as long as $\exists k : \mathbf{f}_{12}^{bl}[k] \; is \; NaN$, will eventually eliminate all velocity NaN-values except for $\mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl})]$ at the very last row of $\mathbf{F}^{bl}$. This can be overcome by setting $\mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl})]$ to equal $\mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl}) - 1]$ at the trip row before and then continue adjusting $\mathbf{f}_5^{bl}[length(\mathbf{f}_5^{bl})]$ and $\mathbf{f}_6^{bl}[length(\mathbf{f}_6^{bl})]$ accordingly.

Figure 4.2 shows the percentages of velocity NaN-values before and after replacing the velocity errors. Notice the high percentage of velocity NaN-values before replacing the errors which results from having same successive arrival times at different locations. These times are measured only in hours and minutes which leads them to look mistakenly the same.

In order to replace $\mathbf{F}^c$ rows with velocities $> ve_{max}$, the indices vector of these rows $\mathbf{id}_{rows}^{hve}$ is first identified as follows:

$$\mathbf{id}_{rows}^{hve} = \underset{j}{argfind}(\mathbf{f}_{12}^{c}[j] > ve_{max})$$

where the function $(argfind)$ is used to identify any trip row index j that satisfies the condition $\mathbf{f}_{12}^{c}[j] > ve_{max}$. Then and for each index i between 1 and $length(\mathbf{id}_{rows}^{hve})$, $\mathbf{f}_{12}^{c}[j]$ is set to ve_{max} where $j = \mathbf{id}_{rows}^{hve}[i]$. Based on this new value of $\mathbf{f}_c^{12}[j]$, both $\mathbf{f}_5^c[j]$ and $\mathbf{f}_6^c[j]$ are updated accordingly where $\mathbf{f}_6^c[j]$ can not exceed $\mathbf{f}_5^c[j+1]$. After that and given the new value of $\mathbf{f}_6^c[j]$, $\mathbf{f}_{12}^c[j+1]$ is updated accordingly. Figure 4.3 shows the velocity distribution after replacing these rows with high velocities.

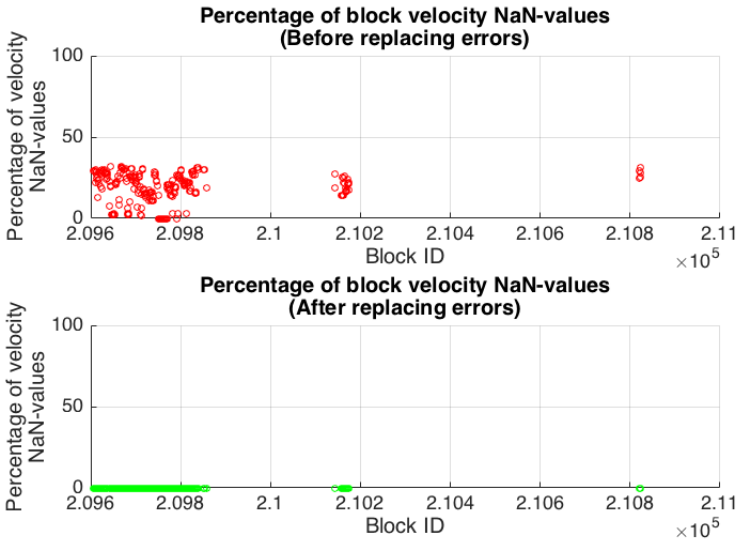


Figure 4.2: Percentage of velocity NaN-values before and after replacing errors

Refer to Algorithms A.4 to A.6 in Appendix A for further details about this data cleaning operation.

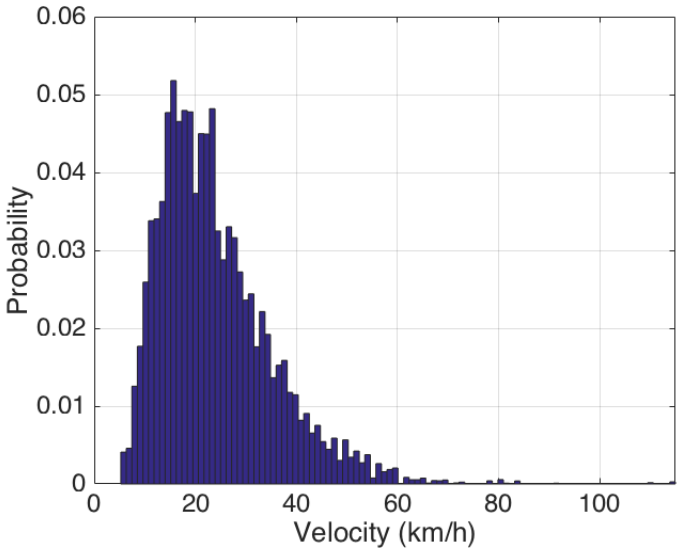


Figure 4.3: Velocity distribution

4.2.4 Data Synthesis

The mobility data is synthesized in order to create trip trajectories with regular and small-enough time intervals. However and before doing so, the matrix $\mathbf{F}^c$ is modified in order to create one feature column for all trip times instead of two separate feature columns for the arrival and departure times. This modification facilitates the

upcoming data synthesis using linear regression between the successive trip times represented by the newly created feature column. Notice that linear regression is used here to assume fixed vehicle velocities between successive trip stops.

Given $\mathbf{F}^c$, the "Modified" features matrix $\mathbf{F}^m$ is set first as follows:

$$\mathbf{F}^m = [\mathbf{f}_1^c \;\; \mathbf{f}_{11}^c \;\; \mathbf{f}_4^c \;\; \mathbf{f}_5^c \;\; \mathbf{f}_6^c \;\; \mathbf{f}_9^c \;\; \mathbf{f}_{10}^c]$$

After that, trip rows with indices 1 to $length(\mathbf{F}^m)$ are added the extra row $\mathbf{row}_{extra}$ that has the same data as $\mathbf{F}^m[i, *]$ except for setting $\mathbf{f}_4^m[i]$ to equal $\mathbf{f}_5^m[i]$ where i is the current row index. This addition is done whenever the current row arrival and departure times are not equal (i.e. $\mathbf{f}_4^m[i] \neq \mathbf{f}_5^m[i]$) and is made by vertically concatenating $\mathbf{row}_{extra}$ to $\mathbf{F}^m$ as follows:

$$\mathbf{F}^m[i+1 : length(\mathbf{F}^m)+1, *] = (\mathbf{row}_{extra}, \mathbf{F}^m[i+1 : length(\mathbf{F}^m), *])$$

The $\mathbf{F}^m$ feature columns $\mathbf{f}_i^m$ for $i : i \in \{1, 2, ..., 7\}$ are therefore defined as follows given that $\mathbf{F}^m = [\mathbf{f}_1^m \;\; \mathbf{f}_2^m \;\; ... \;\; \mathbf{f}_7^m]^{length(\mathbf{F}^m)\times 7}$:

- $\mathbf{f}_1^m$ is the feature vector representing the IDs given to blocks,
- $\mathbf{f}_2^m$ is the feature vector representing the IDs given to map shapes,
- $\mathbf{f}_3^m$ is the feature vector representing the IDs given to trips,
- $\mathbf{f}_4^m$ is the feature vector representing the vehicle arrival times at stops,
- $\mathbf{f}_5^m$ is the feature vector representing the vehicle departure times from stops,
- $\mathbf{f}_6^m$ is the feature vector representing the latitudes of stop locations, and
- $\mathbf{f}_7^m$ is the feature vector representing the longitudes of stop locations.

The departure time feature column $\mathbf{f}_5^m$ is eliminated by setting it to $\emptyset$ since it is already now part of $\mathbf{f}_4^m$. The final $\mathbf{F}^m$ features matrix is therefore defined as follows:

$$\mathbf{F}^m = [\mathbf{f}_1^m \;\; \mathbf{f}_2^m \;\; ... \;\; \mathbf{f}_6^m]^{length(\mathbf{F}^m)\times 6}$$

where all feature column vectors are the same except for: $\mathbf{f}_5^m = \mathbf{f}_6^m$ and $\mathbf{f}_6^m = \mathbf{f}_7^m$. Therefore, we now have:

- $\mathbf{f}_5^m$ as the feature vector representing the latitudes of stop locations, and
- $\mathbf{f}_6^m$ as the feature vector representing the longitudes of stop locations.

Notice that $length(\mathbf{F}^m) = 125949$ at the end of the computations.

After modifying the data and generating $\mathbf{F}^m$, trips are synthesized while making sure that the resulting trip trajectories are matching the city map. This data synthesis produces the trip latitudes trajectory matrix $\mathbf{LATS}_{tr}$ and the trip longitudes trajectory matrix $\mathbf{LONS}_{tr}$ using the "Shapes" features matrix $\mathbf{F}^{sh}$ which is set to $[\mathbf{f}_1^{sh} \;\; \mathbf{f}_2^{sh} \;\; ... \;\; \mathbf{f}_4^{sh}]^{length(\mathbf{F}^{sh})\times 4}$ such as:

- $\mathbf{f}_1^{sh}$ is the feature vector representing the IDs given to map shapes,
- $\mathbf{f}_2^{sh}$ is the feature vector representing the latitudes of shape points,
- $\mathbf{f}_3^{sh}$ is the feature vector representing the longitudes of shape points, and
- $\mathbf{f}_4^{sh}$ is the feature vector representing the sequence numbers of the shape points.

Notice that $\mathbf{LATS}_{tr}[i,t]$ is the synthetic latitude of trip i at time step t and $\mathbf{LONS}_{tr}[i,t]$ is the synthetic longitude of trip i at time step t.

This computation starts by first identifying the vector of all trip IDs $\mathbf{id}_{tr}$ using the formula:

$$\mathbf{id}_{tr} = unique(\mathbf{f}_3^m)$$

Then this computation goes through all trip IDs starting with the trip of index 1 to the trip of index $length(\mathbf{id}_{tr})$. The trajectory matrix $\mathbf{TR}_{tr}$ of the current trip i is extracted first with the rows of indices: $\underset{j}{argfind}(\mathbf{f}_3^m[j] = \mathbf{id}_{tr}[i])$. The linear regression model of the trip latitudes $\mathbf{Mdl}_{lat}$ is then fitted using the current trip trajectory arrival times $\mathbf{TR}_{tr}[*,4]$ as the input data and the current trip trajectory latitudes $\mathbf{TR}_{tr}[*,5]$ as the output data. The linear regression model of the trip longitudes $\mathbf{Mdl}_{lon}$ is also fitted using the current trip trajectory arrival times $\mathbf{TR}_{tr}[*,4]$ as the input data and the current trip trajectory longitudes $\mathbf{TR}_{tr}[*,6]$ as the output data.

Using the models $\mathbf{Mdl}_{lat}$ and $\mathbf{Mdl}_{lon}$, the current trip trajectory matrices $\mathbf{LATS}_{tr}$ and $\mathbf{LONS}_{tr}$ are generated. This is done for every t between 0 and n_T such that it satisfies the condition:

$$min(\mathbf{TR}_{tr}[*,4]) \leq t/n_{T_{day}} \leq max(\mathbf{TR}_{tr}[*,4])$$

which indicates that this t is within the active period of the current trip. At each time step t, the resulting coordinate vector $\mathbf{mdl}_{result}$ is generated using the models $\mathbf{Mdl}_{lat}$ and $\mathbf{Mdl}_{lon}$ as follows:

$$\mathbf{mdl}_{result} = [\mathbf{Mdl}_{lat}(t/n_{T_{day}}) \ \ \mathbf{Mdl}_{lon}(t/n_{T_{day}})]$$

However, $\mathbf{mdl}_{result}$ has not yet matched the map shape of the current trip trajectory. In order to do this matching, the shape trajectory rows $\mathbf{TR}_{sh}$ are first extracted using $\mathbf{F}^{sh}$ which have the indices: $\underset{j}{argfind}(\mathbf{f}_1^{sh}[j] = \mathbf{TR}_{tr}[1,2])$ (i.e. have the same shape ID as that of the current trip trajectory). Then, the vector of all distances $\mathbf{d}$ between the current interpolated coordinate given by $\mathbf{mdl}_{result}$ and all the coordinates $[\mathbf{TR}_{sh}[j,2] \ \ \mathbf{TR}_{sh}[j,3]] \forall j : j \in \{1,2,...,length(\mathbf{TR}_{sh})\}$ is computed. This vector $\mathbf{d}$ is used to match the city map by finding the shape coordinate that is closest to the current interpolated trip coordinate $\mathbf{mdl}_{result}$ using the formulas:

$$\mathbf{LATS}_{tr}[i,t] = \mathbf{TR}_{sh}[\underset{j}{argmin}(\mathbf{d}[j]), 2]$$

$$\mathbf{LONS}_{tr}[i,t] = \mathbf{TR}_{sh}[\underset{j}{argmin}(\mathbf{d}[j]), 3]$$

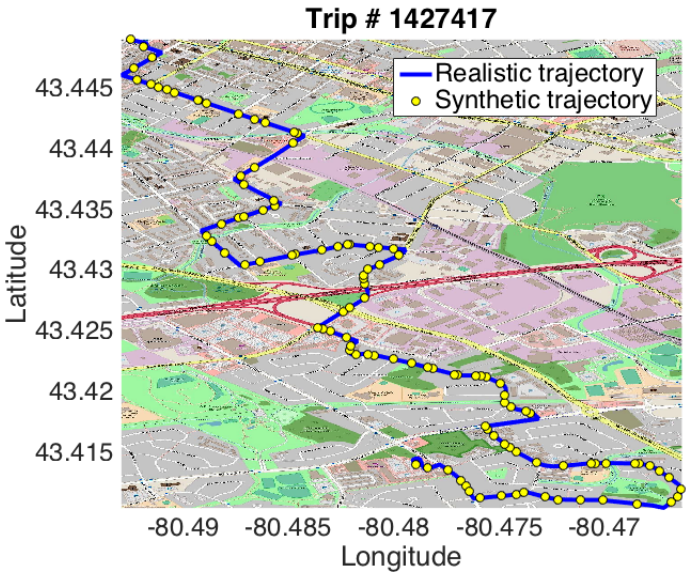


Figure 4.4: Synthetic vs. realistic trip trajectory

where:

- i is the current trip index,
- t is the current time step,
- $\mathbf{TR}_{sh}[j, 2]$ is the shape latitude at point j, and
- $\mathbf{TR}_{sh}[j, 3]$ is the shape longitude at point j.

These new trip trajectory matrices $\mathbf{LATS}_{tr}$ and $\mathbf{LATS}_{tr}$ are now synthesized to have the required granularity while being matched to the city map. Figure 4.4 shows the matching between the synthetic and the realistic trajectories of a trip chosen at random by showing the overlap between them.

In the case study adopted, the matrix $\mathbf{F}^{sh}$ is downloaded as a dataset from the Grand River Transit website found at [49] with $length(\mathbf{F}^{sh}) = 75322$. n_T is set to $27\,hours \times 60\,mins \times 60\,secs/10$ because some trips take place at the $\sim 27^{th}$ hour of the day (i.e. $\sim$ 3:00 AM of the next day). The division by 10 is made given that 10 seconds is the shortest time interval used in the analysis.

Given $\mathbf{LATS}_{tr}$ and $\mathbf{LONS}_{tr}$ of all the trips, the latitude trajectories matrix $\mathbf{LATS}_{bl}$ as well as the longitude trajectories matrix $\mathbf{LONS}_{bl}$ for all blocks are generated where each block is composed of one or more trips. These matrices are synthesized by first identifying the vector of all block IDs $\mathbf{id}_{bl}$ using the formula: $\mathbf{id}_{bl} = unique(\mathbf{f}_1^m)$ and the vector of all trip IDs $\mathbf{id}_{tr}$ using the formula: $\mathbf{id}_{tr} = unique(\mathbf{f}_3^m)$. After that, the computation goes over block IDs starting with the ID of index 1 and ending with the ID of index $length(\mathbf{id}_{bl})$. For each block, the vector of all trip IDs within the current block ID $\mathbf{id}_{tr}^{bl}$ is identified using the formula:

$$\mathbf{id}_{tr}^{bl} = unique(\mathbf{F}^m[\underset{j}{argfind}(\mathbf{f}_1^m[j] = \mathbf{id}_{bl}[i]), 3])$$

where trip rows of the same block ID within the current block i have the set of indices: $\underset{j}{argfind}(\mathbf{f}_1^m[j] = \mathbf{id}_{bl}[i])$. The first trip index of the current block trips $id_{tr}^{bl}|_{first}$ as well as the last one $id_{tr}^{bl}|_{last}$ are found as follows:

$$id_{tr}^{bl}|_{first} = \underset{j}{argfind}(\mathbf{id}_{tr}[j] = \mathbf{id}_{tr}^{bl}[1])$$

$$id_{tr}^{bl}|_{last} = \underset{j}{argfind}(\mathbf{id}_{tr}[j] = \mathbf{id}_{tr}^{bl}[length(\mathbf{id}_{tr}^{bl})])$$

Given that each block is composed of one trip or more, both $\mathbf{LATS}_{bl}$ and $\mathbf{LONS}_{bl}$ can now be computed by summing over their corresponding trip vectors starting with the first trip of index $id_{tr}^{bl}|_{first}$ and ending with the last trip of index $id_{tr}^{bl}|_{last}$ as follows:

$$\mathbf{LATS}_{bl}[i,*] = \sum_{j=id_{tr}^{bl}|_{first}}^{id_{tr}^{bl}|_{last}} \mathbf{LATS}_{tr}[j,*]$$

$$\mathbf{LONS}_{bl}[i,*] = \sum_{j=id_{tr}^{bl}|_{first}}^{id_{tr}^{bl}|_{last}} \mathbf{LONS}_{tr}[j,*]$$

where:

- $\mathbf{LATS}_{bl}[i,*]$ is the latitudes vector of block i for $1 \leq t \leq n_T$, and
- $\mathbf{LONS}_{bl}[i,*]$ is the longitudes vector of block i for $1 \leq t \leq n_T$.

Both $\mathbf{LATS}_{bl}$ and $\mathbf{LONS}_{bl}$ still have gaps between their successive trips with zero values due to stop waiting periods. These inter-trip gaps are overcome for each block ID in $\mathbf{id}_{bl}$. At first, the modified features matrix $\mathbf{F}_{bl}^m$ of the current block with ID $\mathbf{id}_{bl}[i]$ is extracted using the formula:

$$\mathbf{F}_{bl}^m = \mathbf{F}^m[\underset{j}{argfind}(\mathbf{f}_1^m[j] = \mathbf{id}_{bl}[i]), *]$$

After that, the block start time t_{start}^{bl} and the block end time t_{end}^{bl} are identified as follows:

$$t_{start}^{bl} = \mathbf{F}_{bl}^m[1,4]$$

$$t_{end}^{bl} = \mathbf{F}_{bl}^m[length(\mathbf{F}_{bl}^m), 4]$$

Then, this block is scanned for any gaps between its successive trips for the whole n_T time steps. This is done whenever the normalized time index $t/n_{T_{day}}$ lies between t_{start}^{bl} and t_{end}^{bl}. The occurrence of a gap is indicated by having $\mathbf{LATS}_{bl}[i,t] = 0$ where i is the index of the current block and t is the current time index. If a gap exists, then the number of time steps constituting this gap $n_{T_{gap}}$ is counted.

Given $n_{T_{gap}}$, the latitudes trajectory matrix of the gap $\mathbf{TR}_{gap}^{lats}$ as well as the longitudes trajectory matrix of the gap $\mathbf{TR}_{gap}^{lons}$ are formed. This is done using the corresponding trajectory points immediately before and immediately after the inter-trip gap under consideration.

Using $\mathbf{TR}_{gap}^{lats}$ and $\mathbf{TR}_{gap}^{lons}$, linear regression models are fitted to interpolate and find trajectory points within the gap. These models are the gap linear latitudes model $\mathbf{Mdl}_{lat}$ when using $\mathbf{TR}_{gap}^{lats}$ and the gap linear longitudes model $\mathbf{Mdl}_{lon}$ when using $\mathbf{TR}_{gap}^{lons}$. Gap trajectories between t and $(t + n_{T_{gap}})$ are then found using the models $\mathbf{Mdl}_{lat}$ and $\mathbf{Mdl}_{lon}$ as follows:

$$\mathbf{LATS}_{bl}[i, t : t + n_{T_{gap}}] = \mathbf{Mdl}_{lat}((t : t + n_{T_{gap}})/n_{T_{day}})$$

$$\mathbf{LONS}_{bl}[i, t : t + n_{T_{gap}}] = \mathbf{Mdl}_{lon}((t : t + n_{T_{gap}})/n_{T_{day}})$$

After filling the gaps, some overlaps still need to be corrected. These overlaps are of length 1 and they result from the trip trajectory summations made. They are mainly attributed to the fact that many successive trips of the same block end and start at the same time (i.e. have no waiting periods). Whenever this occurs, it has the effect of doubling block trajectory values as a result of adding the corresponding same trip trajectory values.

In order to overcome these overlaps, the block latitude and longitude trajectories are mainly divided by 2 whenever their values exceed the most extreme trip latitude and longitude trajectory values. This is done by first extracting $\mathbf{F}_{bl}^{m}$, t_{start}^{bl} and t_{end}^{bl} like how it has been done before. Then, the current block trajectories $\mathbf{LATS}_{bl}$ and $\mathbf{LONS}_{bl}$ are scanned to detect the values which exceed the most extreme $\mathbf{LATS}_{tr}$ and $\mathbf{LONS}_{tr}$ values (i.e. $max(\mathbf{LATS}_{tr})$ and $min(\mathbf{LONS}_{tr})$) while being surrounded by normal block trajectory values. This scanning takes place within the active block period (i.e. $t_{start}^{bl} \leq t/n_{T_{day}} \leq t_{end}^{bl}$). Notice that whenever an extreme $\mathbf{LATS}_{bl}$ or $\mathbf{LONS}_{bl}$ value is detected, a division by 2 is made as follows:

$$\mathbf{LATS}_{bl}[i, t] \leftarrow \mathbf{LATS}_{bl}[i, t]/2$$

$$\mathbf{LONS}_{bl}[i, t] \leftarrow \mathbf{LONS}_{bl}[i, t]/2$$

where i is the current block index and t is the current time index.
Refer to Algorithms A.7 to A.11 in Appendix A for further details about this data synthesis operation.

4.3 Data Processing

4.3.1 Stop Nodes Selection Optimization

In order to optimize the stop node selections, these selections need to be refined first using clustering. This is done using the function (*cluster*) which needs stop coordinates to be in the cartesian coordinates system. This is done by creating the "Converted Stops" features matrix $\mathbf{F}^{cs}$ set to $[\mathbf{f}_1^{cs}\ \ \mathbf{f}_2^{cs}\ \ ...\ \ \mathbf{f}_6^{cs}]^{length(\mathbf{F}^{cs})\times 6}$ such as:

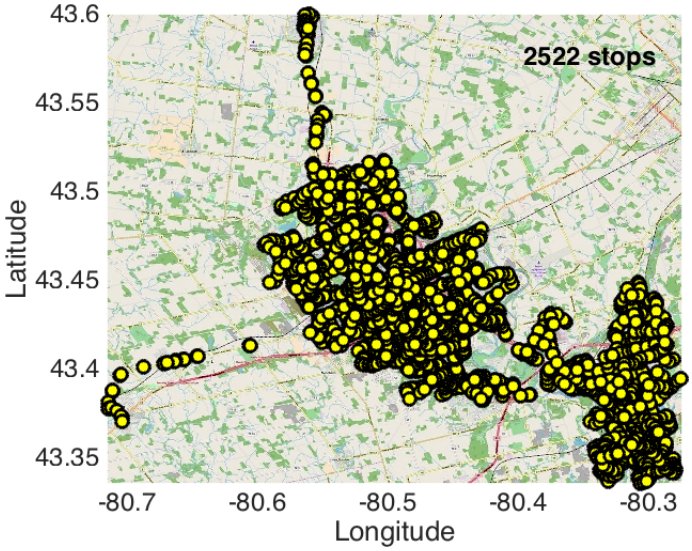


Figure 4.5: All stops

- $\mathbf{f}_1^{cs}$ is the feature vector representing the IDs given to stops,
- $\mathbf{f}_2^{cs}$ is the feature vector representing the latitudes of stop locations where $\mathbf{f}_2^{cs} = \mathbf{f}_5^{s}$,
- $\mathbf{f}_3^{cs}$ is the feature vector representing the longitudes of stop locations where $\mathbf{f}_3^{cs} = \mathbf{f}_6^{s}$,
- $\mathbf{f}_4^{cs}$ is the feature vector representing the converted x-coordinates of stop locations,
- $\mathbf{f}_5^{cs}$ is the feature vector representing the converted y-coordinates of stop locations, and
- $\mathbf{f}_6^{cs}$ is the feature vector representing the converted z-coordinates of stop locations.

These features are created using the following formulas:

$$\mathbf{f}_4^{cs} = r_{earth} \cdot cos(\mathbf{f}_2^{cs}) \cdot cos(\mathbf{f}_3^{cs})$$

$$\mathbf{f}_5^{cs} = r_{earth} \cdot cos(\mathbf{f}_2^{cs}) \cdot sin(\mathbf{f}_3^{cs})$$

$$\mathbf{f}_6^{cs} = r_{earth} \cdot sin(\mathbf{f}_2^{cs})$$

where r_{earth} is the earth's radius of $6,371$ km. Notice that the matrix $\mathbf{F}^{cs}$ has the same length as the matrix $\mathbf{F}^{s}$ (i.e. $length(\mathbf{F}^{cs}) = length(\mathbf{F}^{s}) = 2522$) in the case study adopted. Figure 4.5 shows a scatter plot of all of the 2522 stops.

However, these 2522 stops are far beyond the budget of $n_n = 500$ nodes assumed in the adopted case study where n_n is the total number of system nodes. This large set of stops needs to be refined and optimized such that it ends up with only the optimal set of n_{os} stops. Therefore, n_{os} should satisfy the following:

$$n_{os} = n_n - length(\mathbf{id}_{bl})$$

where $length(\mathbf{id}_{bl})$ gives the total number of blocks or alternatively vehicles which are all assumed to be part of the system given the assumption that content consumers would spend more time in the vehicles compared to stops. Knowing that 253 vehicles are already part of system according to the case study adopted, we have $n_{os} = 500 - 253 = 247$ optimal stops which should also be part of the system. Given n_{os}, stops are refined first using clustering and then they are optimized such that only n_{os} stops are included in the system.

Stops are refined by clustering them using hierarchical clustering. This is done using the function ($cluster$) with the matrix $\mathbf{F}^{cs}[*, 4:6]$ of stop cartesian coordinates as the input. The complete linkage method is assumed and the cutoff distance is set to equal the broadcasting range r_{br} of 300 meters. The resulting vector $\mathbf{id}_{sc}$ represents the vector of cluster IDs for all stops. Computations go through each unique stop cluster in order to identify its medoid. In each stop cluster, the vector of stop IDs $\mathbf{id}_{sc_i}$ belonging to the current cluster sc_i is first extracted using the formula:

$$\mathbf{id}_{sc_i} = \mathbf{F}^{cs}[\underset{j}{argfind}(\mathbf{id}_{sc}[j] = i), 1]$$

where the function ($argfind$) is used to find any index j of a stop that belongs to the current stop cluster i. The coordinates matrix of these stops $\mathbf{CO}_{sc_i}$ is extracted afterwards as follows:

$$\mathbf{CO}_{sc_i} = \mathbf{F}^{cs}[\underset{j}{argfind}(\mathbf{id}_{sc}[j] = i), 2:3]$$

Then, the distance $\mathbf{d}_{sc_i}[j]$ is computed between the coordinate of each cluster member $\mathbf{CO}_{sc_i}[j, *]$ and the cluster coordinates mean found using the formula: $\sum_k \mathbf{CO}_{sc_i}[k, *]/length(\mathbf{CO}_{sc_i})$ using the function ($distance$). The cluster member with the smallest $\mathbf{d}_{sc_i}[j]$ (i.e. cluster medoid) is chosen to be the refined stop representing the current cluster while ignoring the remaining cluster members. The ID of this refined stop is stored as $\mathbf{id}_{rs}[i]$ where $\mathbf{id}_{rs}$ is the vector of all refined stop IDs. At the end, the number of refined stops has been found to be 869 stops (i.e. $length(\mathbf{id}_{rs}) = 869$).

The main motive behind the aforementioned clustering approach is to refine the stops such that those which are within r_{br} of each other are represented using a single refined stop. This should lead most of these refined stops to act as hubs for inter-vehicle communications while avoiding merely having stops in direct contact with each other. Figure 4.6 shows a scatter plot of the resulting 869 refined stops.

After refining the stops, 869 stops are still left which is still larger than the previously-stated optimal stops target of $n_{os} = 247$ stops. To optimize these refined

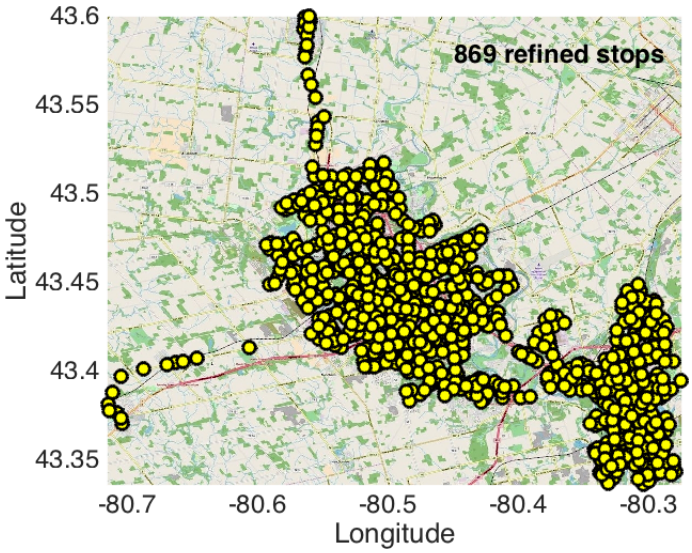


Figure 4.6: Refined stops

stop selections, the vector $\mathbf{pop}_{stops}$ of refined stop popularities is first defined with $length(\mathbf{pop}_{stops}) = length(\mathbf{id}_{rs})$. This vector counts the number of times vehicles pass nearby the refined stops as a measure of their popularities.

In order to determine the values of $\mathbf{pop}_{stops}$, the number of times vehicles pass nearby each refined stop is counted. This is done by iterating through $t = 1 : n_T$ for each block/vehicle and for each t, the coordinates matrix of all active blocks $\mathbf{CO}_{bl}$ is extracted if $\exists k : \mathbf{LATS}_{bl}[k, t] \neq 0$. The distances from all of these blocks or vehicles to the current refined stop coordinates $\mathbf{co}_{s_i}$ are measured using the function $(distance)$ after computing $\mathbf{co}_{s_i}$ using the formula:

$$\mathbf{co}_{s_i} = \mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}_1^s[j] = \mathbf{id}_{rs}[i]), 5 : 6]$$

where the function $(argfind)$ is used to find the index j of matrix $\mathbf{F}^s$ corresponding to the refined stop i. Current refined stop popularity $\mathbf{pop}_{stops}[i]$ is increased by one whenever a vehicle passes nearby refined stop i (i.e. be located $\leq r_{br}$ from the refined stop where $r_{br} = 300$ meters).

After computing all refined stop popularities, these popularities are sorted in a descending order using the function $(sort)$. Based on this order, the first n_{os} stops are chosen as the set of optimal stops with IDs defined by the vector $\mathbf{id}_{os}$. Figure 4.7 shows the refined stop popularities whereas Figure 4.8 shows a scatter plot of the resulting 247 optimal stops.

Before being able to proceed with any further processing, the set of optimal stop coordinates are added to the block latitudes matrix $\mathbf{LATS}_{bl}$ and the block longitudes matrix $\mathbf{LONS}_{bl}$ in order to generate the overall latitudes matrix $\mathbf{LATS}$ and the overall longitudes matrix $\mathbf{LONS}$ for all the nodes, respectively. These two matrices of $\mathbf{LATS}$ and $\mathbf{LONS}$ are needed for the upcoming processing.

Adding the optimal stop coordinates is done such that the first $length(\mathbf{id}_{bl})$ rows

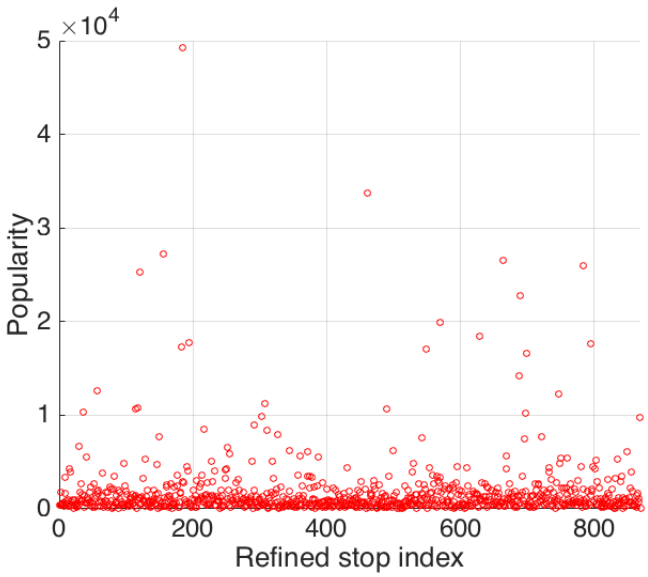


Figure 4.7: Refined stop popularities

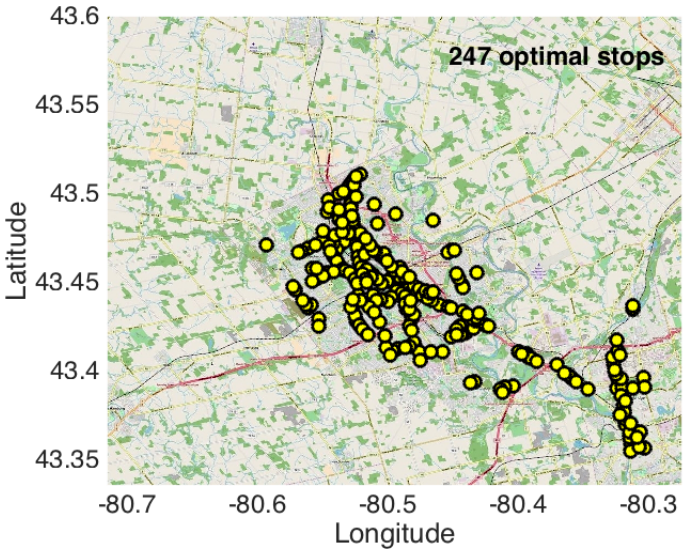


Figure 4.8: Optimal stops

of **LATS** and **LONS** matrices are first assigned the data of $\mathbf{LATS}_{bl}$ and $\mathbf{LONS}_{bl}$ matrices, respectively. This is followed by assigning the next rows with the latitude and longitude data of each optimal stop given that stops are immobile and have fixed coordinates. This is done by using the function (*repmat*) to generate a row containing the repeated latitude $\mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}_1^s[j] = \mathbf{id}_{os}[i]), 5]$ and another row containing the repeated longitude $\mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}_1^s[j] = \mathbf{id}_{os}[i]), 6]$ of each optimal stop i with the ID $\mathbf{id}_{os}[i]$ for n_T number of times. Then, these rows are assigned to the next n_{os} rows of matrices **LATS** and **LONS**.

Refer to Algorithms A.12 to A.15 in Appendix A for further details about this optimization operation of stop nodes selection.

4.3.2 Connectivities Computation

In order to proceed with data processing, connectivities between system nodes need to be detected whenever they occur. This is done by setting first the connectivities matrix $\mathbf{C}$ to the zero matrix of size $(n_T \times n_n \times n_n)$ as follows:

$$\mathbf{C} = \mathbf{0}^{n_T \times n_n \times n_n}$$

After that, pair-wise distances between all n_n nodes are computed over $t = 1 : n_T$ whenever these nodes are active using the function (*distance*) where a node i at time index t is active if $\mathbf{LATS}[i, t] \neq 0$. A connectivity between nodes i and j at time index t occurs such that $\mathbf{C}[t, i, j] = 1$ whenever the distance d between these two nodes is less than or equal to the broadcasting range r_{br} (i.e. $d \leq r_{br}$). Notice that $\mathbf{C}[t, i, j] = \mathbf{C}[t, j, i]$ due to the connectivities matrix $\mathbf{C}$ symmetry and that $r_{br} = 300$ meters as assumed in the adopted case study.

The final outcome is the connectivities matrix $\mathbf{C}$ such as $\mathbf{C}[t, i, j] = 1$ if nodes i and j are within r_{br} of each other at time t and $\mathbf{C}[t, i, j] = 0$, otherwise. Figure 4.9 shows a snapshot of the resulting node connectivities at 5:00 PM. This snapshot zooms into the downtown area of Kitchener, Ontario, Canada which is served by the Grand River Transit service addressed in the case study adopted. Notice that green circles represent vehicles, blue circles represent stops and red lines indicate connectivities whenever they occur between the system nodes.

Refer to Algorithm A.16 in Appendix A for further details about this connectivities computation operation.

4.3.3 Networking Potential Evaluation

Before clustering system nodes in the next subsection, the networking potential of these nodes needs to be evaluated. This is achieved by conducting the following steps which are included for the sake of completeness only and are not part of the proposed content distribution procedure:

- Instant-clustering the vehicles under different minimum degrees of connectivity,

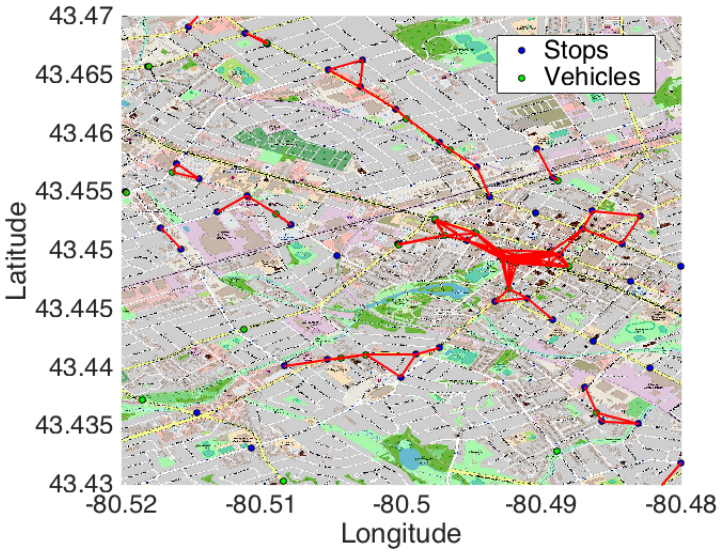


Figure 4.9: Connectivities at 5:00 PM

- Instant-clustering the vehicles under different broadcasting ranges, and
- Computing continuous contact durations.

Instant-clustering in the first two steps has been discussed in my previous work presented in [50]. It differs from the clustering discussed in the next subsection in that it seeks to discover node clusters at certain instants only which are the 24 hours of the day. It does not identify nodes which should be grouped together for a continuous period of time such that content is routed efficiently to them. It is only used to give some insights into the promise of nodes clustering throughout the day with a focus on the more challenging and dynamic vehicle nodes. Such insights allow for better evaluation of the V2V networking potential between system vehicle nodes.

The method of instant-clustering used is hierarchical clustering. This clustering technique is chosen for the first step because it allows for variations in the minimum degree of connectivity by adjusting the linkage method between the two extreme methods of single linkage and complete linkage. Notice that the single linkage method corresponds to less communication reliability where each vehicle has at least one connection with another same-cluster vehicle member. On the other hand, the complete linkage method corresponds to more communication reliability where each vehicle is completely linked to all the other same-cluster vehicle members which might be needed under harsh channel conditions.

Hierarchical clustering is chosen for the second step because it allows for variations in the broadcasting range by adjusting the cutoff distance between the two extreme values of 300 meters and 1000 meters as set by the DSRC standard.

Throughout the aforementioned instant-clustering steps, the following two indicators are used:

- number of clusters; which corresponds to the number of cluster head nodes

such that having less of these nodes increases the chances of having congestions. However, it does also mean less reliance on other network tiers if these cluster head nodes were to act as gateways,

- and cluster size distributions; which shows cluster size distributions using boxplots such that having cluster sizes with larger means or more vehicle nodes increases the chances of congestions at cluster head nodes. Moreover, having cluster sizes with larger inter-quartile ranges means less fairness since some node clusters will have more vehicle nodes and therefore a higher chance of experiencing congestions and a longer time for messages to reach all cluster vehicle nodes.

In the third step mentioned above, continuous contact durations are computed under the more challenging and dynamic scenario of vehicle nodes only and then under the scenario of the case study adopted using all the $n_n = 500$ nodes including the 253 vehicles and the 247 stops. Such contact durations have a direct impact on the networking potential of the proposed system as longer continuous contact durations correspond to a higher chance of successful V2V communication. Notice that the only indicator used in this third step is the contact durations distribution.

Instant-clustering of Vehicles Under Different Minimum Degrees of Connectivity

Assuming the broadcasting range of 1000 meters, the minimum degree of connectivity is varied between same-cluster vehicles using hierarchical clustering by varying the linkage method from single to complete.

As it can be seen in Figure 4.10, the resulting number of vehicle clusters is much lower than the number of vehicles. Moreover, many vehicle nodes are part of a cluster that has more than one node as indicated by the boxplots shown in Figures 4.11a and 4.11b. In particular, the single linkage method has led to: fewer clusters, bigger cluster sizes and larger cluster size inter-quartile ranges. This means fewer cluster heads, less reliance on other network tiers, more cluster head congestions, more time for messages to reach all cluster nodes and less fairness. This is in addition to less channel redundancy and therefore reliability. On the other hand, the complete linkage method has led to: more clusters, smaller cluster sizes and smaller cluster size inter-quartile ranges. This means more cluster heads, more reliance on other network tiers, fewer cluster head congestions, less time for messages to reach all cluster nodes and more fairness. This is in addition to more channel redundancy/reliability.

Instant-clustering of Vehicles Under Different Broadcasting Ranges

Assuming the average linkage method, the broadcasting range is varied using hierarchical clustering by adjusting the cutoff distance from 300 to 1000 meters. The number of clusters throughout the day under the two broadcasting ranges is

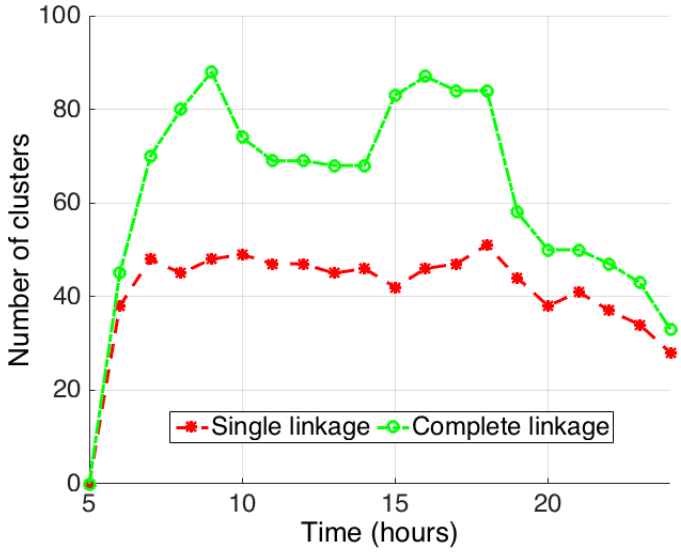
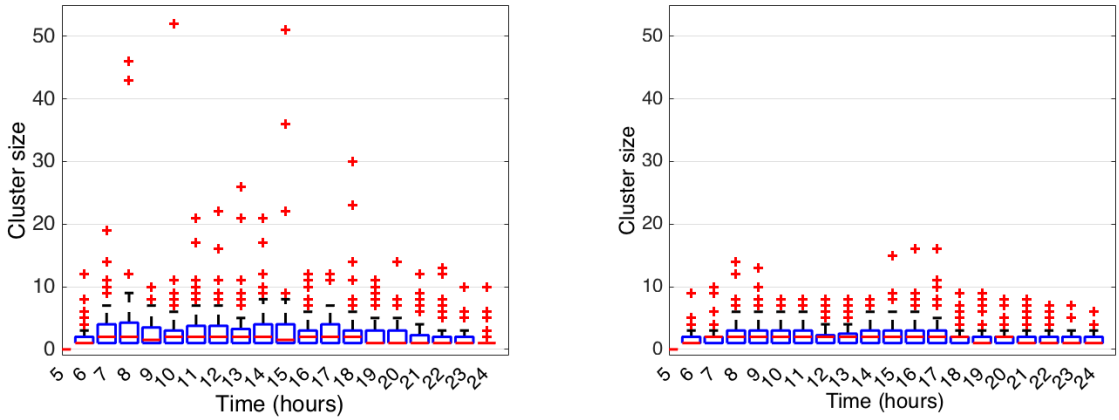


Figure 4.10: Number of clusters vs. time under different linkage methods



(a) Single linkage

(b) Complete linkage

Figure 4.11: Cluster size distribution vs. time under different linkage methods

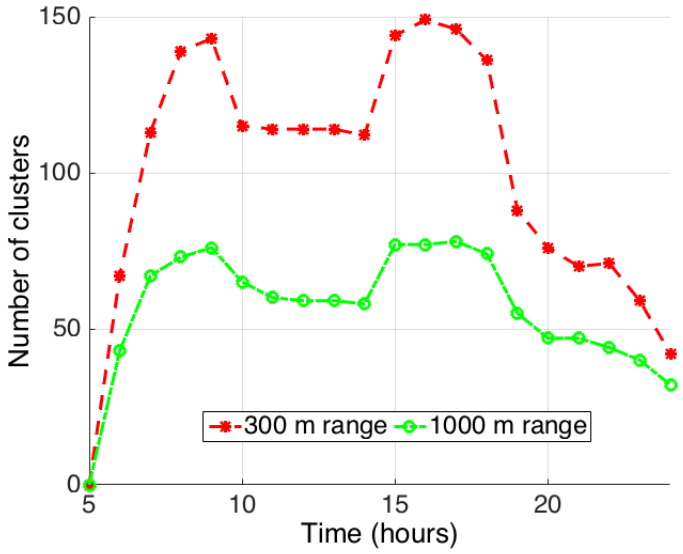


Figure 4.12: Number of clusters vs. time under different broadcasting ranges

shown in Figure 4.12 whereas the cluster size distributions under both ranges are shown in Figures 4.13a and 4.13b.

On one hand, the 300 meter broadcasting range has led to: more clusters, smaller cluster sizes and smaller cluster size inter-quartile ranges. This means more cluster heads, more reliance on other network tiers, fewer cluster head congestions, more fairness and a lower V2V communication probability.

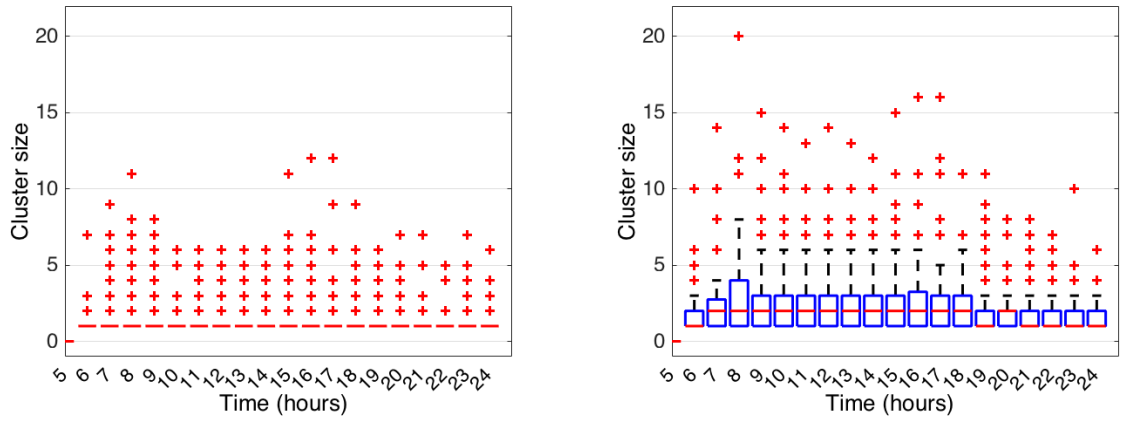
On the other hand, the 1000 meter broadcasting range has led to: fewer clusters, larger cluster sizes and larger cluster size inter-quartile ranges. This means fewer cluster heads, less reliance on other network tiers, more cluster head congestions, less fairness and a higher V2V communication probability.

In general, the resulting number of vehicle clusters is much lower than the number of vehicles. Moreover, many vehicles are part of a cluster that has more than one node as indicated by the boxplots shown in Figures 4.13a and 4.13b.

Continuous-contact Duration Computations

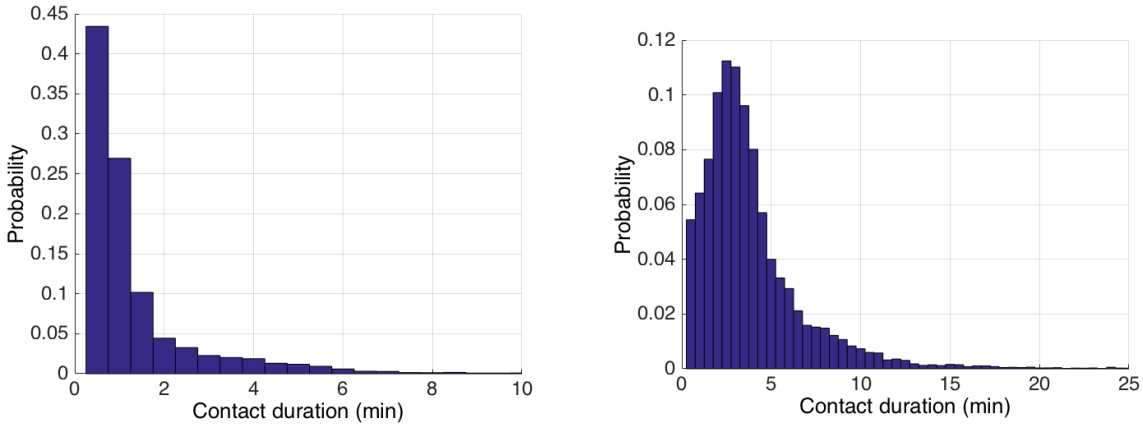
At the beginning, continuous contact durations are computed for the more challenging and dynamic scenario of vehicle nodes only. In this scenario, continuous contact durations throughout the day are computed and their distributions are drawn under the two broadcasting ranges of 300 and 1000 meters as shown in Figures 4.14a and 4.14b, respectively. As it can be seen, the vehicle contact duration has a mean as high as 1.38 minutes under the 300 meter broadcasting range and an even higher mean of 3.89 minutes under the 1000 meter broadcasting range.

Given these promising duration results, continuous contact durations are computed for the case study adopted with a total of $n_n = 500$ nodes including 253 vehicle nodes and 247 stop nodes. These durations are computed under the



(a) 300 m-range (b) 1000 m-range

Figure 4.13: Cluster size distribution vs. time under different broadcasting ranges



(a) 300 m-range (b) 1000 m-range

Figure 4.14: Daily continuous contact duration distribution of vehicles

different n_p periods where $\mathbf{CD}\{p\}$ is the matrix of continuous contact durations in period p. These computations are made given the connectivities matrix $\mathbf{C}$ produced previously. Notice that in the case study adopted, the number of periods n_p is 11 periods where each period p lasts for 2 hours. The starting time indices of these periods are defined using the vector $\mathbf{t}^{start}$ whereas the ending time indices are defined using the vector $\mathbf{t}^{end}$ such as:

$$\mathbf{t}^{start} = [t_1^{start} \;\; t_2^{start} \;\; ... \;\; t_{n_p}^{start}]^{1\times n_p} = [5 \;\; 7 \;\; ... \;\; 25]\, hrs \times 60\, mins \times 60\, secs/10$$

$$\mathbf{t}^{end} = [t_1^{end} \;\; t_2^{end} \;\; ... \;\; t_{n_p}^{end}]^{1\times n_p} = [7 \;\; 9 \;\; ... \;\; 27]\, hrs \times 60\, mins \times 60\, secs/10$$

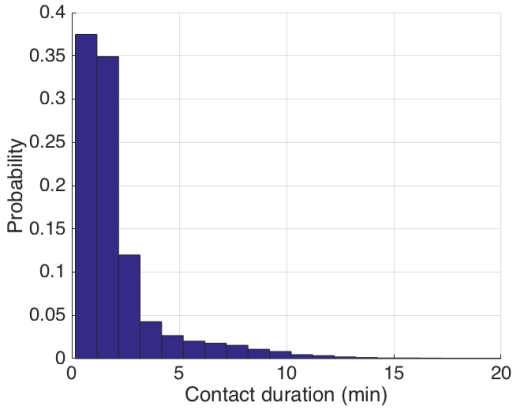
The division by 10 is made given that 10 seconds is the shortest time interval assumed in the case study under consideration.

Continuous contact duration computations proceed through periods 1 to n_p such that for each period p, $\mathbf{CD}\{p\}$ is set initially to $\emptyset$. Then, these computations go through nodes 1 to n_n starting with $\mathbf{cd} = \mathbf{0}^{1\times n_n}$ for each node where $\mathbf{cd}$ is the vector of continuous contact duration counters. These counters are increased by the contact duration step size $\triangle cd$ (i.e. $\mathbf{cd}[j] \leftarrow \mathbf{cd}[j] + \triangle cd$) whenever there is a neighboring node j such that $\mathbf{C}[t, i, j] = 1$ within the period indices $\mathbf{t}^{start}[p] : \mathbf{t}^{end}[p]$ where p is the current period index. Notice that $\triangle cd$ is set to 10 seconds in the case study adopted. Notice also that whenever $\mathbf{C}[t, i, j] = 0$, then the neighboring node j has either never started being in contact with node i or lost its continuous contact with node i given that $t \neq \mathbf{t}^{start}[p]$ and the fact that it used to be in contact at the previous time step (i.e. $\mathbf{C}[t-1, i, j] = 1$). Given the latter case, the current contact duration $\mathbf{cd}[j]$ is added to the matrix of continuous contact durations $\mathbf{CD}\{p\}$ while resetting it to 0 afterwards. This way, these computations end up with the set of matrices of continuous contact durations $\mathbf{CD}\{1 : n_p\}$ for all periods $1 : n_p$.

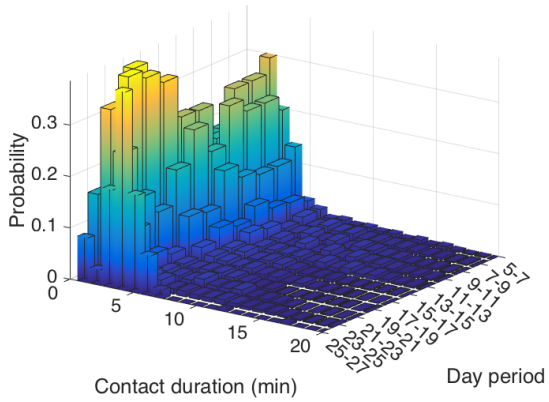
Figure 4.15a shows the distribution of continuous contact durations throughout the whole workday. As it can be seen, the majority of these durations is less than 3 minutes. Figure 4.15b shows the same kind of distribution but for each period of the day separately. It can be seen that periodical distributions are quite similar with the vast majority of durations have lengths below 5 minutes. However, periods between 7:00 AM and 9:00 PM tend to have slightly longer durations. This can be seen more easily in the boxplots of Figure 4.15c. On the other hand, Figure 4.15d shows the first, the second and the third quartiles of these boxplot distributions. The average is almost always about 1.5 minutes for all of these periodical distributions.

These promising continuous contact durations show clearly the networking potential of public transportation vehicles and stops and have encouraged me to continue investigating contact durations while tolerating discontinuous ones this time. Exploiting long discontinuous contact durations by cumulating them and grouping or clustering the corresponding nodes allows for a more efficient content routing given the delay-tolerant nature of popular content distribution as we shall see. Therefore, the next subsection addresses nodes clustering under cumulative contact duration thresholds.

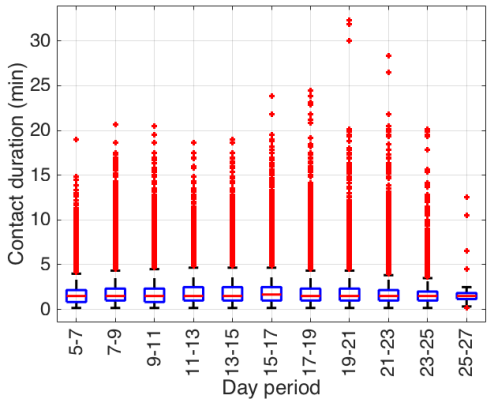
Refer to Algorithm A.17 in Appendix A for further details about this continuous-contact duration computations operation.



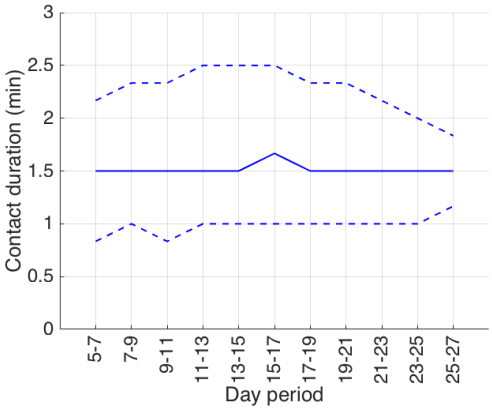
(a) Daily distribution



(b) Distribution vs. day period



(c) Distribution boxplot vs. day period



(d) Distribution quartiles vs. day period

Figure 4.15: Continuous contact durations

4.3.4 Nodes Clustering

Given the promising continuous contact duration results found in the previous subsection, nodes clustering is addressed in this subsection to identify the different node groups to be targeted for content routing as will be presented in Chapter 6. A clustering technique is discussed in this regard which relies on the concept of a minimum cumulative contact duration threshold. This threshold is defined as the minimum cumulative contact duration which a node should have with at least another cluster node to be considered within the same cluster.

This clustering technique is part of the offline operations made throughout the proposed content distribution procedure. It is based on finding the biggest nodes cluster under a certain cd_{min} value and then removing its node connectivities from the connectivities matrix $\mathbf{C}$ before proceeding to the next biggest nodes cluster under another cd_{min} value within the same period p. This approach can lead to all system nodes being clustered under different cd_{min} values. The focus on biggest clusters is made here given the fact that these node clusters have the highest potential in terms of the amount of content that can be distributed.

In order to proceed with the explanation of this clustering approach, the function ($extractCluster$) is explained first. This function extracts the c^{th} biggest nodes cluster in terms of its members during period p. It is used to extract the first biggest cluster with $c = 1$ as well as extracting the next biggest clusters after removing the connectivities of the previous biggest node clusters.

Given a particular period p that starts at t^{start} and ends at t^{end}, the function ($extractCluster$) starts by scanning nodes 1 to n_n in order to compute the vector of instantaneous connectivity summations $\mathbf{c}_p$ each node i has with its neighbors throughout period p starting at $\mathbf{t}^{start}[p]$ and ending at $\mathbf{t}^{end}[p]$. This is done using the formula:

$$\mathbf{c}_p = \sum_{t=\mathbf{t}^{start}[\mathbf{p}]}^{\mathbf{t}^{end}[p]} \mathbf{C}[t, i, *]$$

where $\mathbf{C}$ is the connectivities matrix found previously. Given $\mathbf{c}_p$, the matrix of node cluster i member indices $\mathbf{ID}_{nc}^{prvs}\{i\}$ is found as follows:

$$\mathbf{ID}_{nc}^{prvs}\{i\} = \underset{j}{argfind}(\mathbf{c}_p[j] \times \triangle cd \geq cd_{min})$$

where the function ($argfind$) is used to find the index j of any node cluster i member that satisfies the condition:

$$\mathbf{c}_p[j] \times \triangle cd \geq cd_{min}$$

Notice that $\mathbf{ID}_{nc}^{prvs}\{i\}$ accounts only for those first-hop neighbors of node i which meet the cumulative contact duration threshold cd_{min}. This means that the neighbors of those neighbors (i.e. second-hop neighbors of node i), which do also meet the cd_{min} condition, are not included in $\mathbf{ID}_{nc}^{prvs}$ which should not be the case. In order to overcome this and include any node satisfying the cd_{min} condition while being at maximum $n_{h_{max}}$ loop-free hops away from node i, the

function ($extractCluster$) creates first the set of matrices $\mathbf{ID}_{nc}^{nxt}$ such that $\mathbf{ID}_{nc}^{nxt} = \mathbf{ID}_{nc}^{prvs}$. These $\mathbf{ID}_{nc}^{nxt}$-matrices are used to store the next-iteration member indices of all node clusters compared to the set of matrices $\mathbf{ID}_{nc}^{prvs}$ which is used to store the previous-iteration member indices of all node clusters. These two sets are essential for the multi-hop neighbor inclusion iterations of function ($extractCluster$) in which the previous-iteration set of node neighbors $\mathbf{ID}_{nc}^{prvs}\{i\}$ are scanned to include their neighbors to be part of the next-iteration set of node neighbors $\mathbf{ID}_{nc}^{nxt}\{i\}$ as follows:

$$\mathbf{ID}_{nc}^{nxt}\{i\} \leftarrow \mathbf{ID}_{nc}^{nxt}\{i\} \cup \mathbf{ID}_{nc}^{prvs}\{\mathbf{ID}_{nc}^{prvs}\{i\}[j]\}$$

where j is the current neighbor index of node i according to $\mathbf{ID}_{nc}^{prvs}\{i\}$. This continues at the current hop index h until all nodes with index $i : i \in \{1, 2, ..., n_n\}$ are included. By then, the function ($extractCluster$) moves to the next iteration after setting $\mathbf{ID}_{nc}^{prvs}$ to equal $\mathbf{ID}_{nc}^{nxt}$.

By the end of these iterations, function ($extractCluster$) will have all the multi-hop neighbors for all n_n nodes such that they are less than $n_{h_{max}}$ loop-free hops away from their corresponding nodes. These neighbors do also meet the cd_{min} condition which means that all of them have spent at least a duration of cd_{min} with at least one of node i's neighbors included in the set $\mathbf{ID}_{nc}^{nxt}$. Figure 4.16 shows the sequence diagram throughout period p of a representative nodes clustering scenario with 5 nodes, $n_{h_{max}} = 4$ hops and the duration of $(2 \times cd_{min})$ as the minimum cumulative contact duration.

Function ($extractCluster$) proceeds with identifying the vector of node cluster indices $\mathbf{id}_{nc}$ in addition to the matrix of member indices for each node cluster $\mathbf{ID}_{nc}$ within period p. This is in order to compute the node cluster sizes as given by the vector $\mathbf{sizes}_{nc}$ using the formula:

$$\mathbf{sizes}_{nc}[i] = length(\mathbf{ID}_{nc}\{i\})$$

where i is the node cluster index. This enables the identification of the member node indices of the biggest node cluster $\mathbf{ID}_{nc}^{b}\{c\}$ within period p as follows:

$$\mathbf{ID}_{nc}^{b}\{c\} = \mathbf{ID}_{nc}\{\underset{i}{argmax}(\mathbf{sizes}_{nc}[i])\}$$

where the function ($argmax$) is used to find the index i that gives the biggest nodes cluster size.

After the first biggest cluster, extractions proceed with the next biggest clusters. This is done by going over the vector $\mathbf{cd}_{min}$ of minimum cumulative contact duration thresholds used for each biggest cluster. The total number of time steps $n_{T_{period}}$ is computed first within the period under consideration given t^{start} and t^{end} as follows:

$$n_{T_{period}} = t^{end} - t^{start} + 1$$

then, the connectivities of all node members belonging to the previous biggest cluster with index c are removed by setting them to the corresponding zero matrix. This is done for all $n_{T_{period}}$ time steps as follows:

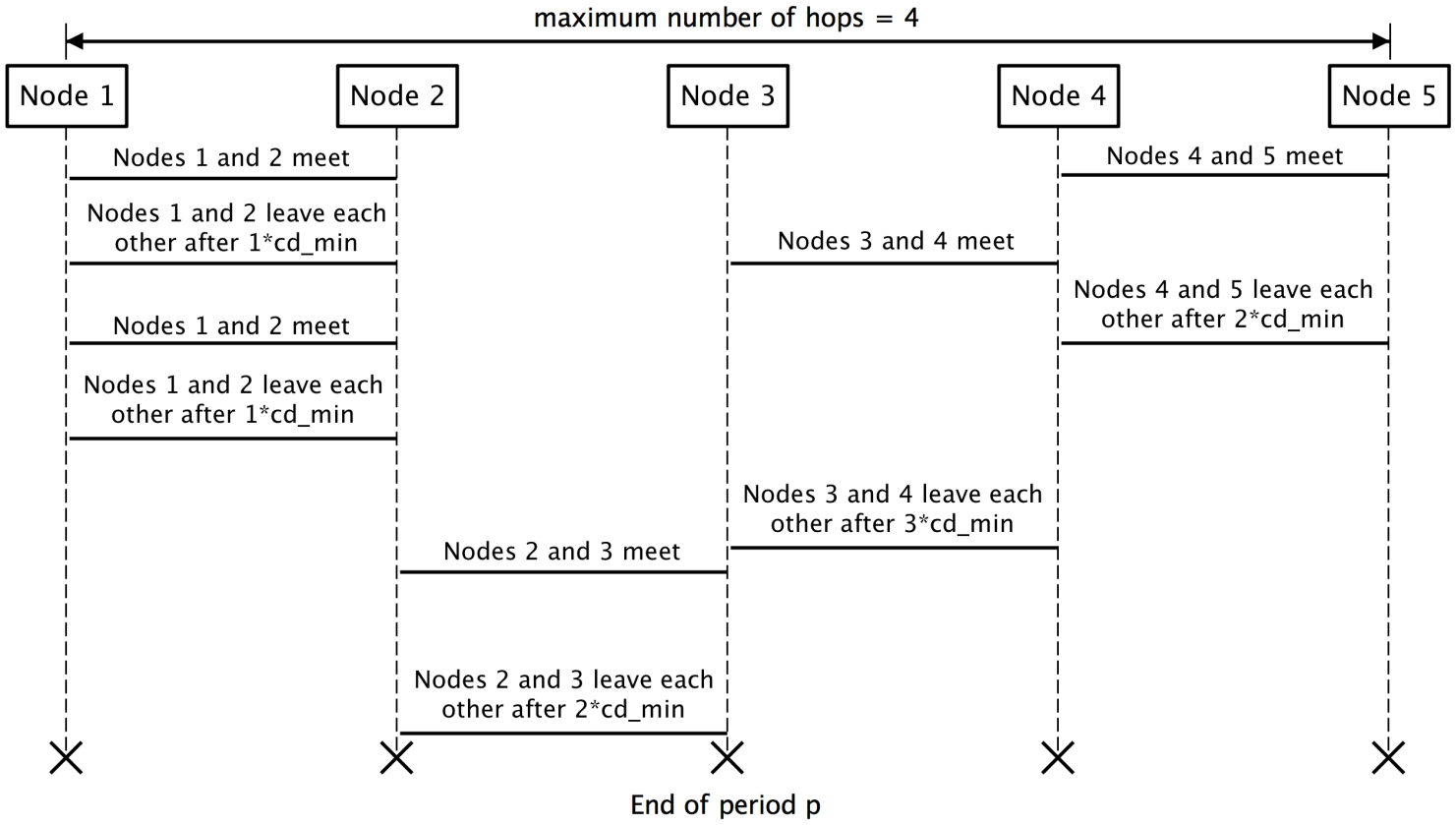


Figure 4.16: Representative nodes clustering scenario

$$\mathbf{C}[t, \mathbf{ID}^b_{nc}\{c\}, *] = \mathbf{0}^{length(\mathbf{ID}^b_{nc}\{c\}) \times n_n}$$

$$\mathbf{C}[t, *, \mathbf{ID}^b_{nc}\{c\}] = \mathbf{0}^{n_n \times length(\mathbf{ID}^b_{nc}\{c\})}$$

where $\mathbf{C}$ is the connectivities matrix, t is the period time index, the matrix $\mathbf{ID}^b_{nc}\{c\}$ is for the member IDs of the previous c^{th} biggest nodes cluster and n_n is the total number of nodes. This update of the matrix $\mathbf{C}$ in done before proceeding with the extraction of the next biggest cluster of index $(c+1)$ using the function $(extractCluster)$. Notice that each extraction produces the matrix $\mathbf{ID}^b_{nc}\{c\}$ of the current c^{th} biggest nodes cluster being extracted.

After going over all the biggest cluster extractions at period p, the matrix $\mathbf{ID}_{nc}\{p\}$ of cluster member IDs of the different biggest clusters at period p will be found:

$$\mathbf{ID}_{nc}\{p\} = \mathbf{ID}^b_{nc}$$

This clustering technique continues for all periods 1 to n_p according to the corresponding vector $\mathbf{cd}_{min}$ of cumulative contact duration thresholds for each of these periods. This vector $\mathbf{cd}_{min}$ should always be specified by the designer in a way that leaves no nodes unclustered. Notice that all of the matrices $\mathbf{ID}_{nc}\{1 : n_p\}$ will be inquired throughout the content distribution procedure before content routing as will be presented in Chapter 6.

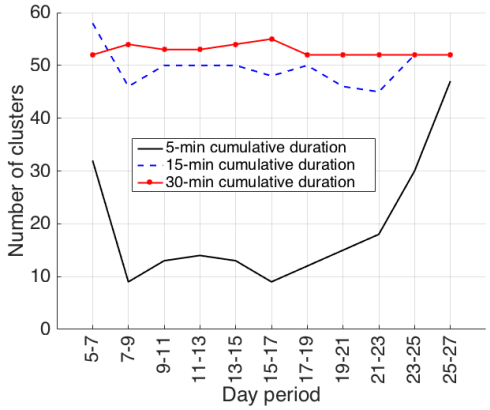
Figure 4.17a shows the resulting number of clusters throughout the day periods under the fixed cd_{min} values of 5, 15 and 30 minutes throughout the biggest clusters extracted throughout the periods where $n_{h_{max}}$ is always set to 20 hops. It can be seen that the number of clusters increases as cd_{min} increases while decreasing the most during busy periods of the day. On the other hand, Figure 4.17b shows the number of unclustered nodes under the same set of cd_{min} and $n_{h_{max}}$ values. It can be seen here that this number increases as cd_{min} increases while decreasing the most during busy periods of the day.

Figures 4.17c and 4.17d show the cluster size means and maxima throughout the day periods, respectively. It can be seen in these figures that the cluster size mean as well as the maximum increases as the value of cd_{min} decreases with highest size means and maxima during busy periods of the day.

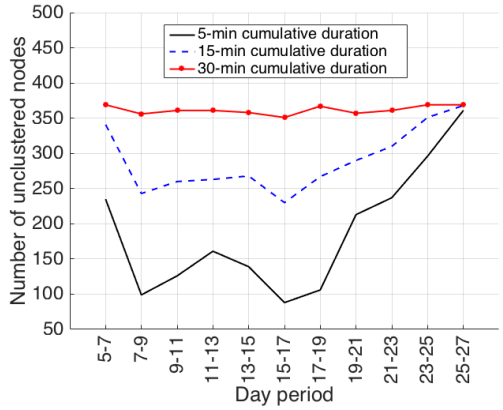
All of these results are attributed to the fact that increasing cd_{min} makes it harder for node clusters to grow and therefore results in decreasing the cluster sizes, their means and maxima. In addition, the number of clusters increases while the number of unclustered nodes gets higher. Notice that these cluster trends become even more evident during busy day periods when more vehicles are in the streets with higher densities.

Figures 4.18a and 4.18b confirm this interpretation by showing a snapshot of the node clusters at the most busy hour of the day (i.e. 5:00 PM) under the cd_{min} values of: 5 and 30 minutes, respectively where yellow circles represent unclustered system nodes and other-color circles represent the different node clusters.

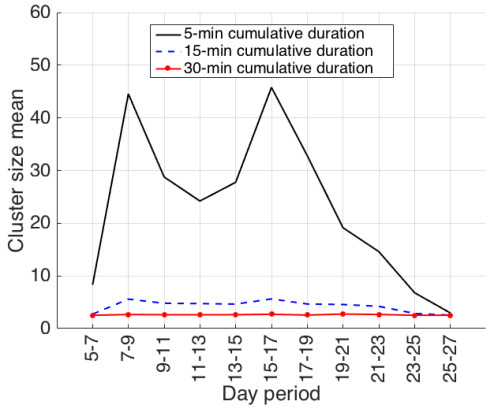
Given the aforementioned clustering results, it is clear that having a fixed cd_{min} for all the clusters within the same period is not always the right thing to do. This is due to the fact that large numbers of nodes are left unclustered. It is also less



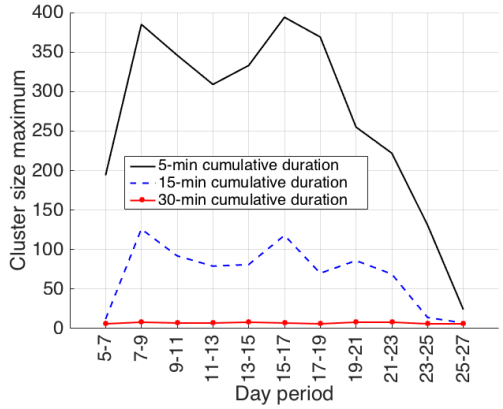
(a) No. of clusters vs. day period



(b) No. of unclustered nodes vs. day period

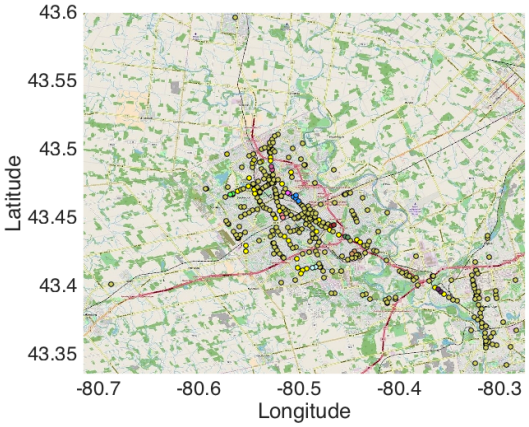


(c) Cluster size mean vs. day period

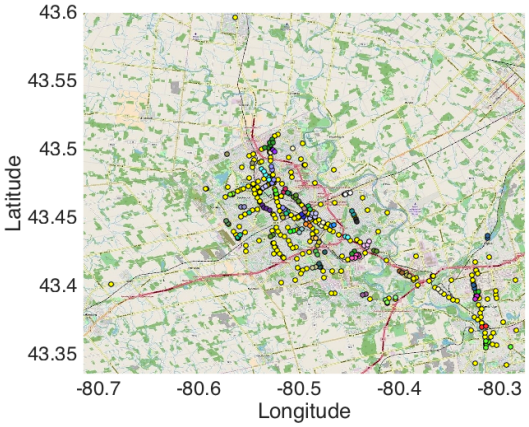


(d) Cluster size maximum vs. day period

Figure 4.17: Nodes clustering



(a) $cd_{min} = 5$ minutes



(b) $cd_{min} = 30$ minutes

Figure 4.18: Clustering under different cd_{min} thresholds

flexible because it forces the designer to choose between either a cd_{min} value that results in a high number of clusters, a high number of unclustered nodes and small cluster sizes or another cd_{min} value that results in the opposite situation. Therefore, the vector $\mathbf{cd}_{min}$ should always be specified by the designer in a way that leaves no nodes unclustered.

Refer to Algorithms A.18 and A.19 in Appendix A for further details about this nodes clustering operation.

4.4 Summary

In this chapter, offline operations related to mobility analysis have been explained in details. These operations have been categorized into: preprocessing and processing operations. The preprocessing operations include: data collection, data sorting, data cleaning and data synthesis whereas the processing operations include: stop nodes selection optimization, connectivities computation and nodes clustering. Other operations related to evaluating the networking potential of public transportation vehicles and stops have also been discussed for the sake of completeness but are not part of the proposed content distribution procedure as indicated throughout the chapter. The outcome of the most recent operation discussed, which is the clustering of system nodes, is the most essential for the upcoming content recommendation and routing discussions to be presented in Chapters 5 and 6, respectively.

Chapter 5

Content Recommender Design

5.1 Overview

Designing a recommender that performs well under different consumer interest and network capacity scenarios is critical for the overall performance of the proposed content distribution system. It allows for efficient interactions with the consumers by exploring their interests in as fewer interactions as possible while prioritizing distributed services content according to the true service popularities among consumers. Therefore, this chapter shows the different steps followed in order to design such a recommender. Notice that the recommendation operations have already been indicated in Lines 3 to 6 and Lines 19 to 20 as part of the online operations shown in Algorithm 3.2. Also notice that the results of this chapter are generic and are not restricted to any specific case study. Finally, notice that the performance of recommender designs discussed in this chapter does not depend on the absolute consumer locations at the different n_n system nodes assuming that proper clustering of the consumer nodes is already provided. In fact, the IDs of cluster nodes in which those consumers are located at throughout the different n_p periods have already been provided throughout the nodes clustering operation discussed previously in Chapter 4.

This chapter starts with synthesizing the matrix $\mathbf{INT}_{true}$ of true consumer interests and the matrix $\mathbf{INT}_{avail}$ of available consumer interests. $\mathbf{INT}_{true}$ is considered to be the ground truth when it comes to consumer interests whereas $\mathbf{INT}_{avail}$ is a varying matrix over time depending on the recommender being used. These two matrices are synthesized given the fact that finding realistic ones which correspond exactly to the adopted case study is extremely difficult, if not impossible, considering the privacy regulations associated with such information. In addition, these two matrices are necessary to evaluate the different recommender designs. Therefore, these two matrices are synthesized for a total number of n_c consumers clustered into n_g groups and a total number of n_s services.

Given the matrices $\mathbf{INT}_{true}$ and $\mathbf{INT}_{avail}$, the design steps followed to come up with the proposed recommender are discussed. These steps start with the most basic recommender and gradually enhance it. Therefore, different recommenders are

created which can be classified into the following four main categories:

- Non–interactive, non-collaborative and non-group-based category,
- Interactive, non-collaborative and non-group-based category,
- Interactive, collaborative and non-group-based category, and
- Interactive, collaborative and group-based category.

Each interactive category has different AI-based techniques implemented based on multi-armed bandits. These techniques decide the nature of consumer interactions and they include: the Greedy search technique, the ϵ-greedy search technique, the Decaying ϵ-greedy search technique and the UPB search technique. For collaborative categories, recommenders implemented filter services based on collaborating consumer interest profiles. For group-based categories, location-based groups of consumers are taken into account.

Throughout the chapter, the purpose and mathematical formulations of all of these recommenders and their functions are explained in details while leaving their algorithms to Appendix B at the end of the thesis.

All of these recommenders are compared in terms of their performance against time which is set to end at $n_{T_{rcmnd}}$. This $n_{T_{rcmnd}}$ represents the number of time intervals in which these recommenders operate. These intervals represent the day periods which do not need to be successive and can represent the same period of the day but at different days.

The performance indicators used to evaluate the recommenders are the ratio of consumer interests explored and the ratio of truly popular service content distributed. These two indicators give insights about the exploration and exploitation aspects of the different recommenders, respectively.

The comparisons between the different recommenders are made under different consumer interest and network scenarios. In particular, the comparisons are made under varying unknown interest ratios, varying group interest distributions and varying network capacities.

The unknown interests ratio is denoted by $ratio_{ui}$ and it measures the ratio of unknown consumer interests in the $\mathbf{INT}_{avail}$ matrix at the first time step. The group interests distribution is parameterized using its standard deviation denoted by σ_i^g. The network capacity is denoted by cap_{net} and it measures the maximum number of services which can be distributed at each time step.

By the end of the chapter, the best recommender is chosen given the extensive experiments conducted under the different consumer interest and network capacity scenarios. This recommender is the UPB collaborative and group-based recommender and is used as part of the online operations made by the proposed content distribution procedure as explained previously in Chapter 3.

5.2 Consumer Interest Profiles Synthesis

The true consumer interests matrix $\mathbf{INT}_{true}$ is synthesized such as $\mathbf{INT}_{true}[i,j] = 1$ if consumer i is truly interested in service j and $\mathbf{INT}_{true}[i,j] = 0$, otherwise. All of the $(n_c \times n_s)$ values of this matrix are initially set to zero. Then, consumer interest values of all n_g groups are generated. For each group, the first things to do are identifying the first group consumer index id_c^f, the last group consumer index id_c^l, the first group service index id_s^f and the last group service index id_s^l as follows:

$$id_c^f = n_c/n_g \times (g-1) + 1$$

$$id_c^l = n_c/n_g \times g$$

$$id_s^f = n_s/n_g \times (g-1) + 1$$

$$id_s^l = n_s/n_g \times g$$

where g is the current consumer group index. Then, the vector $\mathbf{id}_{\mathbf{INT}_{true}}$ is synthesized such that it represents the indices of true interests with value 1 for each group consumer. This vector is produced using the rounded values of function $(normrnd)$ which generates the aforementioned indices given the normal distribution of mean $\parallel (id_s^f + id_s^l)/2 \parallel$ and standard deviation σ_i^g. The length of this vector is n_s/n_g and it satisfies the following:

$$1 \leq \mathbf{id}_{\mathbf{INT}_{true}}[i] \leq n_s \forall i : i \in \{1, 2, ..., n_s/n_g\}$$

given that:

$$\mathbf{id}_{\mathbf{INT}_{true}}[\underset{i}{argfind}(\mathbf{id}_{\mathbf{INT}_{true}}[i] < 1)] = \emptyset$$

$$\mathbf{id}_{\mathbf{INT}_{true}}[\underset{i}{argfind}(\mathbf{id}_{\mathbf{INT}_{true}}[i] > n_s)] = \emptyset$$

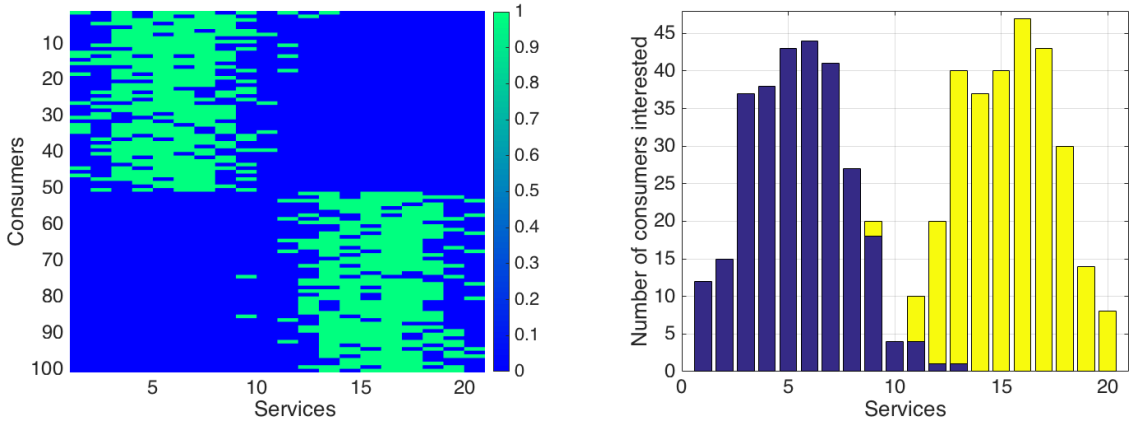
where the function $(argfind)$ is used to find any index i that results in $\mathbf{id}_{\mathbf{INT}_{true}}[i] < 1$ or $\mathbf{id}_{\mathbf{INT}_{true}}[i] > n_s$. With $\mathbf{id}_{\mathbf{INT}_{true}}$, the interests of value 1 in vector $\mathbf{INT}_{true}[j, *]$ are identified and assigned the value 1 using the formula:

$$\mathbf{INT}_{true}[j, \mathbf{id}_{\mathbf{INT}_{true}}] = \mathbf{1}^{1 \times length(\mathbf{id}_{\mathbf{INT}_{true}})}$$

where j is the current group consumer index.

In order to make sure that at least one service is truly liked by each consumer, all n_c consumers are scanned and whenever a consumer i has no liked service (i.e. $\forall j, \mathbf{INT}_{true}[i,j] = 0$), the value 1 is assigned to a true service interest chosen at random using the function $(randi)$ as follows:

$$\mathbf{INT}_{true}[i, randi(\lfloor i/(\tfrac{n_c}{n_g}) \rfloor \times (\tfrac{n_s}{n_g}) + 1, (\lfloor i/(\tfrac{n_c}{n_g}) \rfloor + 1) \times (\tfrac{n_s}{n_g}))] = 1$$



(a) Interests matrix

(b) Group interest distributions

Figure 5.1: True consumer interests with $\sigma_i^g = 2$

where the function $(randi)$ is used to generate a random integer between $(\lfloor i/(\frac{n_c}{n_g})\rfloor \times (\frac{n_s}{n_g}) + 1)$ and $((\lfloor i/(\frac{n_c}{n_g})\rfloor + 1) \times (\frac{n_s}{n_g}))$ representing the service chosen at random for consumer i to be truly liked.

Figure 5.1a shows the heat map of the true consumer interests matrix $\mathbf{INT}_{true}$ produced given that: $n_c = 100$ consumers, $n_g = 2$ groups, $n_s = 20$ services and $\sigma_i^g = n_s/10 = 2$. Figure 5.1b shows the corresponding consumer interest distributions for the 2 consumer groups in blue and yellow colors.

On the other hand, Figure 5.2a shows the heat map of the true consumer interests matrix $\mathbf{INT}_{true}$ produced given that: $n_c = 100$ consumers, $n_g = 2$ groups, $n_s = 20$ services and $\sigma_i^g = n_s/2 = 10$. Figure 5.2b shows the corresponding consumer interest distributions for the 2 consumer groups.

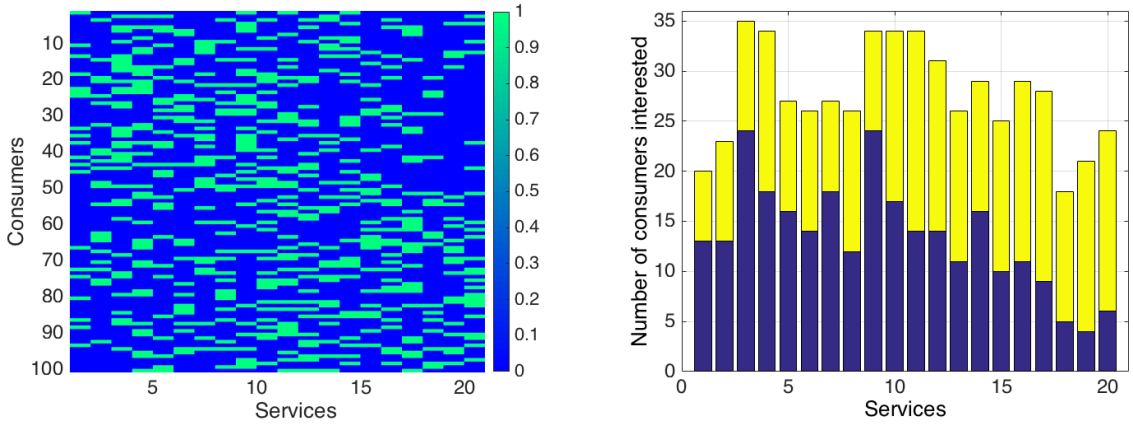
Notice that the consumer true interests under $\sigma_i^g = 2$ have narrow distributions across the 20 services which makes the 2 groups more distinct in terms of the set of popular services within each group. In contrary, the consumer true interests under $\sigma_i^g = 10$ have wide distributions across the 20 services which makes the 2 groups almost indistinguishable in terms of the set of popular services within each group.

Given $\mathbf{INT}_{true}$, another matrix is synthesized such that it shows only those true interests which are currently available. This matrix is called the available consumer interests matrix $\mathbf{INT}_{avail}$ and it is the matrix that would be used by the different recommenders. Notice that $\mathbf{INT}_{avail}[i,j] = 1$ whenever consumer i is currently known to be interested in service j and $\mathbf{INT}_{avail}[i,j] = 0$, otherwise.

To generate $\mathbf{INT}_{avail}$, $\mathbf{INT}_{avail}$ is first set to equal $\mathbf{INT}_{true}$. Then, all n_c consumers and all of their n_s services are scanned in order to set $\mathbf{INT}_{avail}[i,j]$ to be NaN whenever the following condition is met:

$$randn \leq ratio_{ui}$$

where i is the consumer index, j is the service index, $randn$ is a function generating a uniformly random number $\in [0,1]$ and $ratio_{ui}$ is the ratio of unknown interests such



(a) Interests matrix

(b) Group interest distributions

Figure 5.2: True consumer interests with $\sigma_i^g = 10$

that $ratio_{ui} \in [0, 1]$. The result is having the matrix $\mathbf{INT}_{avail}$ with the fraction $ratio_{ui}$ of its values set to NaN; i.e. their true interest values are currently not available or alternatively "hidden".

After generating the matrix $\mathbf{INT}_{avail}$, we need to ensure that currently at least one service for each consumer is known to be liked. This is necessary given the fact that consumers would not participate in the service initially if they have absolutely no liked service at the beginning.

In order to do so, all n_s services of each of the n_c consumers are scanned to find the first service j for each consumer i which satisfies the following:

$$(\mathbf{INT}_{avail}[i, j] \; is \; NaN) \wedge (\mathbf{INT}_{true}[i, j] = 1)$$

whenever the following applies:

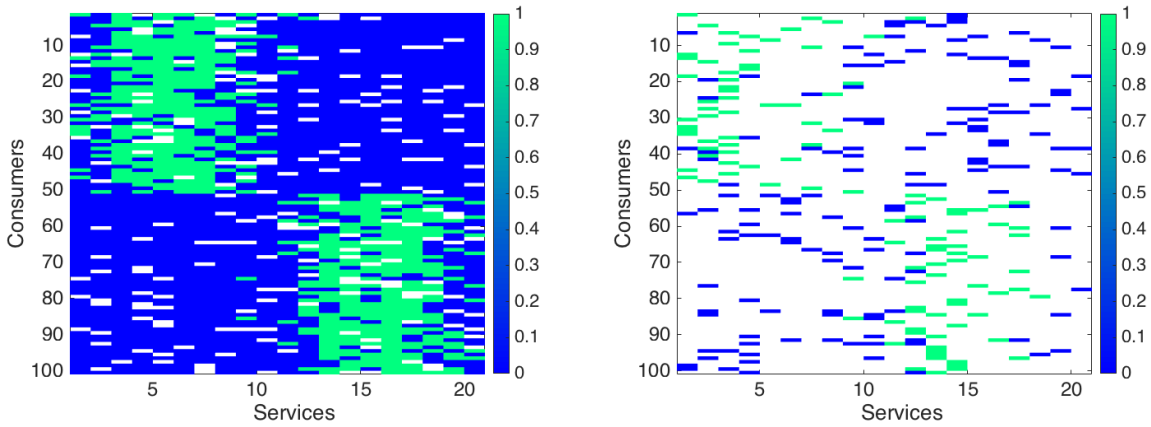
$$\forall k : \mathbf{INT}_{avail}[i, k] \; is \; not \; NaN, \mathbf{INT}_{avail}[i, k] = 0$$

This means that the true interest in service j by consumer i is currently unknown while being actually positive (i.e. $\mathbf{INT}_{true}[i, j] = 1$). This also means that consumer i has no interest in any service k as of now. When these two conditions are met, the true interest of consumer i in service j is revealed as follows:

$$\mathbf{INT}_{avail}[i, j] = \mathbf{INT}_{true}[i, j]$$

This is followed by moving to the next consumer. This way, one service of interest for each consumer is at least generated as of the first time step.

Figures 5.3a and 5.3b show the produced available consumer interests matrices $\mathbf{INT}_{avail}$ under $ratio_{ui} = 0.1$ and $ratio_{ui} = 0.9$, respectively. In both figures, the standard deviation of the consumer interests distribution is given by $\sigma_i^g = 2$. Figures 5.3c and 5.3d also show the available consumer interests matrices $\mathbf{INT}_{avail}$ under $ratio_{ui} = 0.1$ and $ratio_{ui} = 0.9$, respectively. However, the



(a) $\sigma_i^g = 2$ and $ratio_{ui} = 0.1$

(b) $\sigma_i^g = 2$ and $ratio_{ui} = 0.9$

(c) $\sigma_i^g = 10$ and $ratio_{ui} = 0.1$

(d) $\sigma_i^g = 10$ and $ratio_{ui} = 0.9$

Figure 5.3: Different available consumer interest scenarios

standard deviation of the consumer interests distribution is given by $\sigma_i^g = 10$ for both figures this time.

Refer to Algorithms B.1 to B.4 in Appendix B for further details about this operation of consumer interest profiles synthesis.

5.3 Recommender Designs

In order to design the proposed content recommender, a gradual approach is followed starting with the most basic recommender and then enhancing it gradually. This approach proceeds with the following four recommender categories with the first being the most basic:

- Non–interactive, non-collaborative and non-group-based category,
- Interactive, non-collaborative and non-group-based category,

- Interactive, collaborative and non-group-based category, and
- Interactive, collaborative and group-based category.

The mathematical formulations of each recommender within each one these categories are discussed in this section. Further details about the recommenders discussed and their functions can be found in Algorithms B.5 to B.19 in Appendix B at the end of the thesis. In the next section, these recommenders are experimented under the different consumer and network scenarios in order to choose one for the proposed system.

5.3.1 Category 1 Recommender

In this category, one recommender is discussed representing the non-interactive, non-collaborative and non-group-based recommender. It is non-interactive because there are no interactions between it and the consumers in terms of receiving their feedback on the set of distributed services. It is non-collaborative since there are no collaborations between the consumers based on their interest-profile similarities when making service recommendations. It is also non-group-based since all consumers receive the same set of distributed services with no differentiations between them based on the geographical locations of their groups.

From $t = 1$ until the end of the experiment time $n_{T_{rcmnd}}$, the popularity $\mathbf{pop}^{avail}_{services}[s]$ is computed for each service s of the n_s services by summing over all the available consumer interests in that service as follows:

$$\mathbf{pop}^{avail}_{services}[s] = \sum_{i:\mathbf{INT}_{avail}[i,s] \text{ is not } NaN} \mathbf{INT}_{avail}[i,s]$$

where i is the consumer index. Based on the resulting vector $\mathbf{pop}^{avail}_{services}$, service popularities are sorted in a descending order using the function $(sort)$ to get the vector of the most popular service indices $\mathbf{id}_{mps}$ as follows:

$$[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{avail}_{services}, -1)$$

Then, the first cap_{net} services are chosen from $\mathbf{id}_{mps}$ to be distributed as follows:

$$\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$$

where $\mathbf{id}_{ds}$ is the vector of distributed service indices.

5.3.2 Category 2 Recommenders

In what follows, the set of category 2 interactive recommenders are introduced. These recommenders differ in their way of interacting with the consumers while being all non-collaborative and non-group-based.

Starting with the Greedy recommender; which is identical to the previous category 1 recommender except for the consumer interactions taking place. These interactions represent simply the fact that all n_c consumers reveal their true interests in feedback to the services being distributed in each time step t as follows:

$$\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$$

where i is the consumer index and $\mathbf{id}_{ds}$ is the vector of distributed service indices.

In the ϵ-greedy recommender, the service popularities vector $\mathbf{pop}^{avail}_{services}$ is computed and sorted in order to produce the vector of the most popular service indices $\mathbf{id}_{mps}$. However, the distributed services are not all chosen greedily this time. Alternatively, only the first $\lfloor cap_{net} \times (1-\epsilon) \rfloor$ most popular service indices are chosen greedily as follows:

$$\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1-\epsilon) \rfloor]$$

where cap_{net} is the network capacity and $\epsilon = ratio_{ui}$ is the probability of random exploration. The remaining $(cap_{net} - \lfloor cap_{net} \times (1-\epsilon) \rfloor)$ service indices are chosen at random within the vector $\mathbf{id}_{mps}$ between the values $(\lfloor cap_{net} \times (1-\epsilon) \rfloor + 1)$ and n_s using the function $(randi)$. The vector of these remaining service indices is then concatenated to the vector $\mathbf{id}_{ds}$ as follows:

$$\mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds} ... \\ \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1-\epsilon) \rfloor + 1, n_s), 1, cap_{net} - \lfloor cap_{net} \times (1-\epsilon) \rfloor))]$$

This final set of services represented by $\mathbf{id}_{ds}$ is ultimately distributed after which all n_c consumers send their feedback by revealing their true interests towards these services as follows:

$$\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$$

where i is the consumer index.

Proceeding with the Decaying ϵ-greedy recommender which is identical to the ϵ-greedy recommender except for the fact that the probability of random exploration ϵ is decaying over time. The value of ϵ is decided at each time step t as follows:

$$\epsilon = 1 - length(\underset{(i,s)}{argfind}(\mathbf{INT}_{avail}[i, s] \ is \ not \ NaN))/(n_c \times n_s)$$

where $\mathbf{INT}_{avail}[i, s]$ is the available interest of consumer i in service s and the function $(argfind)$ is used to find all pair of indices (i, s) in which the interest of consumer i given service s is currently known. The length of indices produced by the function $(argfind)$ is then normalized by the number of all n_s service interests of all n_c consumers. This normalized ratio represents the fraction of interests known as of the current time step t. The intuition here is to have ϵ that decays as more true consumer interests are revealed. This means that less exploration would take place as the recommender receives more feedback from the consumers about their true interests and time progresses.

Finally, the UPB recommender starts by computing the vector of service popularities $\mathbf{pop}^{avail}_{services}$ for all n_s services by summing over all the available consumer interests for each service s like before. Based on $\mathbf{pop}^{avail}_{services}$, UPBs are computed for all n_s services as follows:

$$\mathbf{u}_{avail}[s] = \mathbf{pop}^{avail}_{services}[s] + length(\underset{i}{argfind}(\mathbf{INT}_{avail}[i, s] \ is \ NaN))$$

where $\mathbf{u}_{avail}$ is the vector of UPBs for all n_s services given the available consumer interests and the function ($argfind$) is used to find any consumer index i such that $\mathbf{INT}_{avail}[i, s]$ is not known given the service index s. The length of the vector resulting from this ($argfind$) function represents the maximum number of consumers which can like service s in addition to the consumers already known to like service s. Therefore, adding this length to the popularity $\mathbf{pop}^{avail}_{services}[s]$ of a certain service s results in the maximum popularity that service s can currently have which is $\mathbf{u}_{avail}[s]$. Based on $\mathbf{u}_{avail}$, the UPB recommender proceeds by first sorting the services in a descending order in terms of their UPBs using the function ($sort$) as follows:

$$[\sim, \mathbf{id}_{mps}] = sort(\mathbf{u}_{avail}, -1)$$

The resulting vector of most popular service indices $\mathbf{id}_{mps}$ is then used to choose the distributed service indices $\mathbf{id}_{ds}$ using the following formula:

$$\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$$

which means choosing the first cap_{net} services, estimated to be the most popular, for distribution followed by the consumer interactions.

5.3.3 Category 3 Recommenders

Before introducing category 3 recommenders, which are all interactive, collaborative but non-group-based, the function ($generateNGBRecommendations$) is introduced. This function is used to incorporate consumer collaborations within the aforementioned recommenders by generating the non-group-based interest recommendations based on the similarities between the interest-profiles of the different consumers. This function does not, however, take into account the different geographical locations of the different consumer groups which means that interest similarities are measured between all consumers as if they are within the same geographical location or group.

This function, has n_c, n_s and $\mathbf{INT}_{avail}$ as inputs and $\mathbf{INT}_{rcmnd}$ as the output where $\mathbf{INT}_{rcmnd}$ is the matrix of consumer interests after making the recommendations. Notice that $\mathbf{INT}_{rcmnd}[i, s]$ is the value of $\mathbf{INT}_{rcmnd}$ for consumer i given service s such as $\mathbf{INT}_{rcmnd}[i, s] = 1$ if service s is either known to be liked by consumer i or recommended to consumer i and $\mathbf{INT}_{rcmnd}[i, s] = 0$, otherwise.

This function starts with setting the Jaccard similarities matrix $\mathbf{J}$ to the zero matrix as follows:

$$\mathbf{J} = \mathbf{0}^{n_c \times n_c}$$

where n_c is the total number of consumers and $\mathbf{J}[i, j]$ is the Jaccard similarity between consumers i and j such as $\mathbf{J}[i, j] = 1$ if consumers i and j have identical service interest profiles and $\mathbf{J}[i, j] = 0$ if they have completely different service interest profiles.

After that, the Jaccard similarity $\mathbf{J}[i, j]$ is measured between all pairs of consumers i and j as follows:

$$\mathbf{J}[i,j] = length(\underset{s}{argfind}(\mathbf{INT}_{avail}[i,s] = \mathbf{INT}_{avail}[j,s]))/n_s$$

where $i \neq j$ and the function ($argfind$) is used to identify any service index s such as $\mathbf{INT}_{avail}[i,s] = \mathbf{INT}_{avail}[j,s]$. The length of the resulting service indices vector measures the number of services in which consumers i and j have currently similar interests. When this length is normalized by the total number of services n_s, we get $\mathbf{J}[i,j] \in [0,1]$.

Given $\mathbf{J}$ and starting with $\mathbf{INT}_{rcmnd} = \mathbf{INT}_{avail}$, $\mathbf{INT}_{rcmnd}$ is recomputed for all n_c consumers whenever the interest of consumer i in service s is not known (i.e. $\mathbf{INT}_{rcmnd}[i,s]$ or alternatively $\mathbf{INT}_{avail}[i,s]$ is NaN). This is done as follows:

$$\mathbf{INT}_{rcmnd}[i,s] \leftarrow \mathbf{INT}_{rcmnd}[\underset{j}{argmax}(\mathbf{J}[i,j]), s]$$

where the function ($argmax$) is used here to identify the consumer j with the maximum Jaccard similarity compared to consumer i. The idea here is to recommend the same interest in service s of consumer j to consumer i based on the fact that consumer j has the most similar interest profile compared to consumer i. Notice that $\mathbf{J}[i,j]$ has already been enforced to equal 0 whenever $i = j$ in order to avoid distorting the above $\mathbf{INT}_{rcmnd}$ computation with meaningless same-consumer recommendations given that $\mathbf{J}[i,j] = 1$ whenever $i = j$.

In what follows, category 3 recommenders are introduced. Notice that all of these recommenders are interactive and collaborative while being non-group-based.

Starting with the Greedy recommender which is very similar to the Greedy recommender of category 2 except for the fact that recommendation collaborations are incorporated by using the matrix $\mathbf{INT}_{rcmnd}$ instead of $\mathbf{INT}_{avail}$ after being evaluated as follows:

$$\mathbf{INT}_{rcmnd} = generateNGBRecommendations(n_c, n_s, \mathbf{INT}_{avail})$$

This is followed by using $\mathbf{INT}_{rcmnd}$ to compute the vector of service popularities $\mathbf{pop}^{rcmnd}_{services}$ based on $\mathbf{INT}_{rcmnd}$ for all n_s services as follows:

$$\mathbf{pop}^{rcmnd}_{services}[s] = \sum_{i:\mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i,s]$$

where s is the service index. Given $\mathbf{pop}^{rcmnd}_{services}$, the vector of most popular service indices $\mathbf{id}_{mps}$ is computed as follows:

$$[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{rcmnd}_{services}, -1)$$

after which the first cap_{net} most popular service indices are chosen greedily to be distributed as follows:

$$\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$$

Consumer interactions, in response to the services presented by $\mathbf{id}_{ds}$, are done similar to before.

Both ϵ-greedy and Decaying ϵ-greedy recommenders are very similar again to their previous counterparts (category 2 ϵ-greedy and Decaying ϵ-greedy recommenders). The only difference is the fact that $\mathbf{INT}_{rcmnd}$ is used instead of $\mathbf{INT}_{avail}$ when computing service popularities $\mathbf{pop}^{rcmnd}_{services}$ where $\mathbf{INT}_{rcmnd}$ is evaluated as shown before in the Greedy recommender. $\mathbf{pop}^{rcmnd}_{services}$ is, in turn, used to compute $\mathbf{id}_{mps}$. The ϵ computation, the $\mathbf{id}_{ds}$ computation and the consumer interactions are all identical to how they have been done before in category 2 ϵ-greedy and Decaying ϵ-greedy recommenders.

Finally, the UPB recommender is also similar to its non-collaborative counterpart (category 2 UPB recommender). One difference is the fact that $\mathbf{INT}_{rcmnd}$ is used instead of $\mathbf{INT}_{avail}$ when computing service popularities $\mathbf{pop}^{rcmnd}_{services}$ where $\mathbf{INT}_{rcmnd}$ is evaluated as shown before in the Greedy recommender. After evaluating $\mathbf{pop}^{rcmnd}_{services}$, $\mathbf{INT}_{rcmnd}$ is also used in computing $\mathbf{u}_{rcmnd}$ using the formula:

$$\mathbf{u}_{rcmnd}[s] = \mathbf{pop}^{rcmnd}_{services}[s] + length(\underset{i}{argfind}(\mathbf{INT}_{rcmnd}[i, s]\ is\ NaN))$$

where s is the service index and $\mathbf{u}_{rcmnd}$ is the vector of service UPBs based on using $\mathbf{INT}_{rcmnd}$. The function $(argfind)$ is used to find the vector of any consumer index i such that $\mathbf{INT}_{rcmnd}[i, s]$ is not known. The length of this vector represents the maximum number of consumers with unknown interests and recommendations who might like the service s. Adding this length to $\mathbf{pop}^{rcmnd}_{services}[s]$ gives the UPB estimate $\mathbf{u}_{rcmnd}[s]$ for service s.

Given $\mathbf{u}_{rcmnd}$, the vector of distributed service indices $\mathbf{id}_{ds}$ is computed as the indices of the first cap_{net} most popular services with the highest $\mathbf{u}_{rcmnd}$ values as follows:

$$[\sim, \mathbf{id}_{mps}] = sort(\mathbf{u}_{rcmnd}, -1)$$

$$\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$$

where $\mathbf{id}_{mps}$ is the vector of the most popular service indices. The consumer interactions are done similar to before.

5.3.4 Category 4 Recommenders

Before proceeding with category 4 recommenders, which are all interactive, collaborative and group-based, the function $(generateGBRecommendations)$ is introduced. Similar to the function $(generateNGBRecommendations)$, this function is also used to incorporate consumer collaborations within the aforementioned recommenders based on the similarities between their interest profiles. However, it does take into account the different geographical locations of the different consumer groups. Therefore, this function goes through the same exact steps of the function $(generateNGBRecommendations)$ except for the fact that the matrix $\mathbf{J}$ of consumer similarities is measured between only those consumers of the same group within the same geographical location. These consumers have the index id^f_c for the first group consumer and the index id^l_c for the last group consumer

which means that these two extra inputs are needed for the function (*generateGBRecommendations*).

In addition, the $\mathbf{INT}_{rcmnd}$ computation in the function (*generateGBRecommendations*) is for the consumer of index $j : j \in \{id_c^f : id_c^l\}$ in order to make sure that this consumer is chosen to be within the same geographical group such that s/he has the most similar interest profile compared to consumer i.

Starting with the Greedy recommender which is very similar to the Greedy recommender of category 3. The main difference is its ability to make interest recommendations based on the geographical group of consumers. In each group 1 to n_g, the recommender starts by identifying both id_c^f and id_c^l of the current group g as follows:

$$id_c^f = n_c/n_g \times (g-1) + 1$$

$$id_c^l = n_c/n_g \times g$$

These two indices are used as additional inputs to the function (*generateGBRecommendations*) which in turn outputs the interests matrix $\mathbf{INT}_{rcmnd}$. Based on $\mathbf{INT}_{rcmnd}$, service popularities given by the vector $\mathbf{pop}_{services}^{rcmnd}$ are computed for all n_s services by summing over all the known consumer $\mathbf{INT}_{rcmnd}$ interests given the same service. However, these interests have to be for consumer indices between id_c^f and id_c^l in order to account for only those consumers within the same group. Therefore, $\mathbf{pop}_{services}^{rcmnd}$ is computed as follows:

$$\mathbf{pop}_{services}^{rcmnd}[s] = \sum_{i \in \{id_c^f : id_c^l\} : \mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i,s]$$

where s is the service index. Based on $\mathbf{pop}_{services}^{rcmnd}$, the first cap_{net} highest popularity services are chosen greedily for distribution. Consumers, within the group, interact with these distributed services by revealing their true interests like before.

The ϵ-greedy recommender is again similar to its non-group-based counterpart (category 3 ϵ-greedy recommender). The main difference is its ability to recognize consumer groups 1 to n_g. In each group, both id_c^f and id_c^l are computed and used as inputs to the function (*generateGBRecommendations*) which in turn outputs $\mathbf{INT}_{rcmnd}$. After that, $\mathbf{INT}_{rcmnd}$ is used to compute $\mathbf{pop}_{services}^{rcmnd}$ which considers all consumers within the same group. With $\mathbf{pop}_{services}^{rcmnd}$, the vector of the most popular service indices $\mathbf{id}_{mps}$ is computed using the function (*sort*). However, only the first $\lfloor cap_{net} \times (1-\epsilon) \rfloor$ most popular services are chosen greedily for distribution where $\epsilon = ratio_{ui}$. The remaining $(cap_{net} - \lfloor cap_{net} \times (1-\epsilon) \rfloor)$ services are chosen randomly for exploration purposes. Consumer interact with the distributed services by revealing their true interests as usual.

The Decaying ϵ-greedy recommender is also similar to its non-group-based counterpart (category 3 Decaying ϵ-greedy recommender) except for its ability to consider consumer groups. It goes over consumer groups 1 to n_g. In each group, it

starts by computing both id_c^f and id_c^l for the current group g. Based on these two indices, $\mathbf{INT}_{rcmnd}$ is computed and used afterwards to compute the vector $\mathbf{pop}_{services}^{rcmnd}$ while considering those consumers within the same group only. The vector of most popular service indices $\mathbf{id}_{mps}$ is computed using $\mathbf{pop}_{services}^{rcmnd}$ for the group. After that, and based on how many interests are currently available, the value of ϵ is computed using the available interests of consumers within the group only using the formula:

$$\epsilon = 1 - length(\underset{(i \in \{id_c^f : id_c^l\}, s)}{argfind}\ (\mathbf{INT}_{avail}[i,s]\ is\ not\ NaN))/(n_c/n_g \times n_s)$$

where the function ($argfind$) is used to find the indices of all known interests of $\mathbf{INT}_{avail}$ such that the consumer indices of these interests are for group consumers only. The length of the resulting vector is then normalized by the number of group consumers n_c/n_g multiplied by the number of services n_s. This gives the fraction of known group interests which is in no need of further exploration.

The first $\lfloor cap_{net} \times (1-\epsilon) \rfloor$ services are then chosen greedily whereas the remaining services are chosen randomly to allow for exploration. Notice that the value ϵ decays over time given the fact that consumers reveal their true interests each time they interact with the distributed services.

Finally, the UPB recommender is similar to its non-group-based counterpart (category 3 UPB recommender). The main difference is again its ability to treat consumers differently depending on the geographical group they belong to. This applies for groups 1 to n_g where each group starts by identifying the consumer indices id_c^f and id_c^l of the current group g. Based on these indices, $\mathbf{INT}_{rcmnd}$ is computed using the function ($generateGBRecommendations$). Given $\mathbf{INT}_{rcmnd}$, the service popularities $\mathbf{pop}_{services}^{rcmnd}$ are computed as well as their UPBs $\mathbf{u}_{rcmnd}$ while considering only those consumers within the current group g. Based on $\mathbf{u}_{rcmnd}$, the first cap_{net} most popular services with the highest UPBs are chosen for distribution to the group consumers. Those consumers interact with the distributed services by revealing their true interests as usual.

5.4 Experiment Design

In order to decide which recommender to choose for the proposed content distribution system, a set of 12 experiments are conducted for all the recommenders discussed previously under different consumer interest and network capacity scenarios. Consumer interest scenarios differ in their group interest standard deviation σ_i^g values and the ratio of unknown consumer interests $ratio_{ui}$ values. Network capacity scenarios differ in their cap_{net} values. Table 5.1 summarizes these 12 experiments such that each experiment lasts for a total of $n_{T_{rcmnd}} = 100$ time intervals. Notice that these time intervals represent day periods which can be spread between different workdays but for the same period of the day.

For experiments 1 to 4, the standard deviation σ_i^g of the group interest distributions is varied while experimenting all combinations of $ratio_{ui}$ and cap_{net}

Experiment No.	σ_i^g	$ratio_{ui}$	cap_{net}
1	varied	low	low
2		low	high
3		high	low
4		high	high
5	low	varied	low
6	low		high
7	high		low
8	high		high
9	low	low	varied
10	low	high	
11	high	low	
12	high	high	

Table 5.1: Experiment Design

values. For experiments 5 to 8, the ratio of unknown consumer interests $ratio_{ui}$ is varied while experimenting all combinations of σ_i^g and cap_{net} values. For experiments 9 to 12, the network capacity cap_{net} is varied while experimenting all combinations of σ_i^g and $ratio_{ui}$ values. Each of σ_i^g, $ratio_{ui}$ and cap_{net} is varied between its low and high value. These values are 2 & 10 for σ_i^g, 0.1 & 0.9 for $ratio_{ui}$ and 2 & 10 for cap_{net}.

For all of these experiments, notice that the number of consumers n_c is set to 100 consumers, the number of services n_s to 20 services and the number of groups n_g to 2 groups. Also notice that Figures 5.1a to 5.3d illustrate the true as well as the available consumer interest matrices under the different σ_i^g and $ratio_{ui}$ values of concern in the experiments.

5.5 Experiment Results

The performance of the experimented recommenders is measured using the following two indicators:

- the interests ratio; which measures the ratio of revealed consumer interests. This indicator shows the degree of exploration made by the recommender under consideration, and
- the popular distributed services ratio; which measures the ratio between the number of distributed popular services and the number of all distributed services. This indicator shows the degree of exploitation made by the recommender under consideration.

These indicators are all used while highlighting the categories of the recommenders being experimented in the results. The experiment setup for each result is also highlighted using the tuple $(\sigma_i^g, ratio_{ui}, cap_{net})$. Notice that category 1 has only one recommender which represents the most basic recommender design.

5.5.1 Varying Group Interest Distributions (Experiments 1 to 4)

In experiments 1 to 4, the standard deviation of the group interest distributions σ_i^g is varied between 2 and 10 in each experiment. The unknown interests ratio $ratio_{ui}$ and the network capacity cap_{net} are both fixed in each experiment according to Table 5.1.

Starting with experiment 1 in which Figure 5.4 shows the interests ratio under the different recommenders to be high given the fact that the ratio of unknown interests $ratio_{ui}$ has already the low value of 0.1. Figure 5.5 of experiment 1 shows the popular distributed services ratio to be low under the recommenders of categories 1 to 3 given the low network capacity cap_{net} of 2. This is due to the fact that with such a low network capacity and consumer groups with different interest distributions, it is impossible to distribute the popular services at each group without being group-based as it is the case with category 4 recommenders. On the other hand, the performance of category 4 recommenders is much better with the UPB recommender being the best compared to its counterparts under both $\sigma_i^g = 2$ and $\sigma_i^g = 10$.

Notice that, in experiment 1, the performance of categories 2 and 3 recommenders under $\sigma_i^g = 10$ is worse than their performance under $\sigma_i^g = 2$. This is given the fact that with wider interest distributions and limited network capacity, it becomes even harder to catch and distribute the truly popular services at each group.

Similar to Figure 5.4 of experiment 1, the interests ratio are already high given that $ratio_{ui} = 0.1$ in Figure 5.6 of experiment 2. However and contrary to experiment 1, the network capacity cap_{net} has the much larger value of 10. This makes the performance of categories 2 and 3 recommenders under $\sigma_i^g = 10$ better than their performance under $\sigma_i^g = 2$ as shown in Figure 5.7. Having a high network capacity in addition to the wider interests distribution allows the recommenders to catch and distribute some of the common popular services between the groups even without being group-based. This translates into a slightly higher popular distributed services ratio.

Notice that in the group-based recommenders of category 4, the performance is even better in experiment 2 compared to its counterpart in experiment 1 given the high network capacity. Even the performance gap between the UPB recommender and the other recommenders is negligible under category 4 of this experiment.

Contrary to both experiments 1 and 2, Figure 5.8 of experiment 3 shows the gradual increase in the interests ratio for recommenders of categories 2 to 4. This is given the much higher unknown interest ratio $ratio_{ui}$ of 0.9. Notice that the recommendations made throughout category 3 recommenders have allowed for faster exploration of consumer interests and therefore faster exploitation as shown in Figure 5.9 of experiment 3. This holds true for interest distributions of both $\sigma_i^g = 2$ and $\sigma_i^g = 10$.

In the categories 2 to 4 of experiment 3, the UPB recommender is the fastest in terms of exploring the consumer interests. However, the category 4 UPB recommender does not explore as fast or as much under the narrow interests distribution of $\sigma_i^g = 2$. This is given the fact that this recommender has managed to decide early on which

services are the most popular within each consumer group with high confidence and therefore has stopped exploring. This is demonstrated by the high popular distributed services ratio of this recommender as shown by Figure 5.9 of experiment 3.

Notice that the greedy recommender in categories 2 to 4 of experiment 3 is not able to explore due to its greediness which has led it into getting stuck in distributing the wrong set of services as demonstrated by the extremely low popular distributed service ratios shown in Figure 5.9 of experiment 3.

In addition, Figure 5.9 shows that the ϵ-greedy recommender of categories 2 to 4 does also perform poorly in terms of the popular distributed services ratio. This is mainly due to its high ϵ value which is set to the high $ratio_{ui}$ value of 0.9. Having a high ϵ has led the recommender to excessive exploration without the proper exploitation that would otherwise distribute the right set of popular services.

In all recommenders of categories 2 to 4 in experiment 3, the UPB recommender is exploiting the most as demonstrated by the high ratios of popular distributed services. However, all recommenders of categories 2 to 3 perform worse compared to category 4 recommenders due to their inability to identify consumer groups and distribute the right set of popular services for each group separately as explained previously. Moreover, all recommenders of categories 2 to 3 perform worse under $\sigma_i^g = 10$ compared to how they perform under $\sigma_i^g = 2$. This is, again as explained previously, due to the fact that catching and distributing popular services under wider interest distributions and low network capacities is much more challenging.

Figure 5.10 of experiment 4 shows the big increase in terms of the interests ratio by all the recommenders of categories 2 to 4 given the high unknown interests ratio $ratio_{ui}$ of 0.9. It does also show the much faster rate of increase in terms of this ratio given the high network capacity cap_{net} of 10.

Similar to before, Figure 5.11 shows that UPB recommenders are all superior compared to the other recommenders in categories 2 to 4. It does also show that UPB recommenders of category 4 perform better due to their ability to group consumers.

When $\sigma_i^g = 10$, Figure 5.11 shows that category 2 and 3 recommenders of experiment 4 perform a bit better compared to how they perform under $\sigma_i^g = 2$. This is due to the same reason discussed before which is the high network capacity cap_{net} of 10 that allows some common popular services to be chosen under $\sigma_i^g = 10$. However, having $\sigma_i^g = 10$ leads the category 4 recommenders to struggle a bit more compared to their situation under $\sigma_i^g = 2$. This is due to the fact that it becomes harder to decide which services are truly popular under $\sigma_i^g = 10$ and therefore more interactions with the consumers are needed despite the high network capacity cap_{net} of 10.

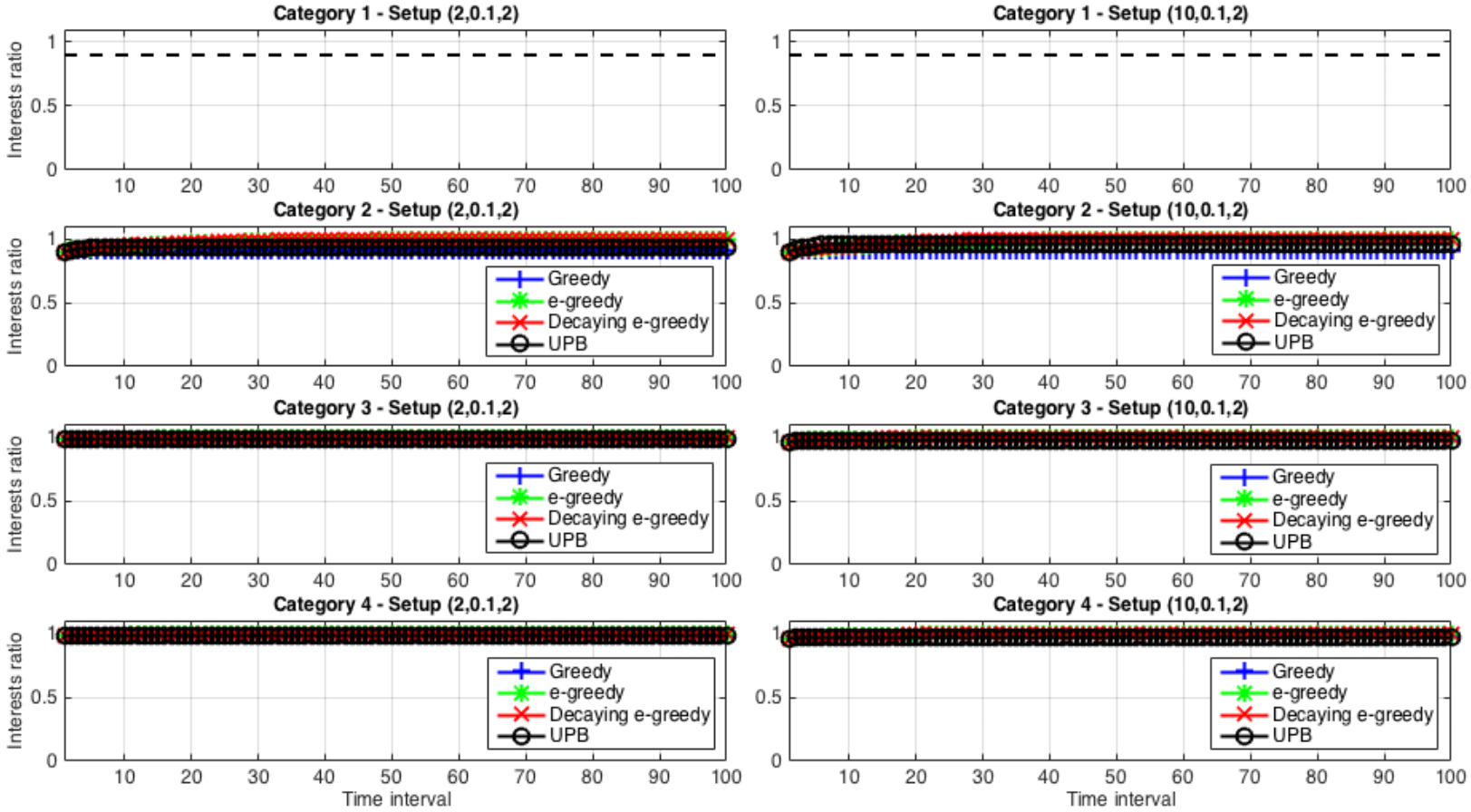


Figure 5.4: Experiment 1: effect on interests ratio

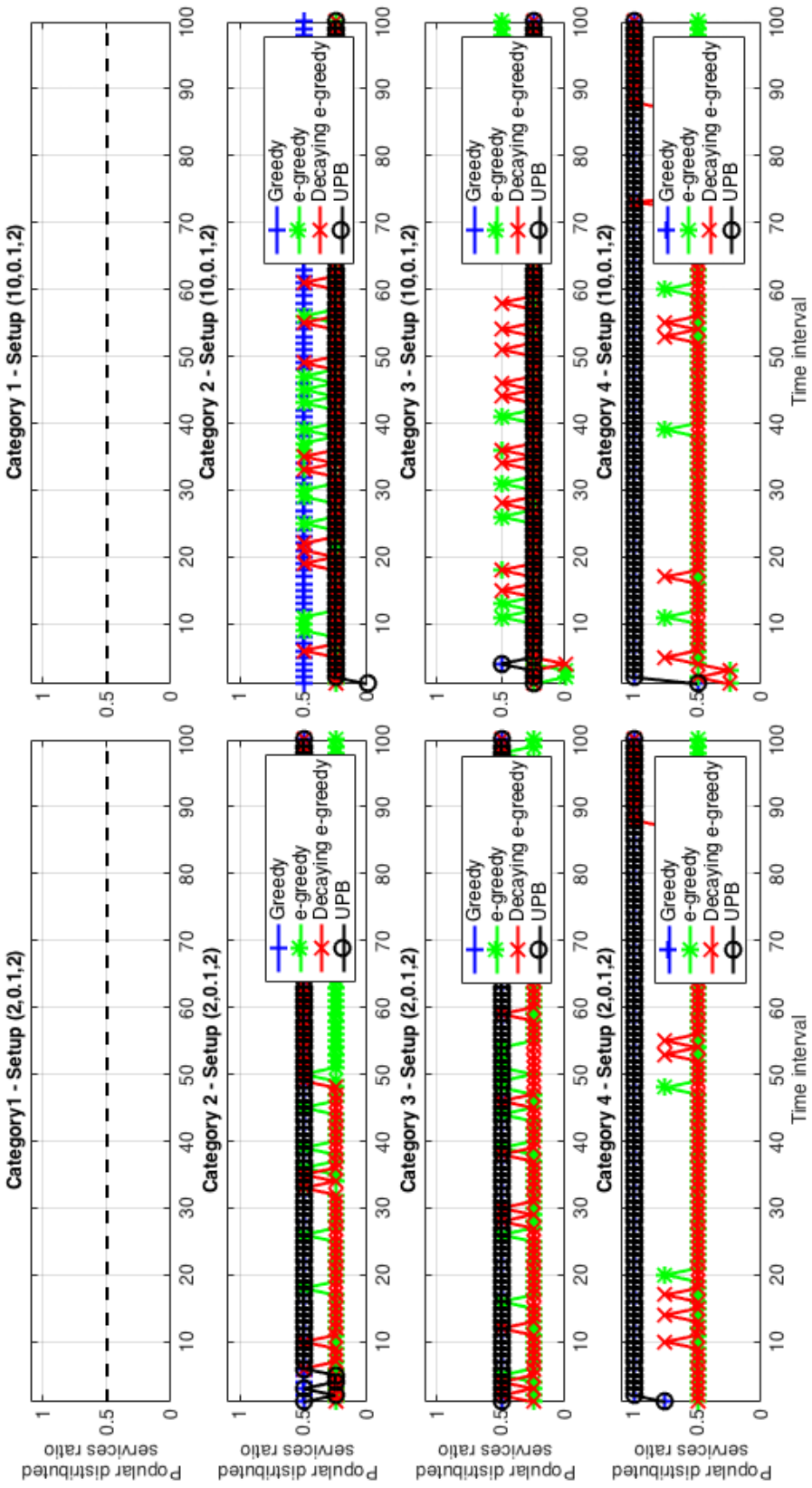


Figure 5.5: Experiment 1: effect on popular distributed services ratio

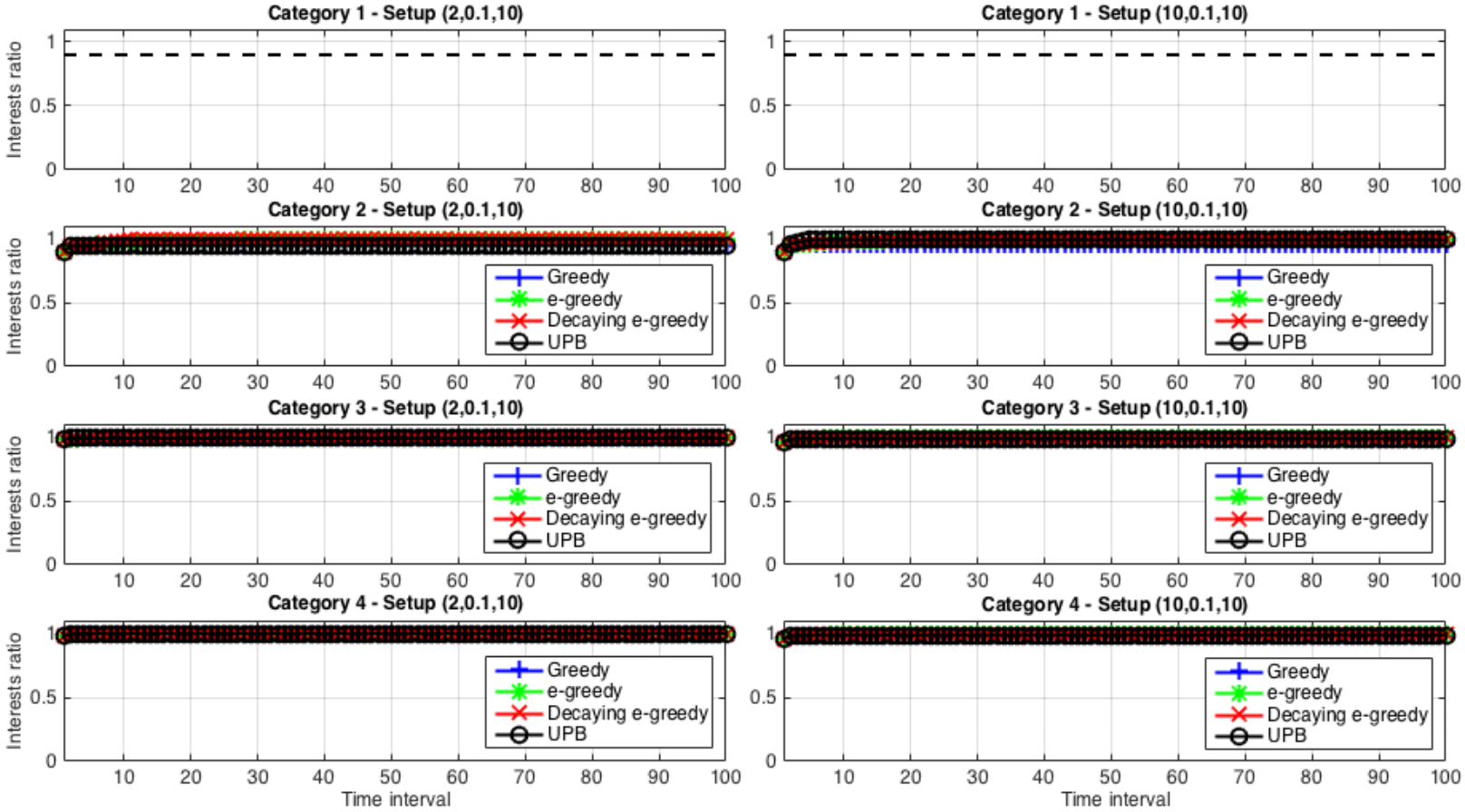


Figure 5.6: Experiment 2: effect on interests ratio

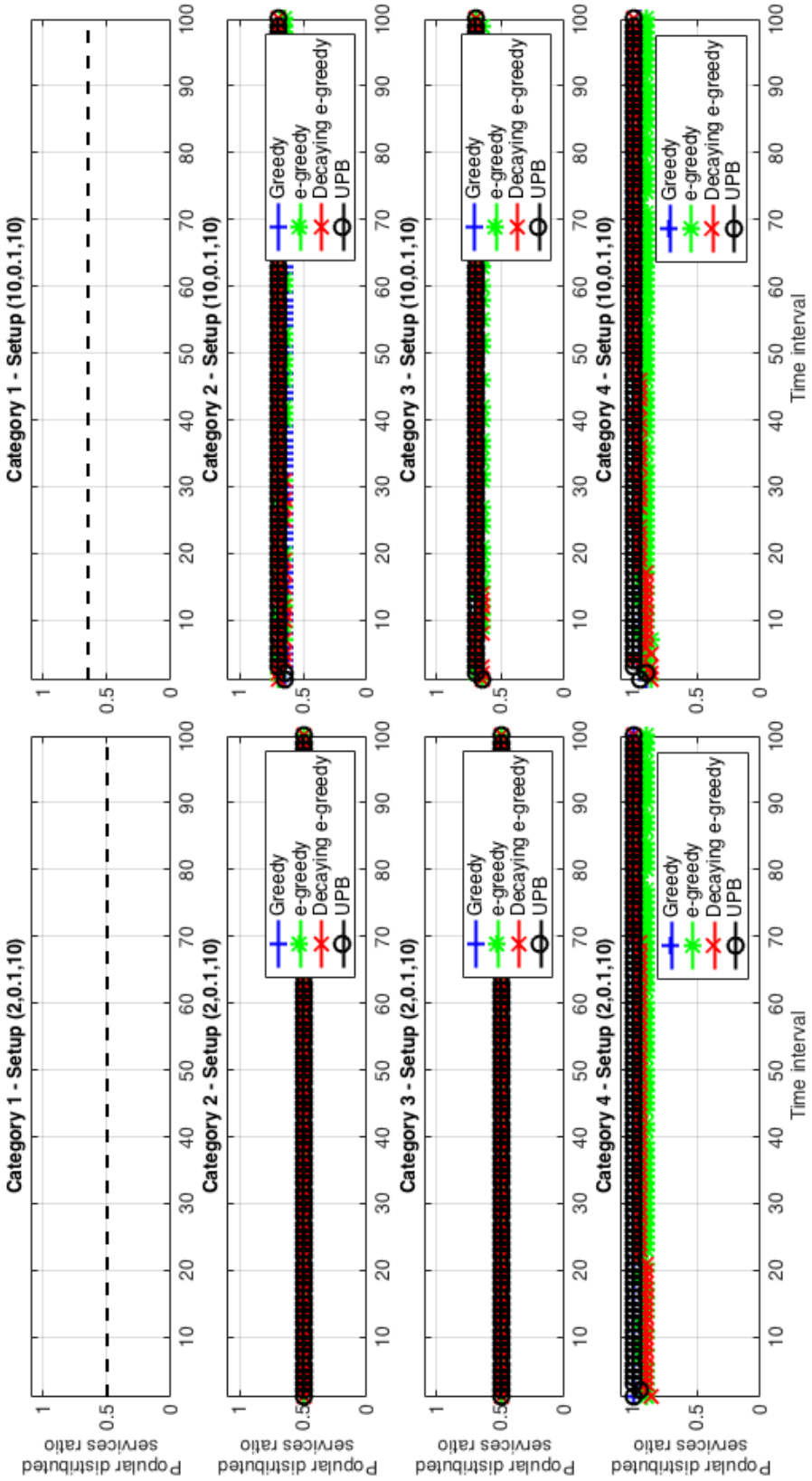


Figure 5.7: Experiment 2: effect on popular distributed services ratio

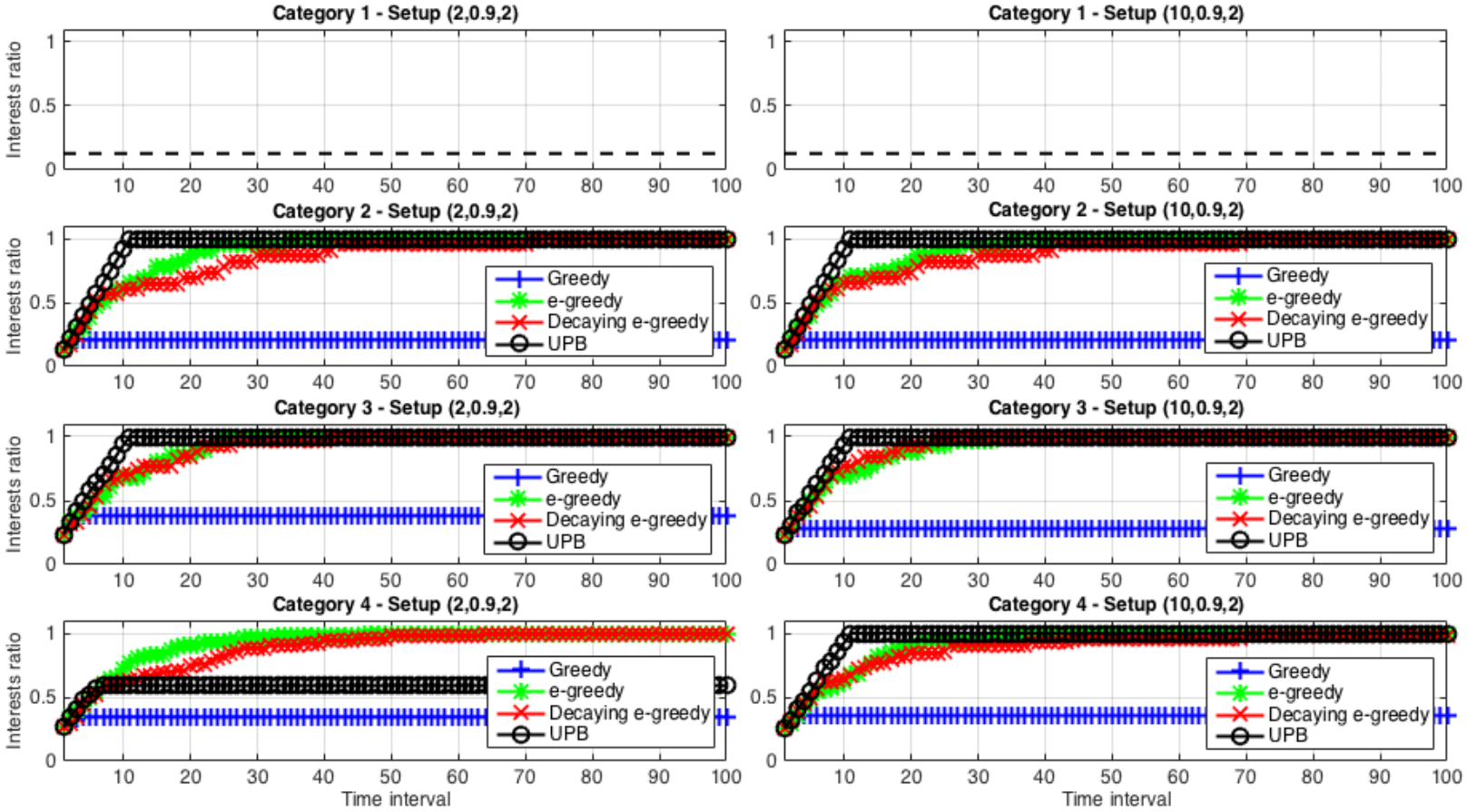


Figure 5.8: Experiment 3: effect on interests ratio

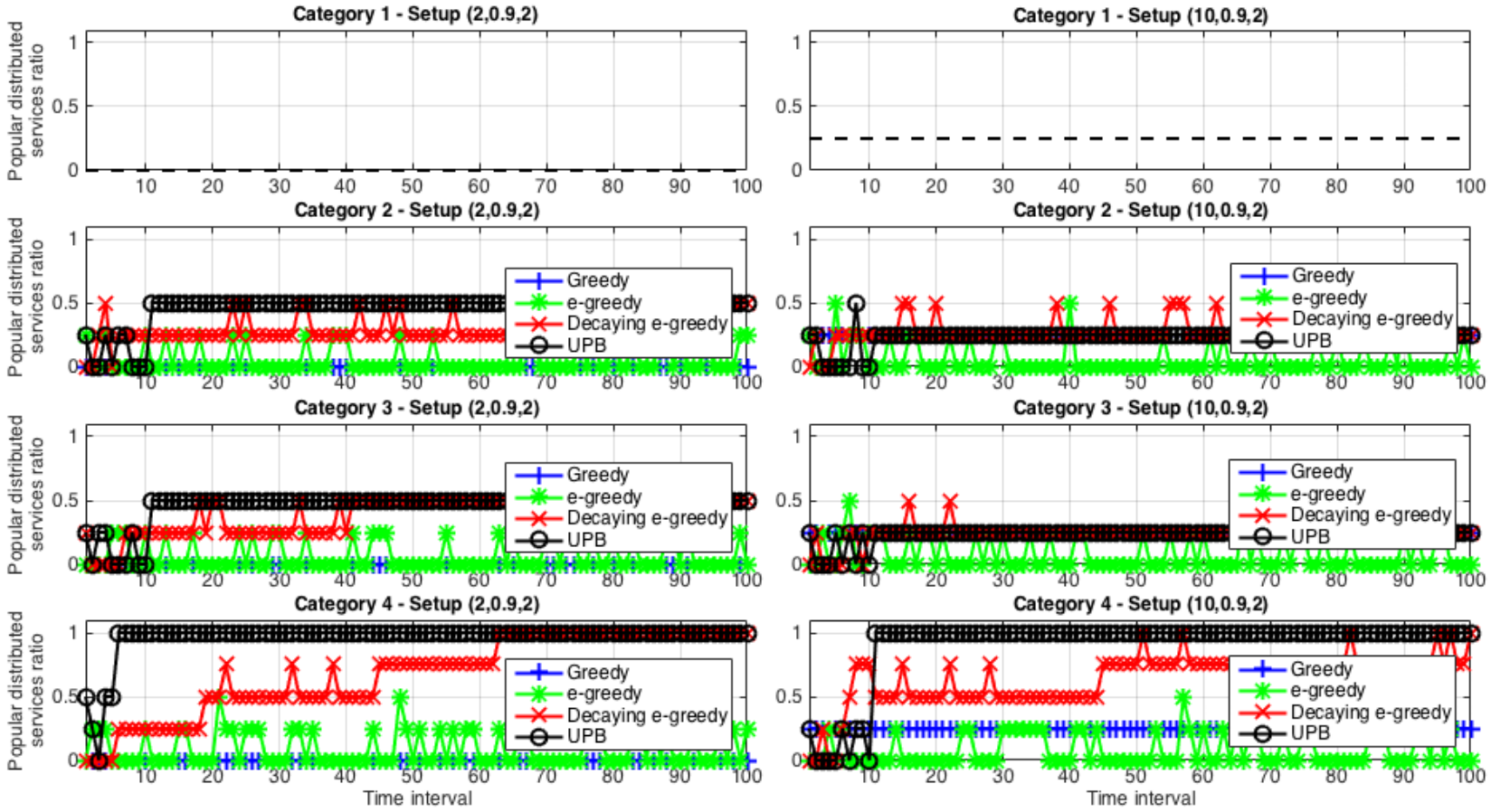


Figure 5.9: Experiment 3: effect on popular distributed services ratio

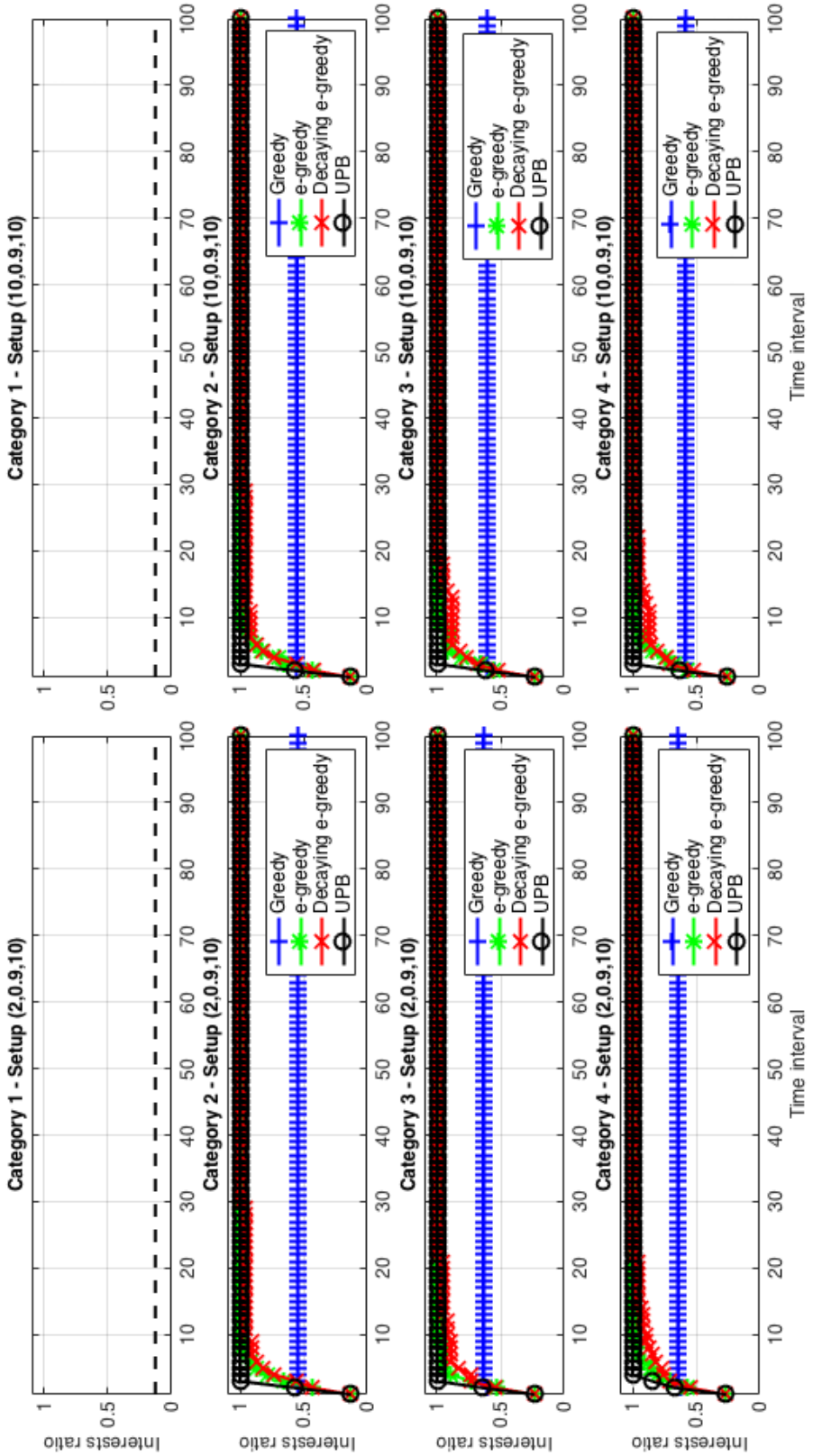


Figure 5.10: Experiment 4: effect on interests ratio

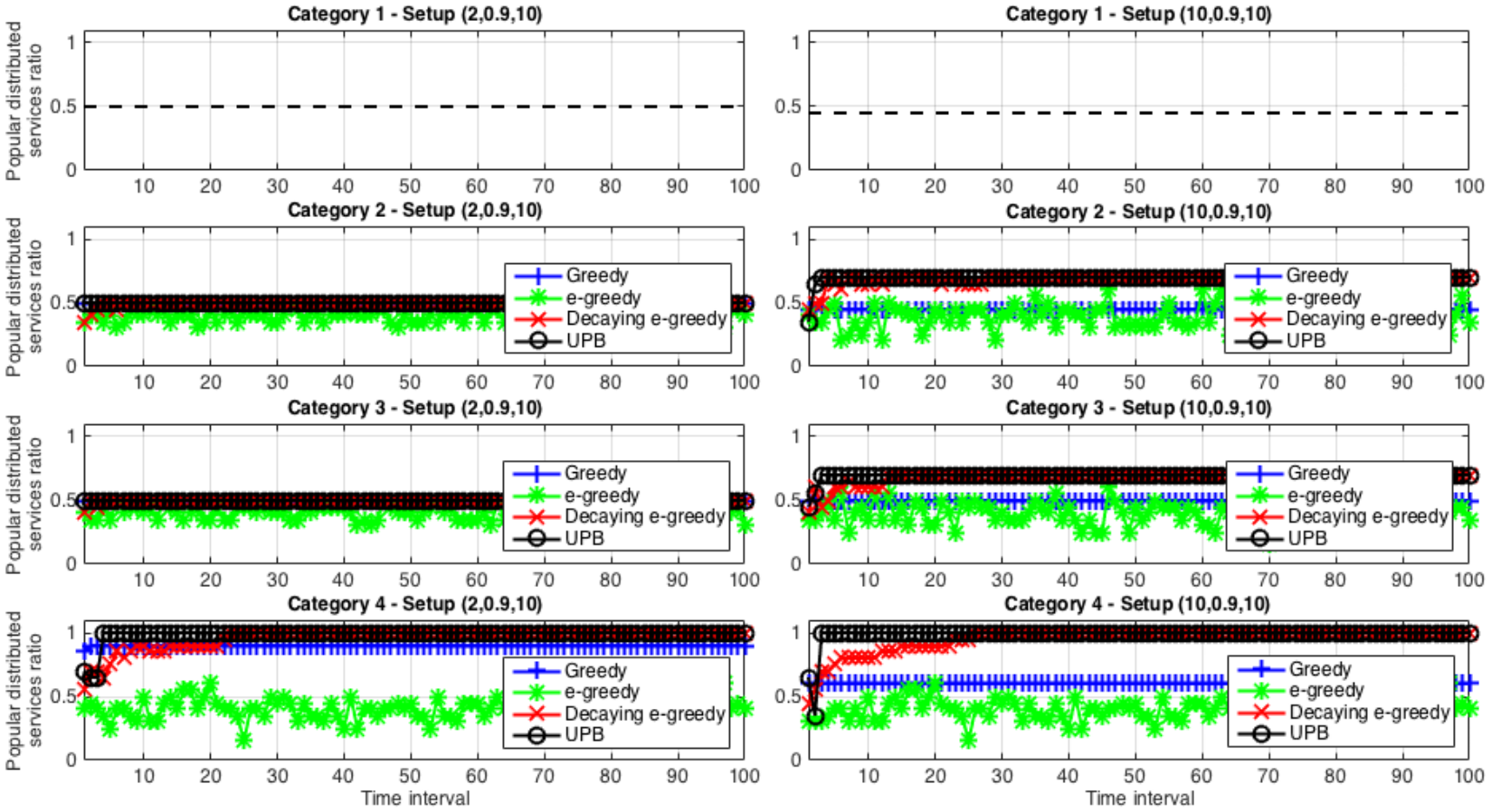


Figure 5.11: Experiment 4: effect on popular distributed services ratio

5.5.2 Varying Unknown-interest Ratios (Experiments 5 to 8)

In experiments 5 to 8, the unknown interests ratio $ratio_{ui}$ is varied between 0.1 and 0.9 in each experiment. The standard deviation of the group interest distributions σ_i^g and the network capacity cap_{net} are both fixed in each experiment according to Table 5.1.

Starting with experiment 5, Figure 5.12 shows the direct effect of higher $ratio_{ui}$ which is having lower interests ratio. Notice that the UPB recommender in categories 2 and 3 explores consumer interests the fastest. However, and as explained before, it does slow down in category 4 due to its ability to stop exploring and identify early the set of truly popular services as indicated by the high popular distributed services ratio shown in Figure 5.13.

As shown in Figure 5.13, increasing $ratio_{ui}$ results in lower ratios of popular distributed services. This is due to the fact that knowing the truly popular services becomes more challenging. In fact, the low network capacity cap_{net} of 2 makes this even worse and means that more consumer interactions are needed.

As before, the UPB recommender in categories 2 to 4 performs the best under all $ratio_{ui}$ values and group-based recommenders perform generally better. However, and under $ratio_{ui} = 0.9$, the greedy recommender gets stuck in the wrong set of services due to its lack of exploration and the ϵ-greedy explores excessively given that $\epsilon = ratio_{ui} = 0.9$.

Compared to Figure 5.12 of experiment 5, the interest ratios are higher and converge faster in Figure 5.14 of experiment 6 given the higher network capacity cap_{net} of 10. When $ratio_{ui} = 0.9$, recommenders need more time to converge and the greedy recommender does not explore enough due to its greediness.

Looking at the popular distributed service ratios shown in Figure 5.15 of experiment 6, we can see that all recommenders under all categories are performing similarly under both $ratio_{ui} = 0.1$ and $ratio_{ui} = 0.9$. This is due to the higher network capacity cap_{net} of 10. However, the ϵ-greedy recommender has a worse performance under $ratio_{ui} = 0.9$ given the excessively high value of ϵ which is set to equal $ratio_{ui}$.

Similar to Figure 5.12 of experiment 5, the interest ratios, shown in Figure 5.16 of experiment 7, are generally lower and slower to converge under $ratio_{ui} = 0.9$. However, the situation here is worse than in experiment 5 due to the wider interest distributions. Wider distributions in addition to the high unknown interests ratio $ratio_{ui}$ of 0.9 and the low network capacity cap_{net} of 2, make exploration more time consuming and requires more consumer interactions. Notice however that the UPB recommender in categories 2 to 4 still comes on top of all the other recommenders in terms of how fast it explores consumer interests. The greedy recommender on the other hand gets stuck as usual due to its greediness and lack of exploration which becomes even more evident under high $ratio_{ui}$ and low cap_{net}.

Figure 5.17 of experiment 7 shows the popular distributed services ratio to be lower under category 2 and 3 recommenders compared to the group-based category 4 recommenders independent from the value of $ratio_{ui}$. This is given the wider interest distributions and the low network capacity cap_{net} of 2. However, the performance gets

worse under $ratio_{ui} = 0.9$ compared to the case under $ratio_{ui} = 0.1$ due to the greater effort needed to explore consumer interests. Also notice that, as we have already seen, the greedy as well as the ϵ-greedy recommenders under $ratio_{ui} = 0.9$ perform poorly in general given the lack of exploration and excessive exploration, respectively. Relatively speaking, the UPB recommender performs the best compared to the other recommenders under categories 2 to 4.

The interest ratios shown in Figure 5.18 of experiment 8 are so similar to those shown in Figure 5.14 of experiment 6. There is only a slight increase in terms of the number of time intervals needed for the interests ratio to converge in the case of experiment 8 recommenders. This is mainly due to the wider interest distributions of $\sigma_i^g = 10$ and the higher unknown interests ratio $ratio_{ui}$ of 10 which make it harder to identify the true set of popular services.

Looking at the popular distributed services ratio shown in Figure 5.19 of experiment 8, we can see that the performance has actually reached a higher ratio compared to that shown in Figure 5.15 of experiment 6 under $ratio_{ui} = 0.9$. This is mainly due to the fact that under wider distributions of $\sigma_i^g = 10$, it becomes possible to have a set of common popular services which are identified and distributed by the non-group based recommenders under categories 2 and 3. However, it takes more time to make such a convergence due to the higher unknown interests ratio $ratio_{ui}$ of 0.9 in addition to the wider interest distribution of $\sigma_i^g = 10$. Notice that, as we have seen before, the greedy as well as the ϵ-greedy recommenders still have a worse performance due to their greediness and excessive exploration, respectively.

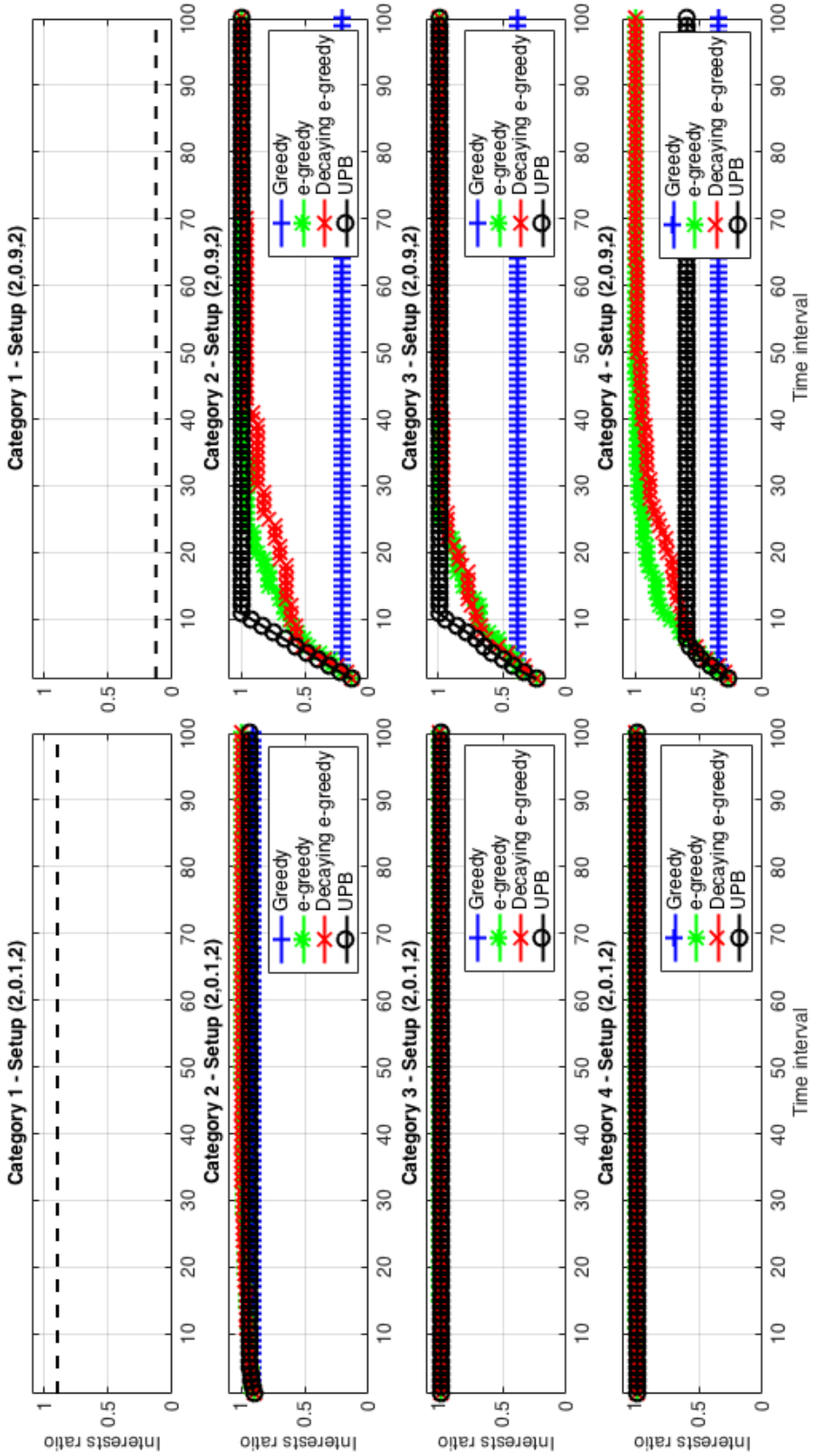


Figure 5.12: Experiment 5: effect on interests ratio

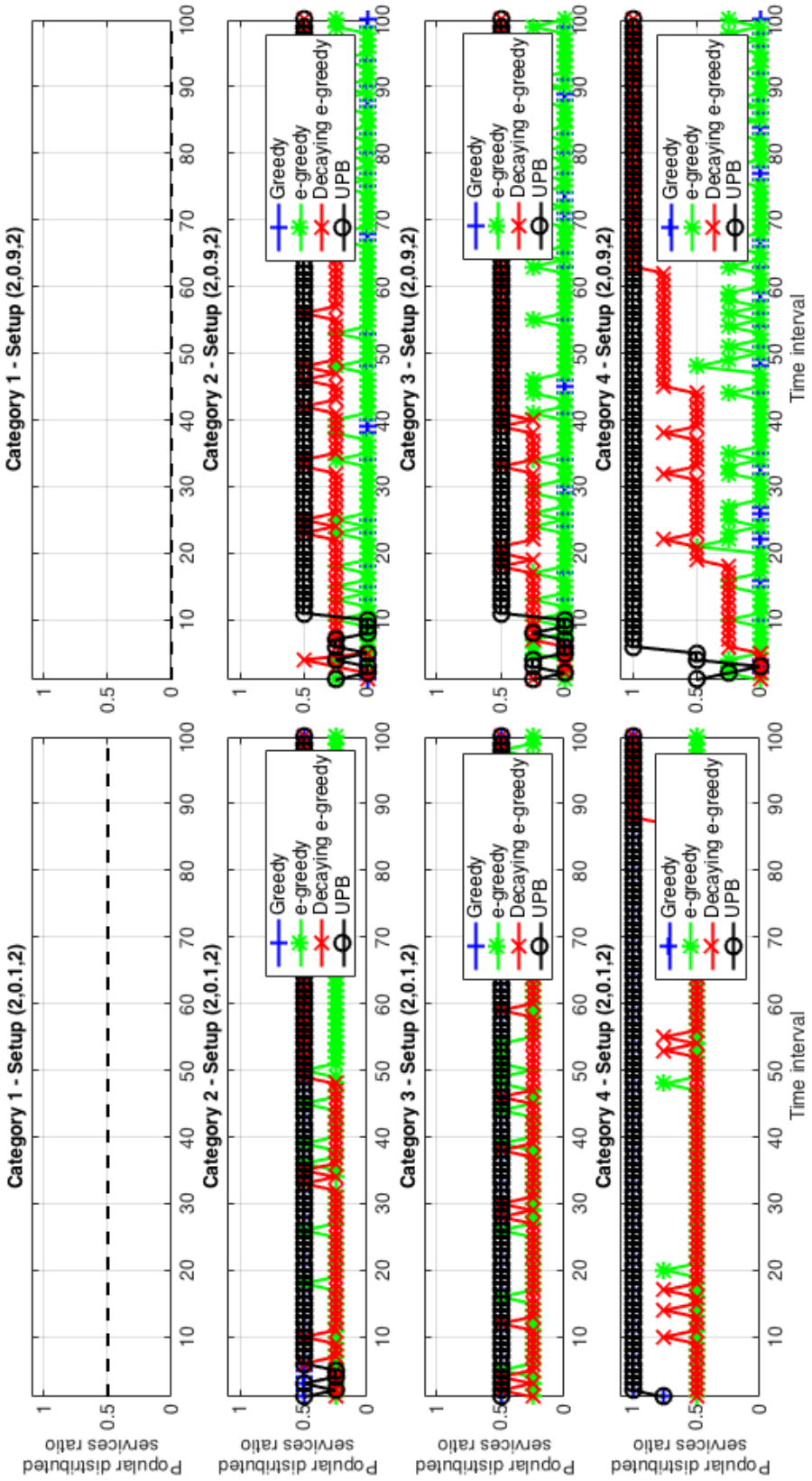


Figure 5.13: Experiment 5: effect on popular distributed services ratio

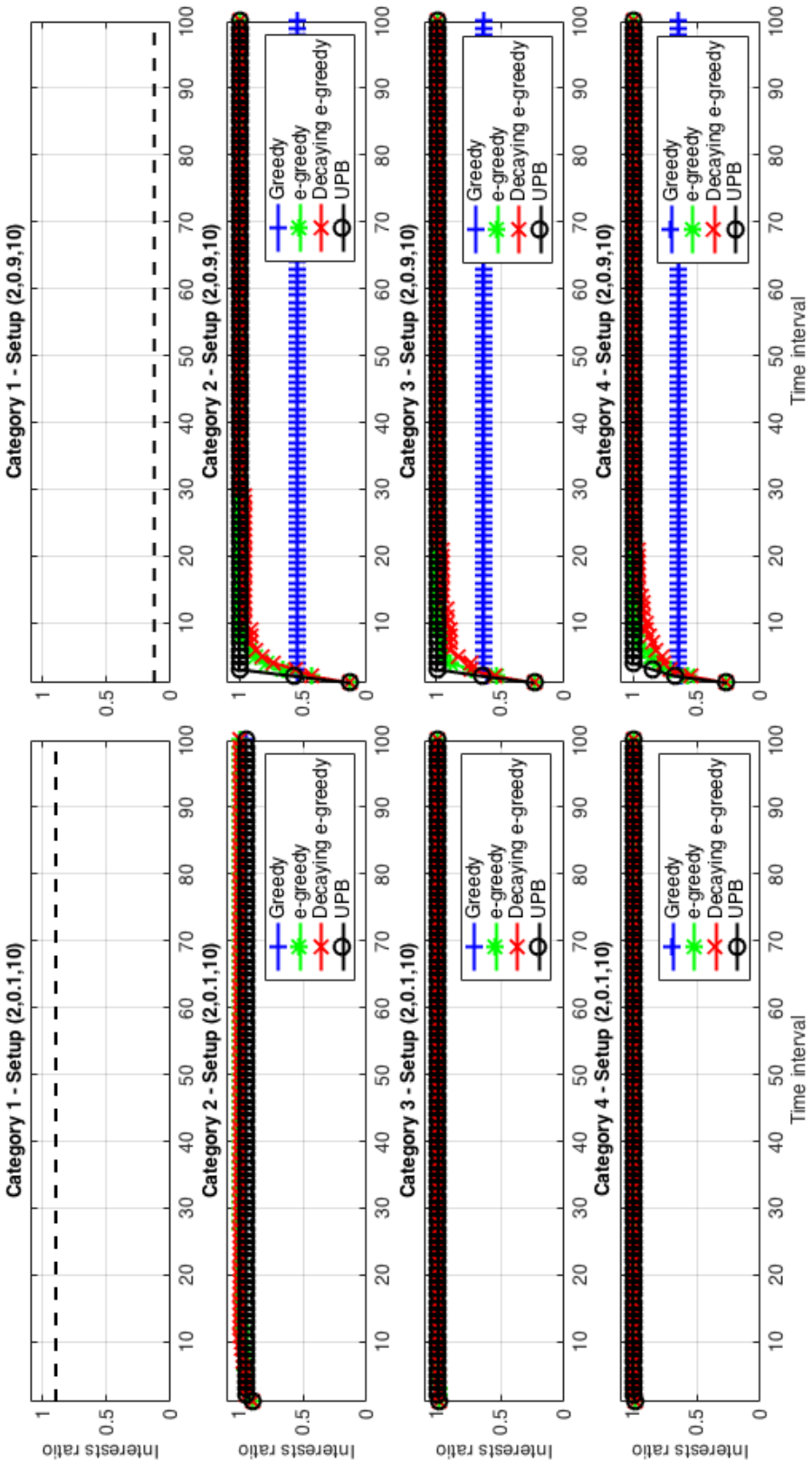


Figure 5.14: Experiment 6: effect on interests ratio

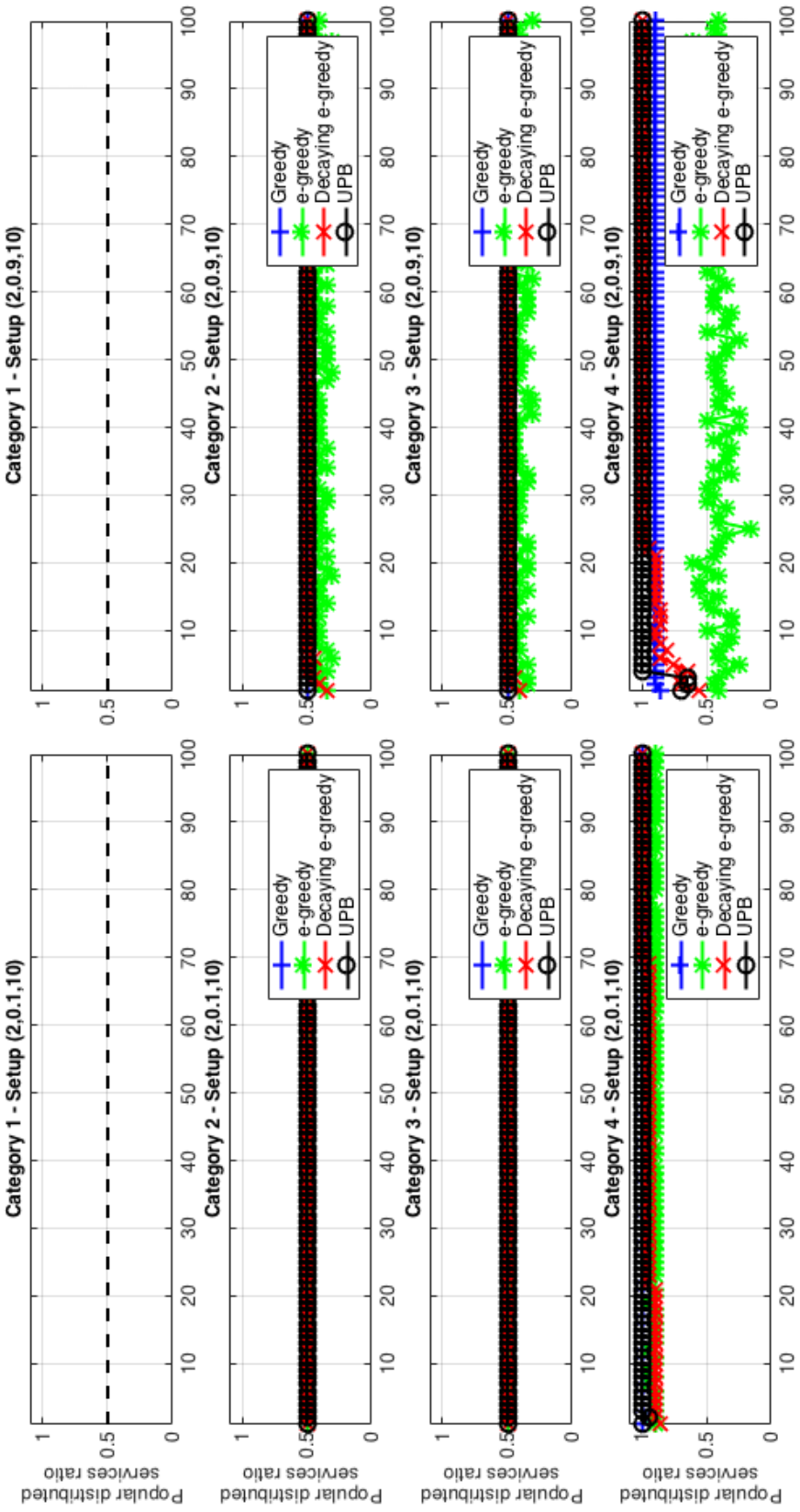


Figure 5.15: Experiment 6: effect on popular distributed services ratio

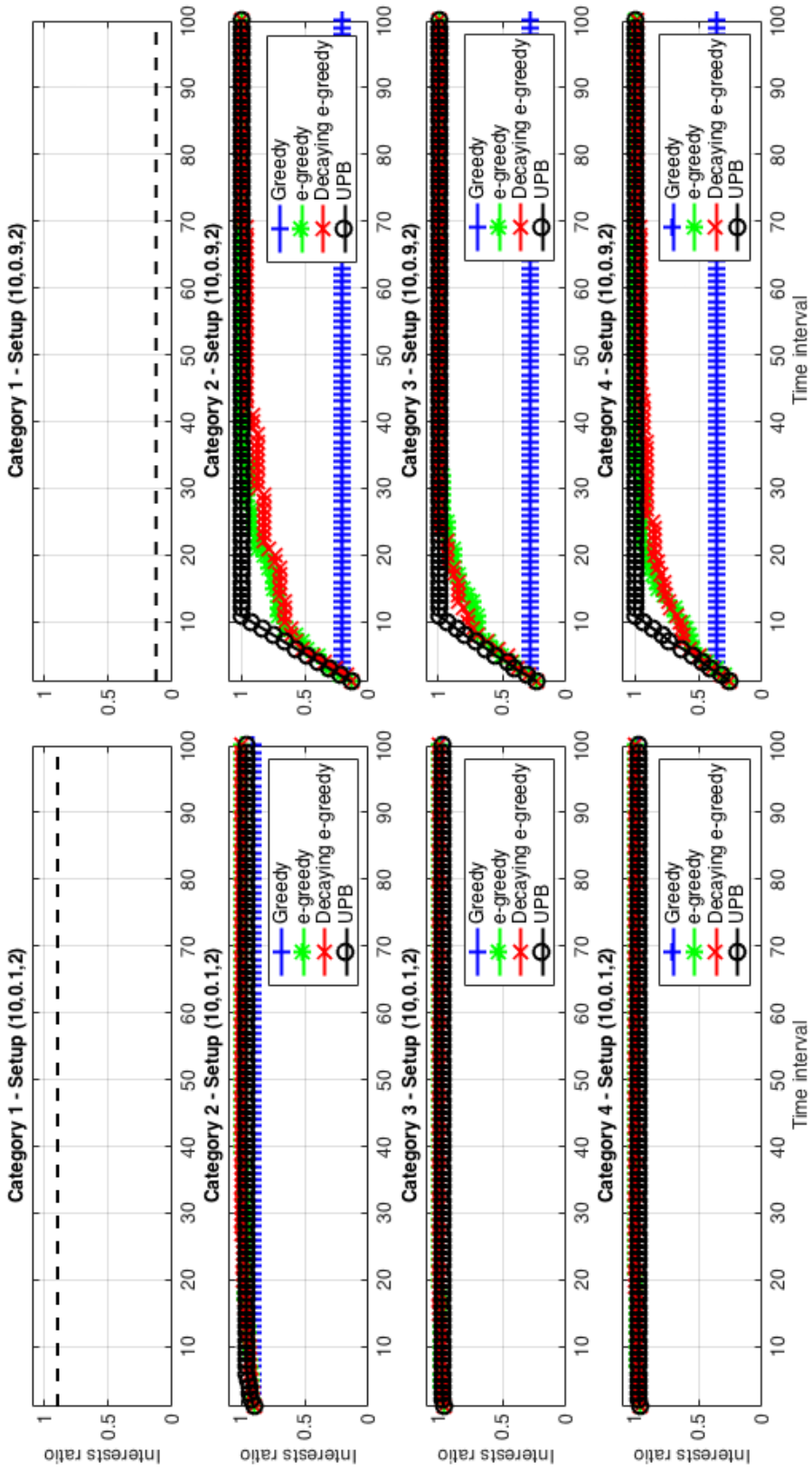


Figure 5.16: Experiment 7: effect on interests ratio

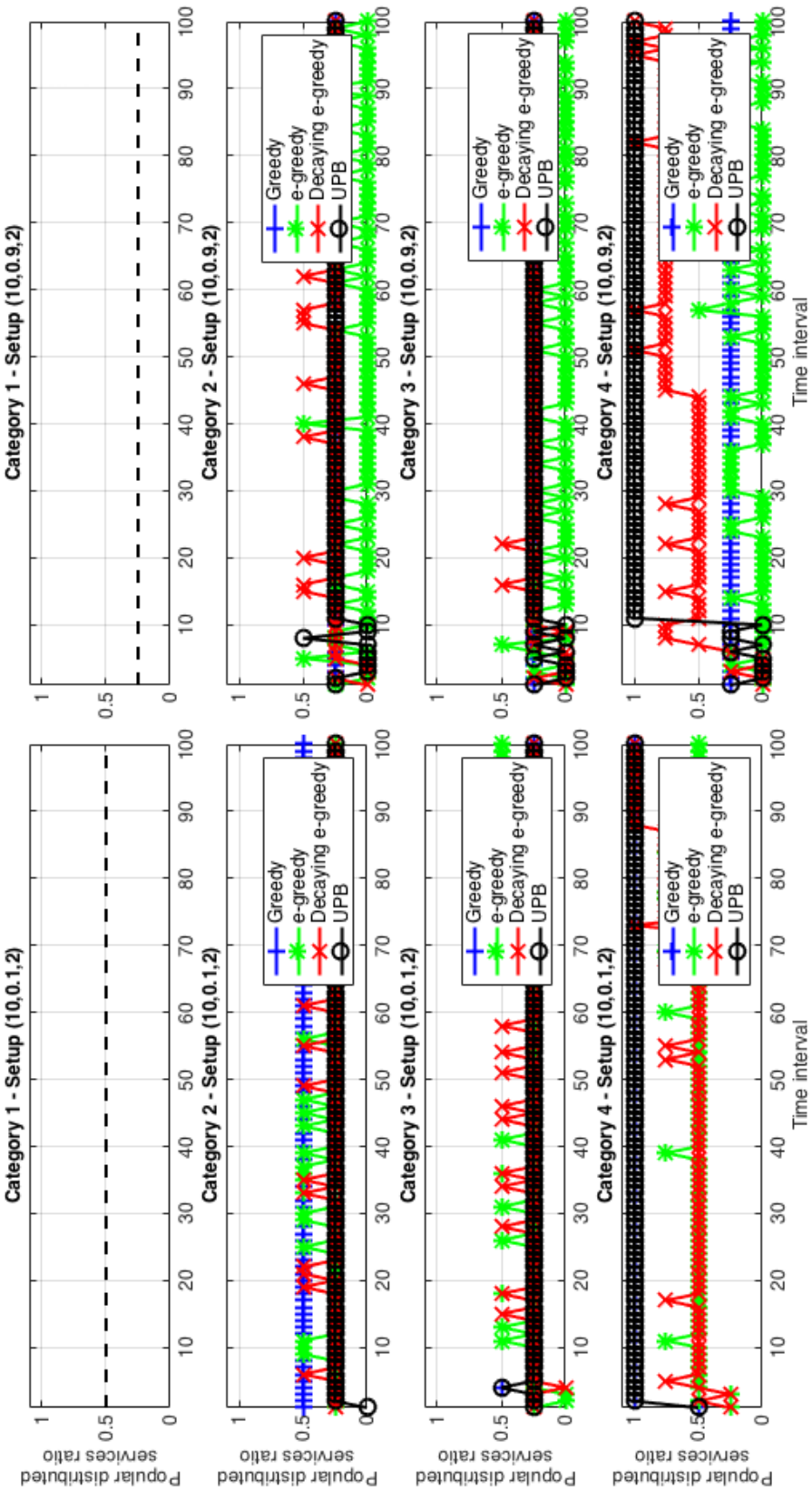


Figure 5.17: Experiment 7: effect on popular distributed services ratio

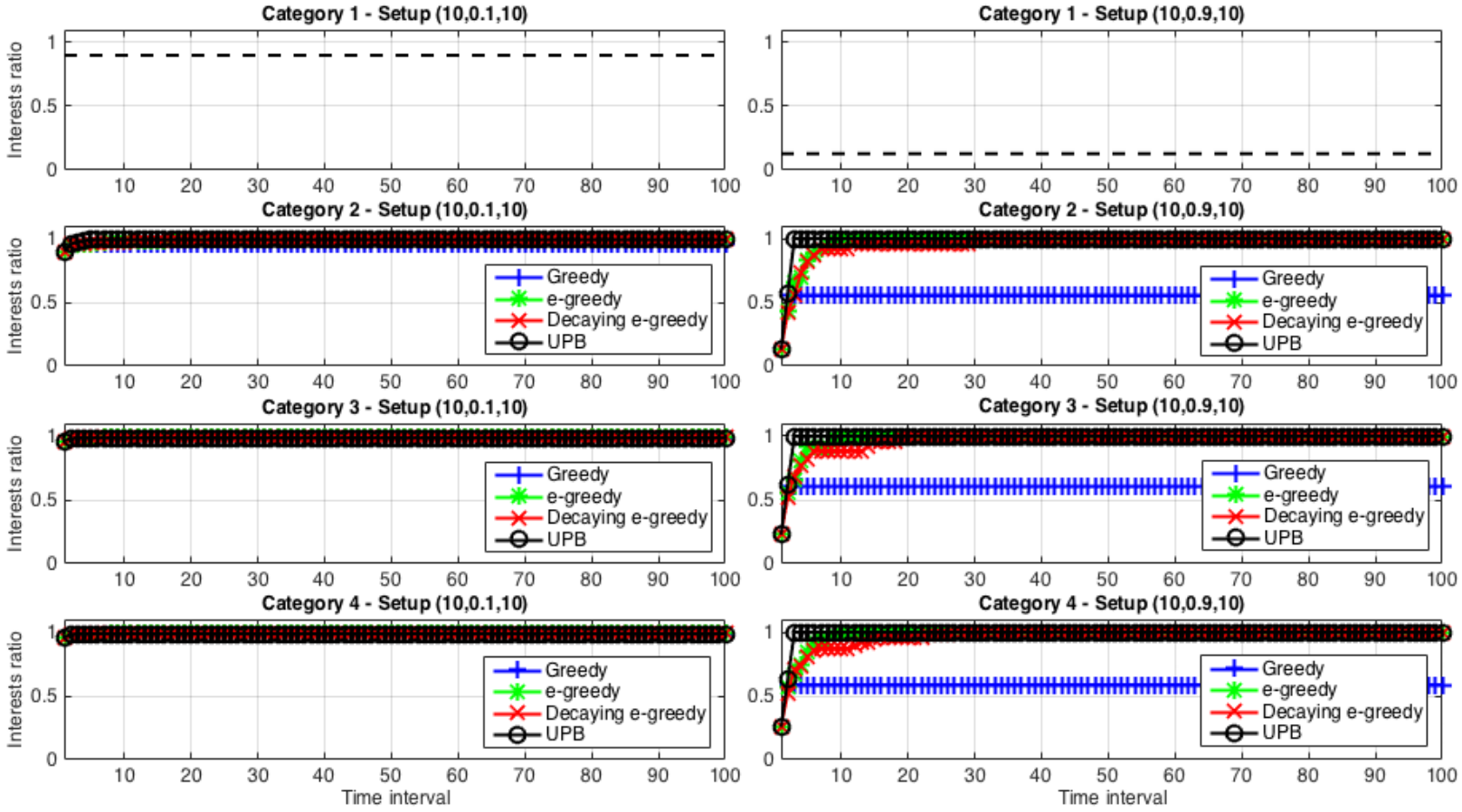


Figure 5.18: Experiment 8: effect on interests ratio

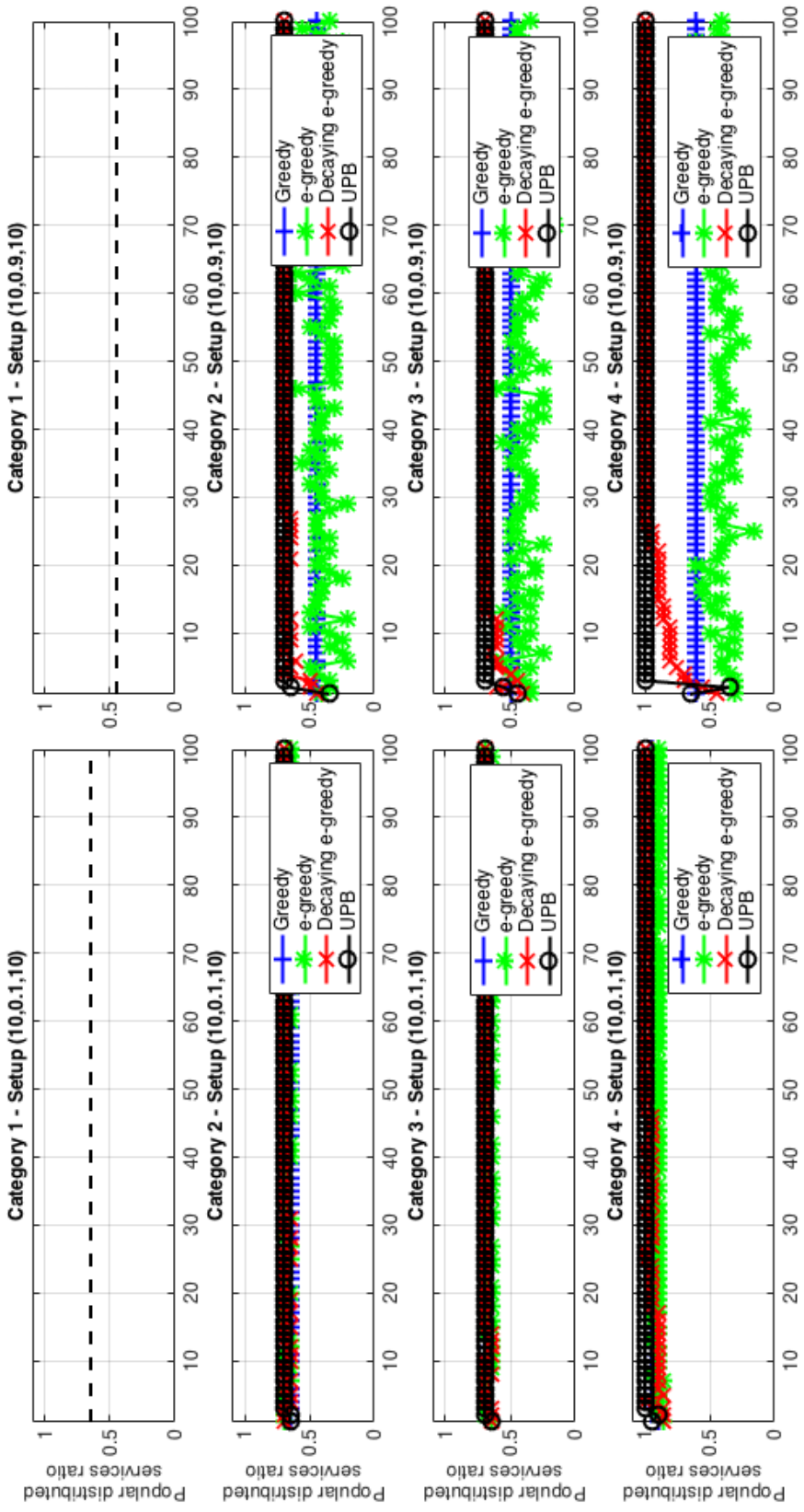


Figure 5.19: Experiment 8: effect on popular distributed services ratio

5.5.3 Varying Network Capacities (Experiments 9 to 12)

In experiments 9 to 12, the network capacity cap_{net} is varied between 2 and 10 in each experiment. The standard deviation of the group interest distributions σ_i^g and the ratio of unknown interests $ratio_{ui}$ are both fixed in each experiment according to Table 5.1.

The interest ratios presented in Figure 5.20 of experiment 9 are already high given that $ratio_{ui} = 0.1$ in addition to the narrow interest distributions of $\sigma_i^g = 2$. The network capacity does not make that difference in terms of interest ratios under such circumstances.

On the other hand, the popular distributed services ratios presented in Figure 5.21 of experiment 9 show that varying the network capacity cap_{net} from 2 to 10 has resulted in a better performance. This is due to the fact that fewer interactions are needed under the higher network capacity of $cap_{net} = 10$. However, the UPB recommender under categories 2 to 4 performs generally better than the other recommenders with the group-based version of it being the one with the best performance.

Figure 5.22 of experiment 10 shows the interest ratios to converge to higher values in a faster rate under the higher network capacity cap_{net} of 10. This is quite natural given the faster rate of consumer interactions under the higher network capacity cap_{net} of 10. The recommendations made throughout category 3 recommenders have led to faster convergence and therefore exploitation as shown in Figure 5.9 of experiment 3. Overall, the UPB recommender is the fastest recommender to explore while managing to identify the set of popular services early and stopping the exploration under category 4 as discussed previously.

Notice that the greedy recommender does not explore that much in general due to its greediness. This becomes more evident as the network capacity cap_{net} gets smaller. This greediness results in a pretty poor performance in terms of the popular distributed services ratio under $cap_{net} = 2$ as shown in Figure 5.23 of experiment 3. However, this ratio is not as bad under the higher network capacity cap_{net} of 10 which leads to more exploration before getting stuck into the suboptimal and less-popular set of services.

Looking at the popular distributed service ratios shown in Figure 5.23 of experiment 10, we can see that the performance of all the recommenders gets better as the network capacity cap_{net} gets higher. Overall, the UPB recommender performs better than the other recommenders with the group-based version of it being the best. The ϵ-greedy recommender performs poorly especially under the small network capacity cap_{net} of 2 given the high ratio of unknown interests $ratio_{ui}$ of 0.9. However, this performance gets a little bit better under the higher network capacity cap_{net} of 10.

Similar to the interest ratios in Figure 5.20 of experiment 9, the interest ratios in Figure 5.24 of experiment 11 are quite high due to the already low ratio of unknown interests $ratio_{ui}$ of 0.1. Having the wide interest distributions of $\sigma_i^g = 10$ does not prevent the interest ratios from quickly converging. In addition, the change in the

network capacity does not make that difference in terms of interest ratios.

In terms of the popular distributed service ratios shown in Figure 5.25 of experiment 11, there is a clear improvement as the network capacity cap_{net} increases. Notice the high network capacity cap_{net} of 10 has allowed for the discovery of the set of common popular services under the wide interest distribution of $\sigma_i^g = 10$. As usual, the UPB recommender is the best performing recommender except under categories 2 and 3. This is mainly due to the wide interest distribution of $\sigma_i^g = 10$ and the low network capacity cap_{net} of 2.

Due to the high unknown interests ratio $ratio_{ui}$ of 0.9, the interest ratios shown in Figure 5.26 of experiment 12 demonstrate the time needed for recommenders to converge. Notice that the recommendations made throughout category 3 recommenders have led to faster convergence and therefore faster exploitation as shown in Figure 5.27 of experiment 12. The UPB recommender converges and explores the fastest whereas the greedy recommender stops exploration early especially under the low network capacity cap_{net} of 2. This early stopping of exploration results in low popular distributed service ratios as shown in Figure 5.27 of experiment 12 for both network capacities of $cap_{net} = 2$ and $cap_{net} = 10$.

The popular distributed service ratios shown in Figure 5.27 of experiment 12 demonstrate the superiority of the UPB recommender under all categories while having the group-based version of it being the best. On the other hand, the ϵ-greedy recommender explores consumer interests excessively given that $\epsilon = ratio_{ui} = 0.9$ which results in low popular distributed service ratios especially under the low network capacity of $cap_{net} = 2$. Overall, having the higher network capacity of $cap_{net} = 10$ improves the performance compared to that under the lower network capacity of $cap_{net} = 2$.

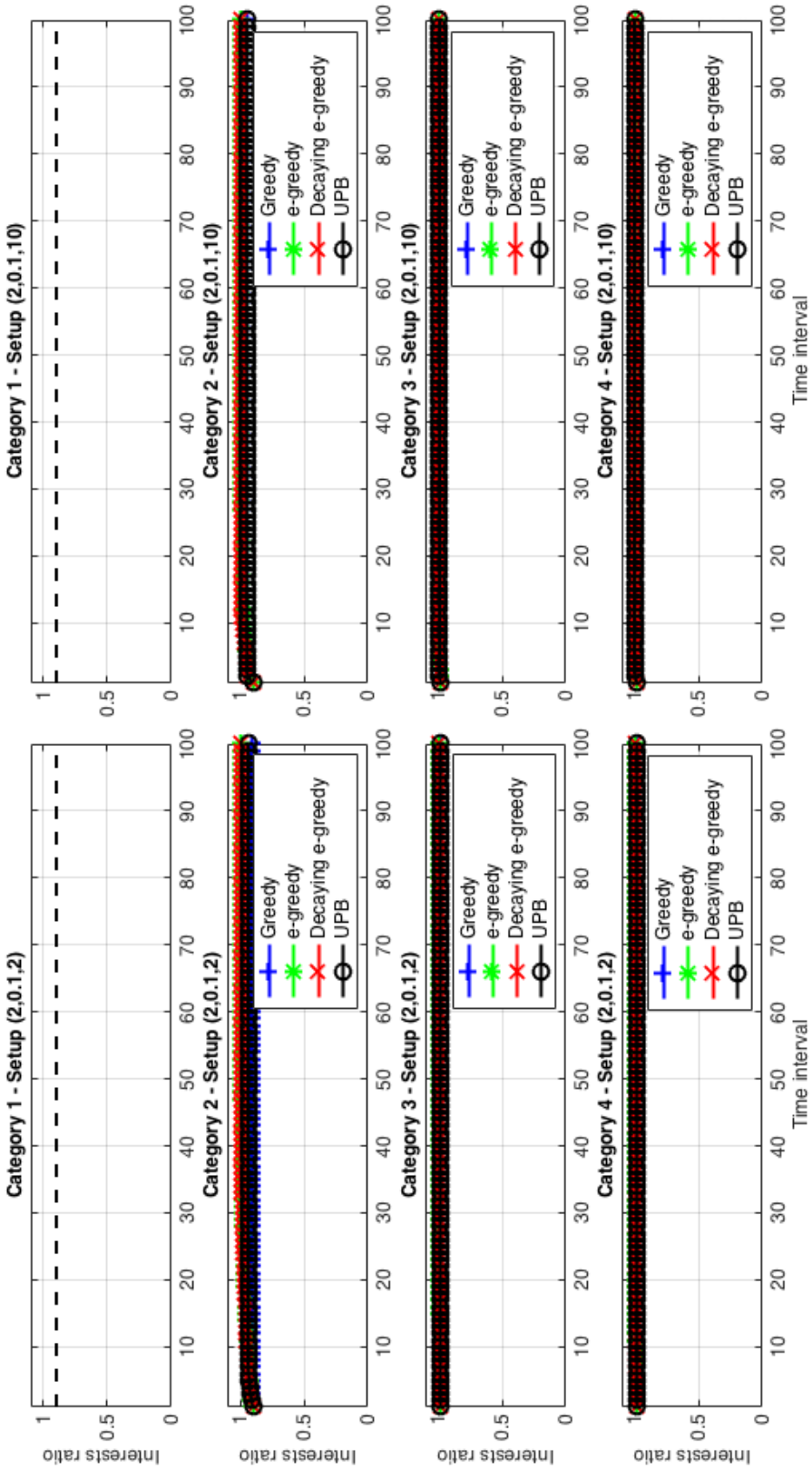


Figure 5.20: Experiment 9: effect on interests ratio

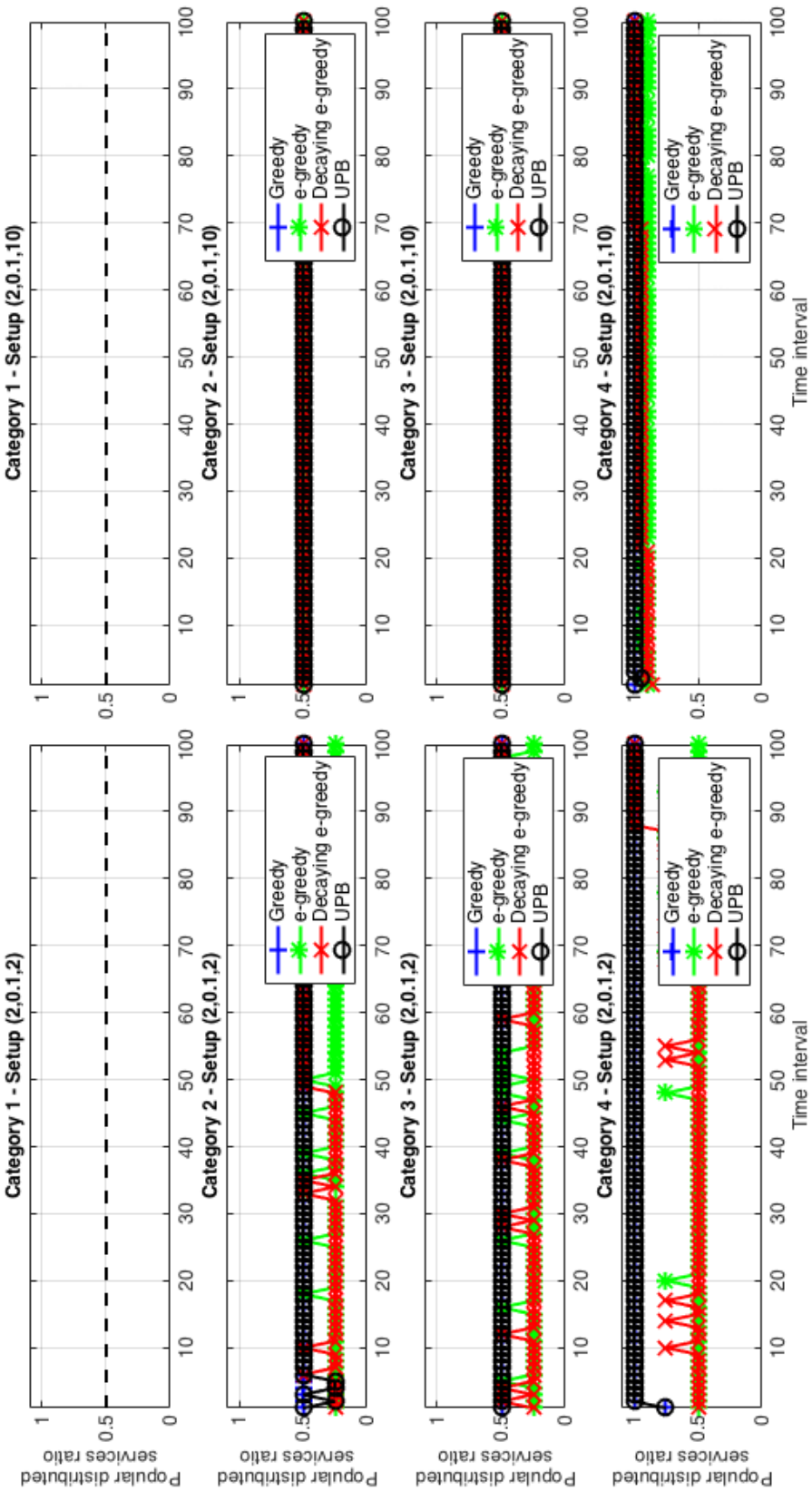


Figure 5.21: Experiment 9: effect on popular distributed services ratio

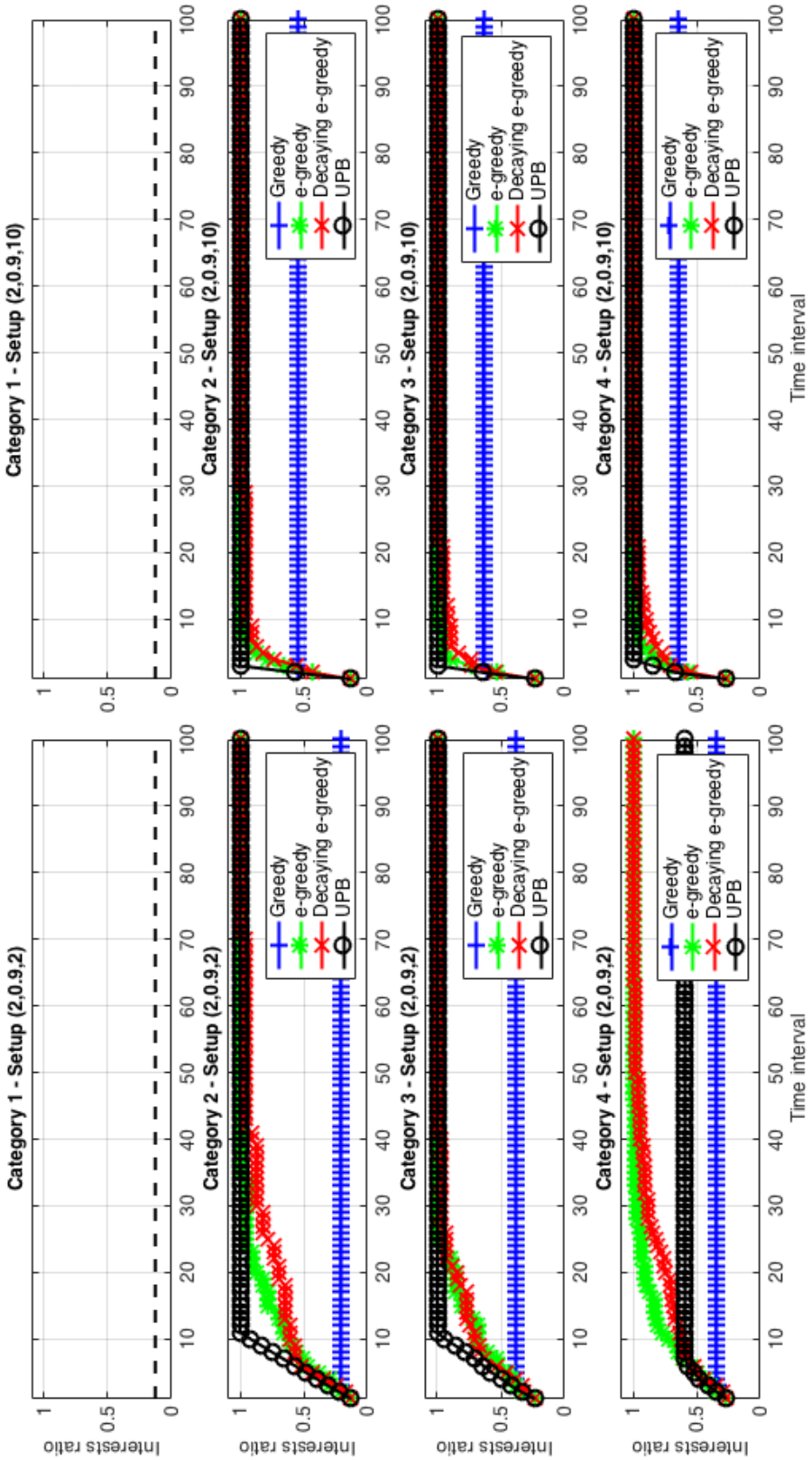


Figure 5.22: Experiment 10: effect on interests ratio

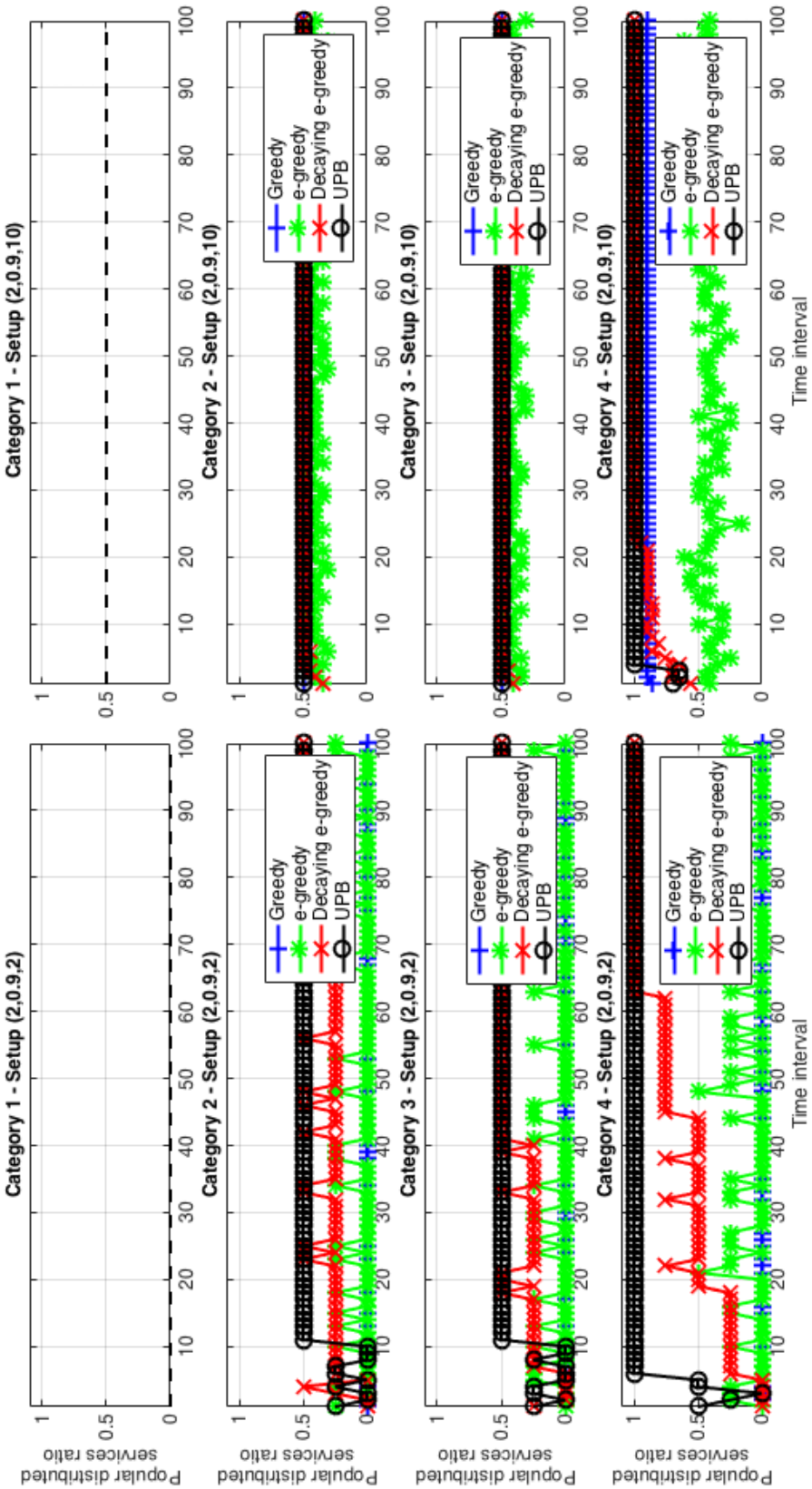


Figure 5.23: Experiment 10: effect on popular distributed services ratio

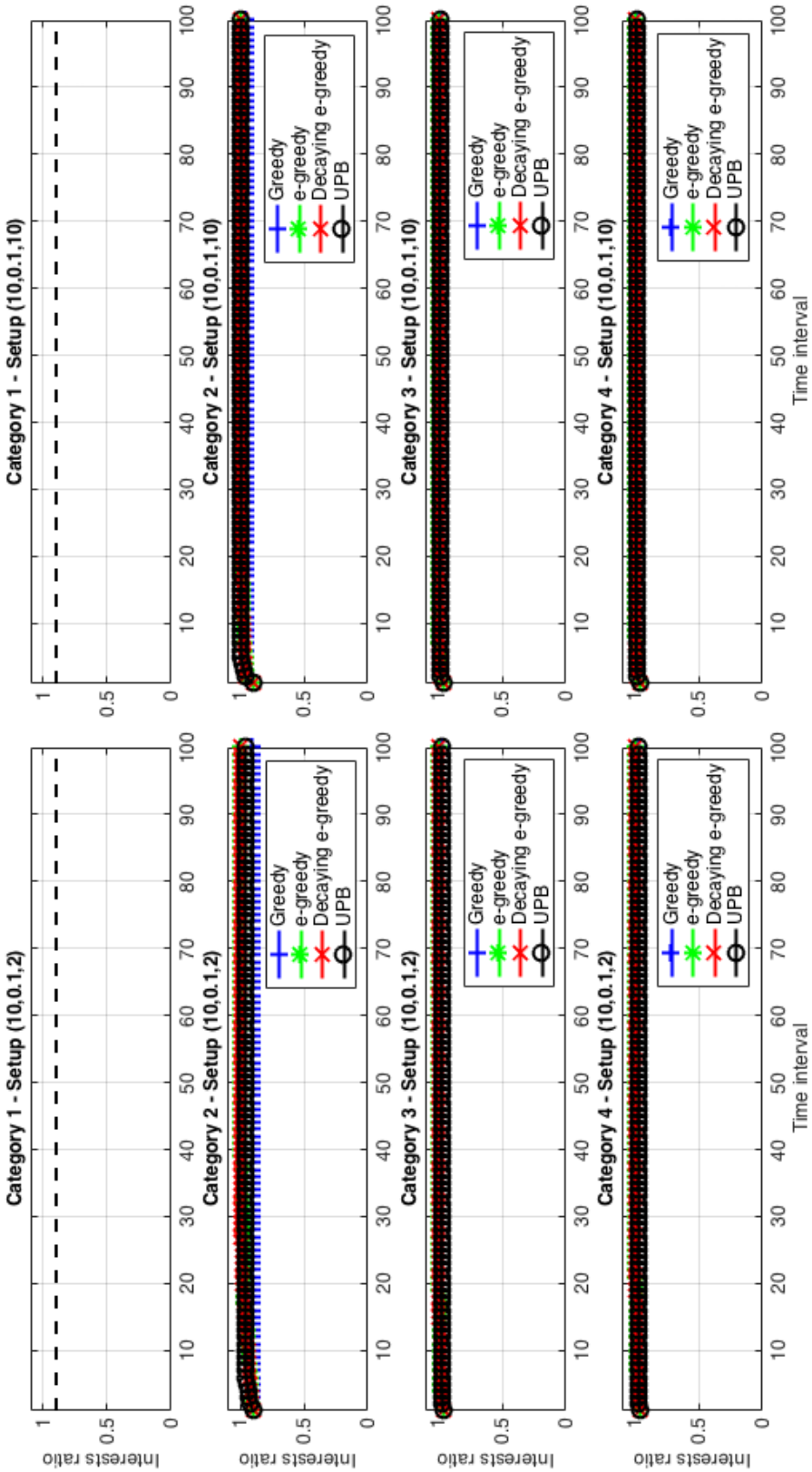


Figure 5.24: Experiment 11: effect on interests ratio

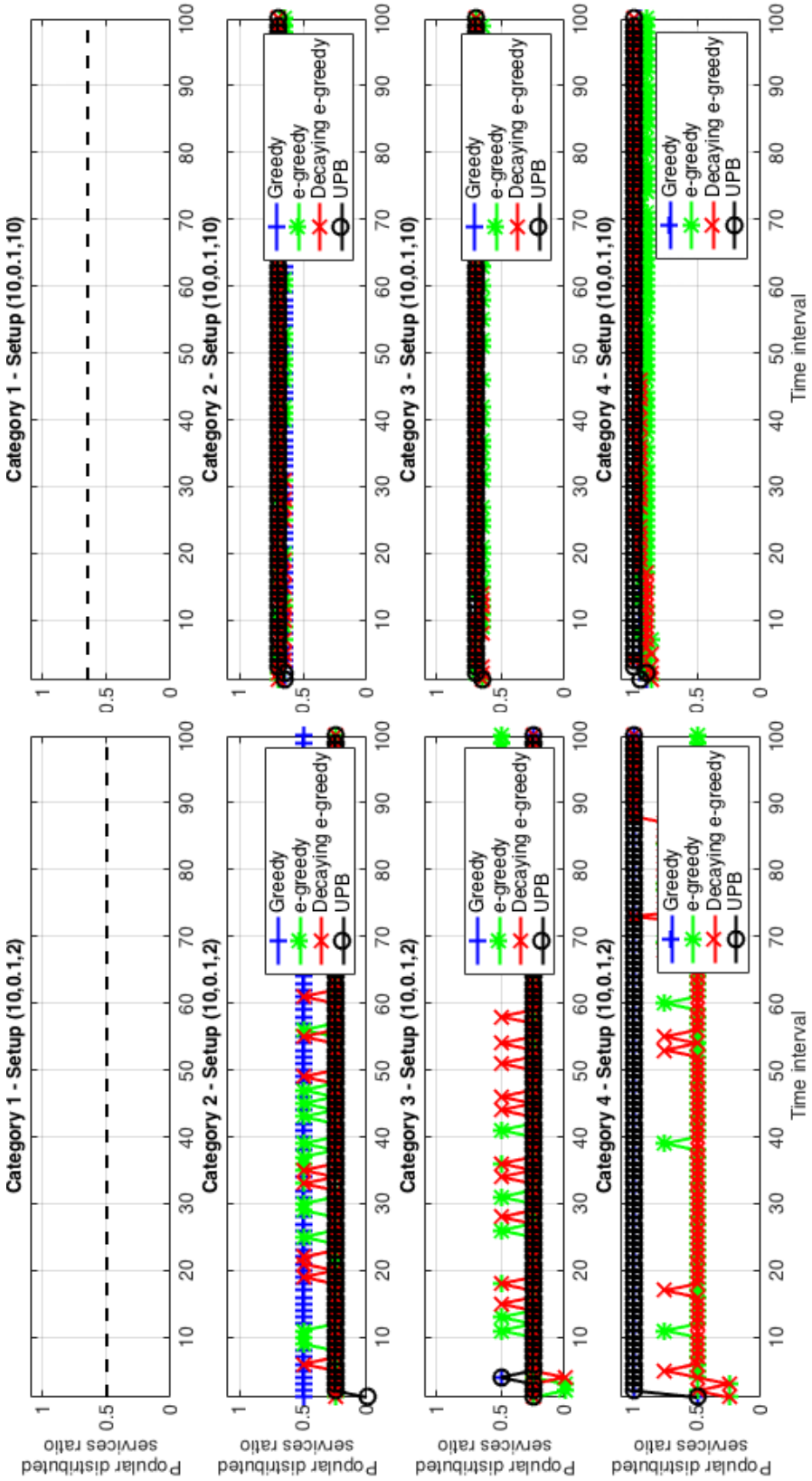


Figure 5.25: Experiment 11: effect on popular distributed services ratio

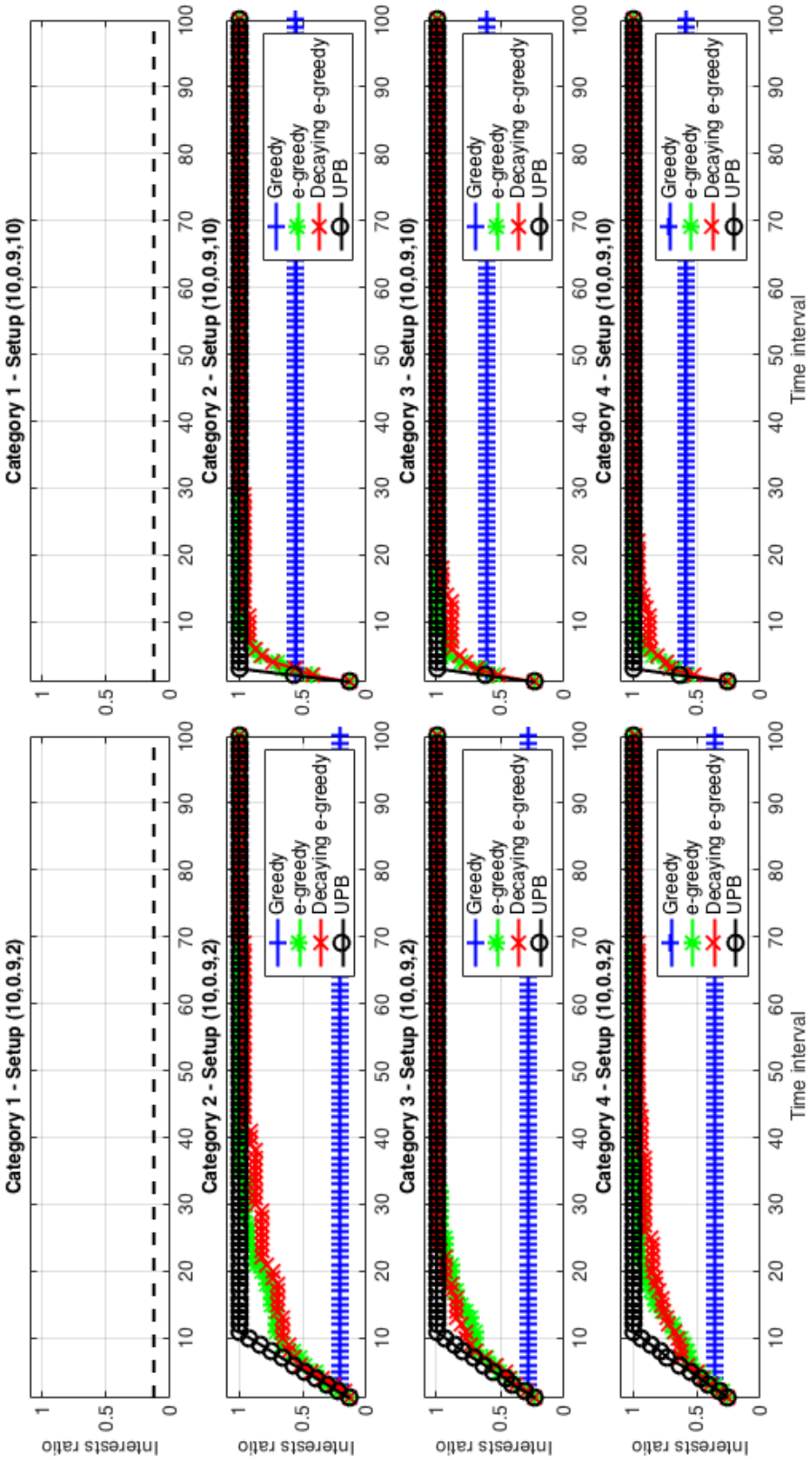


Figure 5.26: Experiment 12: effect on interests ratio

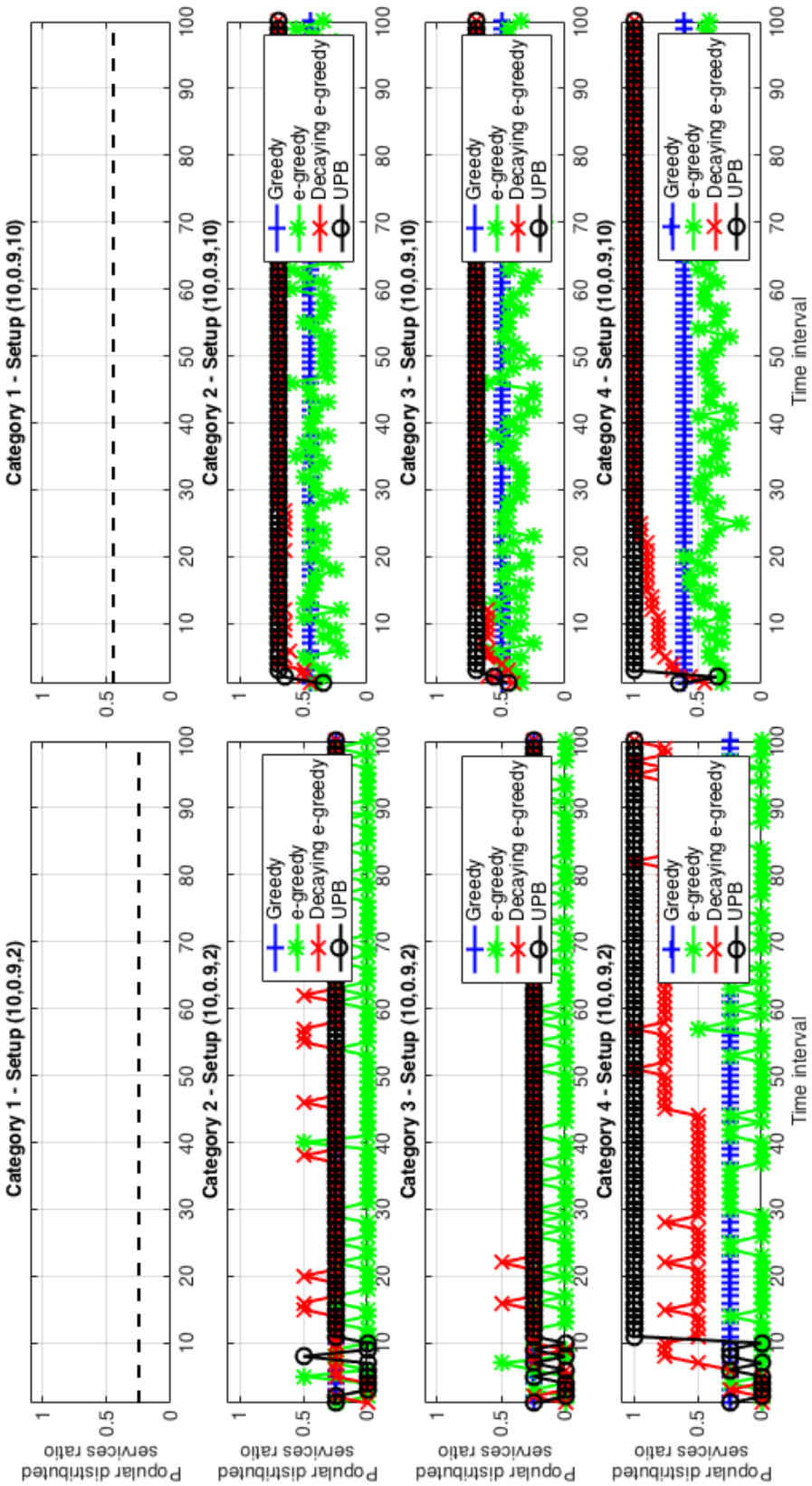


Figure 5.27: Experiment 12: effect on popular distributed services ratio

5.6 Discussion

From experiments 1 to 12, it can be seen that the UPB collaborative and group-based recommender under category 4 makes the best trade off between exploration and exploitation. It has managed to explore consumer interests with the fastest rate compared to the other recommenders under varying consumer and network scenarios as indicated by the resulting consumer interest ratios. In particular, these scenarios include different group interest distributions, unknown interest ratios and network capacities. In fact, this exploration stops as soon as exploitation reaches its optimal performance. Moreover, this recommender exploits its knowledge about consumer interests optimally as indicated by the popular distributed service ratios.

The superior performance of the category 4 UPB recommender is attributed mainly to its ability to bound service popularities given the available consumer interests and exploit the services with the highest resulting bounds greedily. This "optimism in the face of uncertainty" allows for the best tradeoff between exploration and exploitation. Notice that as the time progresses and as the recommender interacts more with the consumers and acquire more knowledge about their true interests, the amount of uncertainty and therefore exploration decreases while exploitation increases.

Another reason for the superior performance of the category 4 UPB recommender is its ability to exploit interest similarities between consumers when making recommendations allowing for faster exploration and therefore exploitation.

Finally, being a group-based recommender allows distributed services to be customized for each group according to the interests of its consumers. This ability leads to more efficient consumer interactions and faster exploration. It also leads to a more personalized consumer experience and better exploitation.

Given all of the above, the UPB collaborative and group-based recommender is chosen to be part of the proposed content distribution system with the main rule of recommending content throughout online operations.

5.7 Summary

In this chapter, the different steps followed throughout designing the recommender are explained. These steps start with synthesizing consumer profiles to allow for recommender experimentation. After that, different recommenders are explained starting with the most basic one and enhancing it gradually. All of these recommenders are experimented in order to choose the best recommender under different consumer interest and network capacity scenarios. The UPB collaborative and group-based recommender has been found to be the best in terms of trading off exploration and exploitation. It does so while benefiting from the similarities between consumer interest profiles and considering the different location groups which consumers belong to. In conclusion, this recommender has been chosen to be part of the proposed content distribution system given its superior performance under varying consumer interest and network capacity scenarios.

Chapter 6

Content Routing Design

6.1 Overview

The purpose of this chapter is to discuss the offline operations of the proposed content distribution procedure which are related to content routing. These operations show the steps followed to design the way content is routed to the different system nodes starting with the first phase of direct V2I content routing, followed by the V2V content segment exchanges and ending with the last phase of V2I content routing.

These operations have already been indicated in Lines 8 to 10 as part of the offline operations shown in Algorithm 3.1 as discussed previously in Chapter 3. They include: estimating the maximum number of content data segments, dividing and allocating the content data segments to the system nodes and optimizing the V2V segment exchange tables at the nodes using bayesian optimization.

Throughout the exchange tables optimization, the optimization search space is visualized in addition to explaining the regression technique used throughout bayesian optimization. This technique has been chosen after comparing it to bayesian optimizations using other regression techniques. This comparison is in terms of the number of segments exchanged and optimization execution times. The regression techniques compared are: GP, RF, BNN and batch-based RF. The performance of the resulting optimal policy is compared against that of the worst policy found so far as well as against the performance of a policy chosen naively.

However and before explaining the offline operations mentioned or proceeding with the rest of the chapter, the details of some functions used throughout these operations are explained first. Notice that throughout the chapter, the purpose and mathematical formulations of all functions and operations are explained in details while leaving their algorithms to Appendix C at the end of the thesis. Moreover, the case study of the Grand River Transit bus service offered throughout the Region of Waterloo, Ontario, Canada is used with the assumptions discussed before in Chapter 3.

6.2 Functions Used

Throughout the upcoming offline operation discussions, the following functions are used:

- *divideData* Function; which is used to decide how many content data segments should be given to each system node,
- *allocateData* Function; which is used to choose which segments are allocated to which system node,
- *computeFeatures* Function; which is used to compute the node features at each time step. These domain-specific features are used to measure the V2V communication potential of nodes such that nodes with the highest potential are given the highest priority to start the V2V segment exchanges,
- *controlRange* Function; which is used to control the broadcasting range of a transmitting node at a certain time step,
- *targetSegments* Function; which is used to target a specific set of content data segments for transmission at a certain time step before making the choice on which segment to transmit,
- *transmitData* Function; which is used to transmit the content data segment chosen out of the set of targeted segments at a certain time step, and
- $followPolicy$ Function; which is used to make nodes follow a certain policy. This policy is parameterized using a vector of feature weights and is used to compute the transmission priorities of nodes at each time step.

Explaining these functions is instrumental for understanding the offline operations explained afterwards in this chapter. However and before explaining these functions, the c^{th} biggest cluster connectivities matrix $\mathbf{C}_b\{c\}$ is extracted at each period p that starts at t^{start} and ends at t^{end}. This extraction is made for all the biggest clusters in each period p of the n_p periods. These $\mathbf{C}_b\{c\}$ matrices are needed as inputs for the upcoming functions as well as the offline operations addressed in this chapter.

To do these extractions, the total number of time steps $n_{T_{period}}$ is computed first within the period under consideration given t^{start} and t^{end} as follows:

$$n_{T_{period}} = t^{end} - t^{start} + 1$$

Then, $\mathbf{C}_b\{c\}$ is computed at each time index t as follows:

$$\mathbf{C}_b\{c\}[t, *, *] = \mathbf{C}[t + t^{start} - 1, \mathbf{ID}^b_{nc}\{c\}, \mathbf{ID}^b_{nc}\{c\}]$$

where t ranges from 1 to $n_{T_{period}}$, $\mathbf{C}_b\{c\}[t, i, j]$ is the connectivity between node i and node j from the c^{th} biggest nodes cluster at time index t. Notice that the time index of the connectivities matrix $\mathbf{C}$ is shifted by $(t^{start} - 1)$ since matrix $\mathbf{C}_b$ has time indices between 1 and $n_{T_{period}}$ only corresponding to the time indices of period p that

starts at t^{start} and ends at t^{end}. Also notice that the node indices matrix of the c^{th} biggest nodes cluster $\mathbf{ID}^b_{nc}\{c\}$ has already been computed in Chapter 4 as part of the cluster node indices matrix $\mathbf{ID}_{nc}\{p\}$ at period p.

Refer to Algorithm C.1 in Appendix C for further details about this connectivities matrix extraction operation. Further details about the upcoming functions can also be found in Appendix C in Algorithms C.2 to C.8 at the end of the thesis.

6.2.1 *divideData* Function

This function is used to divide content data segments between the nodes of the c^{th} biggest cluster extracted using the function (*extractCluster*) shown previously in Chapter 4. This division is done in proportion to the total connectivities summation each node has with the other nodes in the cluster. The number of all content data segments divided between the nodes is denoted by $\mathbf{n}_{as}[k]$ where k is the index used to choose a specific value of vector $\mathbf{n}_{as}$.

The function (*divideData*) starts by computing the number of nodes in the biggest cluster under consideration n^b_n as follows:

$$n^b_n = length(\mathbf{ID}^b_{nc}\{c\})$$

where $\mathbf{ID}^b_{nc}\{c\}$ is the node indices matrix of the c^{th} biggest nodes cluster computed previously in Chapter 4.

After computing n^b_n and given t^{start} and t^{end}, $n_{T_{period}}$ is computed and the function goes over all n^b_n nodes in order to compute the number of content data segments $\mathbf{n}_{ns}[i]$ for each node i as follows:

$$\mathbf{n}_{ns}[i] = \sum_{t=1}^{n_{T_{period}}} \sum_{j=1}^{n^b_n} \mathbf{C}_b\{c\}[t,i,j]$$

where $\mathbf{C}_b\{c\}$ is the connectivities matrix of the c^{th} biggest nodes cluster extracted previously. Given the resulting values of the vector $\mathbf{n}_{ns}$, $\mathbf{n}_{ns}$ is recomputed using the formula:

$$\mathbf{n}_{ns} \leftarrow \lfloor \mathbf{n}_{as}[k] \times \mathbf{n}_{ns} / (\sum_{i=1}^{n^b_n} \mathbf{n}_{ns}[i]) \rfloor$$

The idea here is to divide the k^{th} total number of content data segments $\mathbf{n}_{as}[k]$ across the n^b_n nodes in proportion to their corresponding connectivities summation with the other nodes in the cluster normalized by the total connectivities summation of all n^b_n nodes. Notice that one segment is the least unit of content that can be given. Also notice that the vector of the different numbers of content data segments $\mathbf{n}_{as}$ has been set to [50 100 ... 2000] according to the case study adopted in Chapter 3. This vector is used when estimating the maximum number of distributable data segments as will be presented in Section 6.3.

6.2.2 *allocateData* Function

This function is used to allocate specific content data segments to each node according to the segment proportions computed by the function (*divideData*).

At the beginning, the function (*allocateData*) sets the matrix of segment allocations $\mathbf{SA}$ of size $n_n^b \times \sum_{i=1}^{n_n^b} \mathbf{n}_{ns}[i]$ to the zero matrix where $\mathbf{SA}[i,j] = 1$ if segment j is allocated to node i and $\mathbf{SA}[i,j] = 0$, otherwise. After that, it goes over all the n_n^b nodes in order to allocate content data segments to each node. For the first node i, we have:

$$\mathbf{SA}[i, 1 : \mathbf{n}_{ns}[i]] = \mathbf{1}^{1 \times length(\mathbf{n}_{ns}[i])}$$

which means that all the first $\mathbf{n}_{ns}[i]$ content data segments are allocated to node i where $\mathbf{1}^{1 \times length(\mathbf{n}_{ns}[i])}$ is a vector of size $(1 \times length(\mathbf{n}_{ns}[i]))$ with all values set to 1. It does also mean that, as of the first node, the last segment allocated has the index $sa_l = \mathbf{n}_{ns}[i]$. Therefore, the following applies for the remaining nodes:

$$\mathbf{SA}[i, sa_l + 1 : sa_l + \mathbf{n}_{ns}[i]] = \mathbf{1}^{1 \times length(\mathbf{n}_{ns}[i])}$$

$$sa_l \leftarrow sa_l + \mathbf{n}_{ns}[i]$$

which means allocating each node i with the next $\mathbf{n}_{ns}[i]$ content data segments and shifting the index of the last segment allocated sa_l each time by $\mathbf{n}_{ns}[i]$ until all data segments are allocated.

6.2.3 *computeFeatures* Function

This function is used to compute the vector of features for each node within the c^{th} biggest nodes cluster at time step t. These features are chosen based on my domain knowledge of the field of vehicular networking and their weighted summation gives the node utilities needed to prioritize the V2V segment exchanges of the different nodes at each time step t as will be presented in the upcoming function ($followPolicy$).

The inputs to the function (*computeFeatures*) include:

- the c^{th} biggest cluster connectivities matrix $\mathbf{C}_b\{c\}$,
- the segment allocations matrix $\mathbf{SA}$ computed using the function (*allocateData*), and
- the vector $\mathbf{n}_{ns}$ representing the number of data segments at each node which are divided using the function (*divideData*).

The features for all the n_n^b nodes within the c^{th} biggest nodes cluster are given by the matrix $\mathbf{F}^n$ defined as $\mathbf{F}^n = [\mathbf{f}_1^n \ \mathbf{f}_2^n \ \mathbf{f}_3^n \ \mathbf{f}_4^n]^{n_n^b \times 4}$ where:

- $\mathbf{f}_1^n$ is the feature vector representing each node's normalized number of direct neighbors,

- $\mathbf{f}_2^n$ is the feature vector representing each node's average fraction of segments missing at the neighboring nodes which are available at the node,
- $\mathbf{f}_3^n$ is the feature vector representing each node's normalized number of segments missing across all neighboring nodes which are available at the node, and
- $\mathbf{f}_4^n$ is the feature vector representing each node's difference between 1 and the normalized number of indirect neighboring nodes located around the node and its direct neighbors.

Initially, this matrix $\mathbf{F}^n$ is set to the zero matrix $\mathbf{0}^{n_n^b \times 4}$ at each time step t it is computed. After that, the function ($computeFeatures$) goes over all the n_n^b nodes to compute the feature values for each node i. The first feature $\mathbf{f}_1^n[i]$ of node i is computed as follows:

$$\mathbf{f}_1^n[i] = \sum_{j=1}^{n_n^b} \mathbf{C}_b\{c\}[t,i,j]/n_n^b$$

which counts the number of times a node j within the cluster has a connectivity with node i at time step t indicating that it is a direct neighbor. This count is then normalized by the total number of nodes within the cluster n_n^b. If there are no neighbors to node i (i.e. $\mathbf{f}_1^n[i] = 0$), then $\mathbf{f}_{2:4}^n[i]$ are assumed to equal 0. Otherwise, the function proceeds with computing the remaining node features $\mathbf{f}_{2:4}^n[i]$. However, the function needs to compute first the vector of segment indices $\mathbf{id}_s^n$ available at node i as follows:

$$\mathbf{id}_s^n = \underset{j}{argfind}(\mathbf{SA}[i,j] = 1)$$

where the function ($argfind$) is used to find the vector containing any index j of a segment that is available at node i such as $\mathbf{SA}[i,j] = 1$. The vector of neighboring node indices $\mathbf{id}_{ne}^n$ of node i also needs to be computed as follows:

$$\mathbf{id}_{ne}^n = \underset{j}{argfind}(\mathbf{C}_b\{c\}[t,i,j] = 1)$$

where the function ($argfind$) is used here to find the vector containing any index j of a node that has a connectivity with node i at time step t within the c^{th} biggest cluster such as $\mathbf{C}_b\{c\}[t,i,j] = 1$. Given $\mathbf{id}_{ne}^n$, the function ($computeFeatures$) goes over all the neighboring node indices in $\mathbf{id}_{ne}^n$ in order to find the vector of missing segment indices $\mathbf{id}_{ms}^{ne}$ at each neighboring node $\mathbf{id}_{ne}^n[j]$ as follows:

$$\mathbf{id}_{ms}^{ne} = \underset{k}{argfind}(\mathbf{SA}[\mathbf{id}_{ne}^n[j],k] = 0)$$

where the function ($argfind$) is used to find any index k of a missing segment at neighboring node $\mathbf{id}_{ne}^n[j]$ such that $\mathbf{SA}[\mathbf{id}_{ne}^n[j],k] = 0$. If there are no missing segments (i.e. $\mathbf{id}_{ms}^{ne} = \emptyset$) at neighboring node $\mathbf{id}_{ne}^n[j]$, then this node is assumed to have no impact on $\mathbf{f}_2^n[i]$. Otherwise, the function ($computeFeatures$) computes $\mathbf{f}_2^n[i]$ as follows:

$$\mathbf{f}_2^n[i] \leftarrow (\mathbf{f}_2^n[i] \times (j-1) + length(\mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne})/length(\mathbf{id}_{ms}^{ne}))/j$$

where j is the index of the current neighboring node. Notice that the number of missing segments at the neighboring node is determined using the function ($length$) as follows: $length(\mathbf{id}_{ms}^{ne})$ and the number of those missing segments which are available at node i is also determined using the function ($length$) as follows: $length(\mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne})$.

The function ($computeFeatures$) proceeds afterwards with computing the vector $\mathbf{id}_{cms}^{ne}$ representing the common missing segment indices of neighboring nodes which are also available at node i. For the first neighboring node, this is computed as follows:

$$\mathbf{id}_{cms}^{ne} = \mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne}$$

For the remaining nodes, it is computed as follows:

$$\mathbf{id}_{cms}^{ne} \leftarrow \mathbf{id}_{cms}^{ne} \cap (\mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne})$$

Given $\mathbf{id}_{cms}^{ne}$, the third feature $\mathbf{f}_3^n[i]$ for node i is computed as follows:

$$\mathbf{f}_3^n[i] = length(\mathbf{id}_{cms}^{ne})/(\sum_{j=1}^{n_n^b} \mathbf{n}_{ns}[j])$$

Where the normalization is made here by dividing over the total number of node segments $\sum_{j=1}^{n_n^b} \mathbf{n}_{ns}[j]$.

The function ($computeFeatures$) does also find the vector of indirect neighboring node indices $\mathbf{id}_{nene}^n$ around node i. This vector is initially set to the empty vector $\emptyset$ before proceeding with its computation. This computation is made using the following two formulas:

$$\mathbf{id}_{nene}^n \leftarrow \mathbf{id}_{nene}^n \cup \underset{k}{argfind}(\mathbf{C}_b\{c\}[t, \mathbf{id}_{ne}^n[j], k] = 1)$$

$$\mathbf{id}_{nene}^n \leftarrow \mathbf{id}_{nene}^n - (\mathbf{id}_{ne}^n \cup \{i\})$$

where the function ($argfind$) is used to find any index k of an indirect neighboring node that is a neighbor to the neighbor $\mathbf{id}_{ne}^n[j]$ of node i at time step t within the c^{th} biggest nodes cluster such that $\mathbf{C}_b\{c\}[t, \mathbf{id}_{ne}^n[j], k] = 1$. This, however, includes node i and might also include its direct neighbors with the indices vector $\mathbf{id}_{ne}^n$. Therefore, the second formula above ensures that this does not happen by recomputing $\mathbf{id}_{nene}^n$ as shown. Given $\mathbf{id}_{nene}^n$, the fourth feature $\mathbf{f}_4^n[i]$ for node i is computed as follows:

$$\mathbf{f}_4^n[i] = 1 - length(\mathbf{id}_{nene}^n)/n_n^b$$

where the normalization is made here by dividing over n_n^b.

6.2.4 *controlRange* Function

This function controls the broadcasting range of a transmitting node id^{tx} at time step t. This is done by outputting the refined indices vector $\mathbf{id}_{ne}^{tx}$ of neighboring nodes around id^{tx} and their distances vector $\mathbf{d}_{ne}^{tx}$ from it. These refined neighboring nodes are chosen to receive from id^{tx} in a way that does not interfere with the broadcasting range of any nearby transmitting node. A power control scheme can be implemented to enforce such set of neighbors.

The inputs to the function (*controlRange*) include:

- the unrefined indices vector $\mathbf{id}_{ne}^{tx}$ of neighboring nodes around the transmitting node id^{tx} before controlling its broadcasting range,
- the matrix of member indices of the c^{th} biggest cluster node $\mathbf{ID}_{nc}^{b}\{c\}$ extracted previously in Chapter 4, and
- the matrix of node statuses $\mathbf{STAT}^n$ defined as $\mathbf{STAT}^n = [\mathbf{stat}_1^n \ \mathbf{stat}_2^n \ \mathbf{stat}_3^n]^{n_n^b \times 3}$ where:
  - $\mathbf{stat}_1^n$ is the vector representing the transmission status of each node such as:
    - $\mathbf{stat}_1^n[i] = \begin{cases} 1 & if\ node\, i\ is\ transmitting \\ 0 & otherwise \end{cases}$
  - $\mathbf{stat}_2^n$ is the vector representing the coverage status of each node such as:
    - $\mathbf{stat}_2^n[i] = \begin{cases} 1 & if\ node\, i\ is\ under\ coverage \\ & of\ another\ transmitting\ node \\ 0 & otherwise \end{cases}$
  - and $\mathbf{stat}_3^n$ is the vector representing the reception status of each node such as:
    - $\mathbf{stat}_3^n[i] = \begin{cases} 1 & if\ node\, i\ is\ receiving \\ 0 & otherwise \end{cases}$

The function (*controlRange*) starts by going over the transmitting node neighbors, with the indices specified by the vector $\mathbf{id}_{ne}^{tx}$, and computing their distances from id^{tx} using the function (*distance*). These distances are computed given the latitudes matrix **LATS**, the longitudes matrix **LONS** and the matrix $\mathbf{ID}_{nc}^{b}\{c\}$ which all have already been computed in Chapter 4. Notice that the time index t in the matrices **LATS** and **LONS** has to be shifted by $(t^{start} - 1)$ first because it counts the time steps only within period p under consideration.

After that, the function (*controlRange*) proceeds by eliminating any neighboring node $\mathbf{id}_{ne}^{tx}[j]$ around id^{tx} that is either transmitting (i.e. $\mathbf{stat}_1^n[\mathbf{id}_{ne}^{tx}[j]] = 1$) or receiving (i.e. $\mathbf{stat}_3^n[\mathbf{id}_{ne}^{tx}[j]] = 1$) from the set of nodes receiving from id^{tx} where j is the neighboring node index. In addition, any neighboring node k that is located more than $\mathbf{d}_{ne}^{tx}[j]$ away from id^{tx} (i.e. $\mathbf{d}_{ne}^{tx}[k] \geq \mathbf{d}_{ne}^{tx}[j]$) is also eliminated. The intuition here

is that node id^{tx} should reduce its broadcasting range so that it does not interfere with any transmitting/receiving node or any other node located beyond that. This elimination is made by creating the indices vector $\mathbf{id}_{ene}^{tx}$ of all eliminated neighboring nodes around id^{tx} as follows:

$$\mathbf{id}_{ene}^{tx} \leftarrow \mathbf{id}_{ene}^{tx} \cup \underset{k}{argfind}(\mathbf{d}_{ne}^{tx}[k] \geq \mathbf{d}_{ne}^{tx}[j])$$

where the function ($argfind$) is used to find any index k of a neighboring node that has $\mathbf{d}_{ne}^{tx}[k] \geq \mathbf{d}_{ne}^{tx}[j]$ where j is the index of the currently transmitting/receiving neighboring node. Notice that $\mathbf{id}_{ene}^{tx}$ is initially set to the empty vector $\emptyset$.

Following the $\mathbf{id}_{ene}^{tx}$ vector computation, the function ($controlRange$) eliminates all neighboring nodes represented by this vector from the indices vector $\mathbf{id}_{ne}^{tx}$ and the distances vector $\mathbf{d}_{ne}^{tx}$ as follows:

$$\mathbf{id}_{ne}^{tx}[\mathbf{id}_{ene}^{tx}] = \emptyset$$

$$\mathbf{d}_{ne}^{tx}[\mathbf{id}_{ene}^{tx}] = \emptyset$$

This way the broadcasting range of id^{tx} is controlled given the fact that transmission is restricted to only those neighboring nodes of the final indices vector $\mathbf{id}_{ne}^{tx}$ such that the broadcasting range of id^{tx} is set to $max(\mathbf{d}_{ne}^{tx})$ as will be presented in the upcoming function ($transmitData$).

6.2.5 *targetSegments* Function

This function finds the indices vector $\mathbf{id}_{ts}$ of targeted segments which are available at the transmitting node id^{tx} and are candidates for transmission. The chosen segment, on the other hand, is picked up from these segments using the upcoming function ($transmitData$).

The inputs to the function ($targetSegments$) include:

- the segment allocations matrix $\mathbf{SA}$ computed using the function ($allocateData$), and
- the refined indices vector $\mathbf{id}_{ne}^{tx}$ of neighboring nodes around the transmitting node id^{tx} which is computed using the function ($controlRange$).

The function ($targetSegments$) starts by computing the indices vector $\mathbf{id}_{s}^{tx}$ of segments available at the transmitting node id^{tx} as follows:

$$\mathbf{id}_{s}^{tx} = \underset{j}{argfind}(\mathbf{SA}[id^{tx}, j] = 1)$$

where the function ($argfind$) is used to find any index j of an available segment at id^{tx} such as $\mathbf{SA}[id^{tx}, j] = 1$. After that, the function ($targetSegments$) goes over all neighbors of id^{tx} and whenever a neighbor $\mathbf{id}_{ne}^{tx}[j]$ is not covered (i.e. $\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 0$), this neighbor is inquired for the indices vector $\mathbf{id}_{ms}^{ne}$ of its missing segments as follows:

$$\mathbf{id}_{ms}^{ne} = \underset{k}{argfind}(\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], k] = 0)$$

where the function ($argfind$) is used here to find any index k of a missing segment at neighbor $\mathbf{id}_{ne}^{tx}[j]$ such as $\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], k] = 0$. Notice that node $\mathbf{id}_{ne}^{tx}[j]$ can neither be transmitting nor receiving given the range-control or refinement made by the function ($controlRange$) before using the vector $\mathbf{id}_{ne}^{tx}$ here (i.e. $\mathbf{stat}_1^n[\mathbf{id}_{ne}^{tx}[j]] = 0$ and $\mathbf{stat}_3^n[\mathbf{id}_{ne}^{tx}[j]] = 0$). Also notice that whenever node $\mathbf{id}_{ne}^{tx}[j]$ is covered (i.e. $\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 1$), the missing segments of this node will have no impact on $\mathbf{id}_{ts}$ since this node is not going to receive from id^{tx} anyway without having interference.

Given $\mathbf{id}_{ms}^{ne}$, the indices vector $\mathbf{id}_{ts}$ of targeted segments can be computed as follows:

$$\mathbf{id}_{ts} \leftarrow \mathbf{id}_{ts} \cup (\mathbf{id}_{s}^{tx} \cap \mathbf{id}_{ms}^{ne})$$

where the vector $\mathbf{id}_{ts}$ is set initially to the empty vector $\emptyset$.

Notice that after going over all the neighboring nodes of id^{tx}, the vector $\mathbf{id}_{ts}$ will include the indices of all targeted segments which are missing in at least one neighboring node while being available at the transmitting node id^{tx}.

6.2.6 *transmitData* Function

This function is used to transmit the chosen data segment id_{cs} by the transmitting node id^{tx} at time step t. This transmission action is added to the matrix of actions $\mathbf{A}$ whenever it occurs. It does also update the statuses matrix $\mathbf{STAT}^n$ whenever it happens to reflect the nodes which are currently transmitting, receiving and those which are just being covered by the transmitting nodes. Notice that the decision to choose node id^{tx} to make the transmission is made by the function ($followPolicy$) as will be presented. Also notice that, the matrix $\mathbf{A}$ is set initially to the empty matrix $\emptyset$ in the same function.

The inputs to the function ($transmitData$) include:

- the node statuses matrix $\mathbf{STAT}^n$ before transmitting the data. Notice that $\mathbf{STAT}^n$ is set initially to the zero matrix $\mathbf{0}^{n_n^b \times 3}$ as will be shown in the function ($followPolicy$),
- the refined indices vector $\mathbf{id}_{ne}^{tx}$ of neighboring nodes around the transmitting node id^{tx} and their distances vector $\mathbf{d}_{ne}^{tx}$ from id^{tx}. These two vectors have already been evaluated using the function ($controlRange$),
- the matrix $\mathbf{SA}$ of segment allocations as evaluated by the function ($allocateData$), and
- the indices vector $\mathbf{id}_{ts}$ of targeted segments as computed using the function ($targetSegments$).

At the beginning, the function ($transmitData$) sets $\mathbf{stat}_1^n[id^{tx}]$ to 1 to indicate that id^{tx} is currently transmitting. After that, the popularities vector $\mathbf{pop}_{segs}$ of all data segments at the neighboring nodes, as indicated by $\mathbf{id}_{ne}^{tx}$, is computed as follows:

$$\mathbf{pop}_{segs} = \sum_{i \in \{\mathbf{id}_{ne}^{tx} : \mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}] = \mathbf{0}\}} \mathbf{SA}[i, *]$$

Notice that only those neighboring nodes which are not covered (i.e. $\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}] = \mathbf{0}$) are included in the vector summation above. This is given the fact that covered nodes are not going to receive the data anyway due to interference and therefore are not part of the segment popularities computation.

Given $\mathbf{pop}_{segs}$, the segment with the lowest popularity is chosen from the set of targeted segments for transmission as follows:

$$id_{cs} = \mathbf{id}_{ts}[\underset{i \in \mathbf{id}_{ts}}{argmin}(\mathbf{pop}_{segs}[i])]$$

where id_{cs} is the index of this chosen segment and the indices vector $\mathbf{id}_{ts}$ is for the targeted segments. The function ($argmin$) is used here to find the index i of that segment with the lowest popularity such that $i \in \mathbf{id}_{ts}$. Notice that the intuition here is that the segment with the lowest popularity should be chosen because it is the one which is mostly missing among neighbors and therefore is the most probably needed.

The function ($transmitData$) proceeds by going over all the neighboring nodes, as indicated by $\mathbf{id}_{ne}^{tx}$, in order to detect those nodes which are not covered while missing the chosen data segment id_{cs}. These nodes are detected whenever a node $\mathbf{id}_{ne}^{tx}[j]$ meets the following condition:

$$(\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 0) \wedge (\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], id_{cs}] = 0)$$

These nodes receive the segment id_{cs} from the transmitting node id^{tx} and therefore their statuses are updated to indicate that they are receiving and therefore are covered as follows:

$$\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 1$$

$$\mathbf{stat}_3^n[\mathbf{id}_{ne}^{tx}[j]] = 1$$

Their segment allocations are also updated as follows:

$$\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], id_{cs}] = 1$$

This transmission action is added to the actions matrix $\mathbf{A}$ in two steps as follows:

$$\mathbf{A} \leftarrow (\mathbf{A}, [(t + t^{start} - 1) \;\; id^{tx} \;\; max(\mathbf{d}_{ne}^{tx}) \;\; id_{cs} \;\; \mathbf{0}^{1 \times n_n^b}])$$

$$\mathbf{A}[length(\mathbf{A}), 4 + \mathbf{id}_{ne}^{tx}[j]] = 1$$

where the columns of matrix $\mathbf{A}$ are defined as follows:

- $\mathbf{A}[*, 1]$ represents the time indices at which actions take place where the time step t is shifted by $(t^{start} - 1)$ to add the starting time of the period p under consideration,
- $\mathbf{A}[*, 2]$ represents the indices of the transmitting nodes taking the actions,
- $\mathbf{A}[*, 3]$ represents the broadcasting ranges of the transmitting nodes such that no interference occurs with neighboring transmitting nodes. This is done by choosing the broadcasting range of id^{tx} to equal $max(\mathbf{d}_{ne}^{tx})$,
- $\mathbf{A}[*, 4]$ represents the indices of the chosen segments for transmission, and
- $\mathbf{A}[*, 4 + 1 : 4 + n_n^b]$ indicates the neighboring nodes which are receiving the chosen content data segments from the transmitting nodes. This is made by setting the value of $\mathbf{A}[i, 4 + j] = 1$ if the neighboring node j is the one receiving the chosen data segment when the action i takes place and $\mathbf{A}[i, 4 + j] = 0$, otherwise.

As it can be seen, the update of matrix $\mathbf{A}$ is made in two steps so that the first step adds all the information pertaining to the action taking place except for which nodes are receiving the data. In the second step, the node receiving the chosen data segment is indicated using ones and zeros as explained.

After going over all the neighboring nodes, as indicated by $\mathbf{id}_{ne}^{tx}$, the function $(transmitData)$ updates the statuses of all of these nodes to reflect the fact that they are currently covered by the broadcasting range of id^{tx} as follows:

$$\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}] = \mathbf{1}^{length(\mathbf{id}_{ne}^{tx}) \times 1}$$

6.2.7 *followPolicy* Function

Figure 6.1 shows the overall flowchart of the function $(followPolicy)$. This function produces the final matrix $\mathbf{A}$ of actions which should be taken by the n_n^b nodes within the c^{th} biggest nodes cluster throughout the period p starting at t^{start} and ending at t^{end}. These actions are decided such that these nodes follow a given policy which is specified by the weights vector ω given to the different node features. Based on these weights and features, node utilities are computed and used to decide the order of transmission given to the nodes such that they do not interfere with each other. Notice that this order is made to prioritize node transmissions such as every higher priority node transmits only if it has segments to transmit, otherwise the lower priority node proceeds with its transmission. Inputs to the function $(followPolicy)$ include:

- the member indices matrix $\mathbf{ID}_{nc}^b\{c\}$ of the c^{th} biggest nodes cluster, as computed in Chapter 4, and their connectivities matrix $\mathbf{C}_b\{c\}$,
- the segment allocations matrix $\mathbf{SA}$ as evaluated by the function $(allocateData)$,
- the vector $\mathbf{n}_{ns}$ representing the number of content data segments at each node which are divided using the function $(divideData)$,

- the latitudes matrix **LATS** as computed in Chapter 4, and
- the longitudes matrix **LONS** as computed in Chapter 4.

The function ($followPolicy$) starts by initializing the actions matrix $\mathbf{A}$ to the empty matrix $\emptyset$ and computing $n_{T_{period}}$ given t^{start} and t^{end}. After that, it goes over all time steps while initializing the node statuses $\mathbf{STAT}^n$ to the zero matrix $\mathbf{0}^{n_n^b \times 3}$ at the beginning of each time step. However, the current coverage status of any node within the current cluster that is within the broadcasting range of a currently-transmitting node at a nearby cluster should not be zeroed due to this initialization of matrix $\mathbf{STAT}^n$; in fact, it should be activated such that no useless transmissions are ought to target this covered node. The features matrix $\mathbf{F}^n$ is also computed at each time step t using the function ($computeFeatures$). Given $\mathbf{F}^n$ and the given policy weights vector ω, the vector $\mathbf{ut}$ of node utilities is computed as follows:

$$\mathbf{ut} = \mathbf{F}^n \times \omega$$

where $\omega[i]$ is the weight given to feature i and $\omega[i] \in \{-10, -9, ..., 0, ..., 9, 10\} \forall i$ in the case study adopted. Based on $\mathbf{ut}$, the vector $\mathbf{id}_{tx}^{ord}$ of node indices, sorted in the order of transmission, is computed as follows:

$$[\sim, \mathbf{id}_{tx}^{ord}] = sort(\mathbf{ut}, -1)$$

where the function ($sort$) is used to sort the nodes in a descending order in terms of their utility values. Notice that the vector $\mathbf{id}_{tx}^{ord}$ specifies the order of transmission or alternatively the priority of transmission given to the nodes whenever they have data to transmit, otherwise the next priority node proceeds with the transmission.

Given $\mathbf{id}_{tx}^{ord}$, the function ($followPolicy$) goes over the nodes while taking the order of transmission specified by $\mathbf{id}_{tx}^{ord}$ into account as follows:

$$id^{tx} = \mathbf{id}_{tx}^{ord}[i]$$

where the index i allows the function to go over the nodes specified by the vector $\mathbf{id}_{tx}^{ord}$ according to their transmission order and setting id^{tx} accordingly. Therefore, $\mathbf{id}_{tx}^{ord}[1]$ produces the index of the node given the first priority to start transmission.

Given node id^{tx}, the function checks the status $\mathbf{stat}_2^n[id^{tx}]$ to know if this node is currently covered by the broadcasting range of another node. If node id^{tx} is covered (i.e. $\mathbf{stat}_2^n[id^{tx}] = 1$), then it can not transmit and no action can be taken. However, if it is not covered, then the vector $\mathbf{id}_{ne}^{tx}$ is computed to know if there are any neighboring nodes using the following formula:

$$\mathbf{id}_{ne}^{tx} = \underset{j}{argfind}(\mathbf{C}_b\{c\}[t, id^{tx}, j] = 1)$$

where the function ($argfind$) is used to find any index j of a neighboring node that has a connectivity with node id^{tx} at time t such as $\mathbf{C}_b\{c\}[t, id^{tx}, j] = 1$. If all neighbors are covered (i.e. $\forall j \in \mathbf{id}_{ne}^{tx}, \mathbf{stat}_2^n[j] = 1$) or there are no neighbors (i.e. $\mathbf{id}_{ne}^{tx} = \emptyset$),

then there is no point of having node id^{tx} transmitting data and no action would be taken. However, if there is at least one uncovered neighbor, then the broadcasting range of id^{tx} is controlled so that it does not cause any interference with neighboring transmitting nodes using the function ($controlRange$). This results in the refined vector of neighboring nodes indices $\mathbf{id}_{ne}^{tx}$ and their distances vector $\mathbf{d}_{ne}^{tx}$. However, this can also result in a shrinkage in the broadcasting range of id^{tx} that would left no neighbors (i.e. $\mathbf{id}_{ne}^{tx} = \emptyset$). If this is the case, then node id^{tx} would have no neighbors left and no action would be taken. If this is not the case, then the indices vector $\mathbf{id}_{ts}$ of targeted segments is computed using the function ($targetSegments$). This vector $\mathbf{id}_{ts}$ can be empty due to the mismatch between what is already available at id^{tx} and what is missing at the neighboring nodes or due to the segment sufficiency at neighboring nodes. If this is the case, then node id^{tx} would not need to transmit and no action would be taken. If this is not the case, then node id^{tx} would transmit the chosen segment id_{cs} using the function ($transmitData$). The outputs of this ($transmitData$) function are the updated matrix $\mathbf{A}$, with the new action taken by node id^{tx}, and the updated matrix $\mathbf{STAT}^n$ showing the current node statuses which reflect the set of nodes transmitting, receiving and just covered. With these new updated matrices, the "for-loop" going over the nodes in the function ($followPolicy$) proceeds to the next node until all n_n^b nodes are checked after which the "for-loop" going over the time indices moves to the next time index. These iterations result eventually in the final matrix $\mathbf{A}$ of actions which should be taken by the nodes in order to follow the policy specified by the feature weights vector ω.

6.3 Maximum Number of Data Segments Estimation

Before dividing and allocating content data segments or beginning the search for the optimal V2V segment exchange policy, the maximum number of all data segments n_{as}^{max} is estimated. This number represents the maximum number of segments which can be distributed among the system nodes at the beginning of the content distribution cycle using V2I communication such that a minimum exchange ratio threshold er_{min} is met. This threshold is defined as the minimum ratio between the number of exchanged data segments using V2V communication and the number of data segments distributed initially using V2I communication. Meeting this threshold is essential in order to avoid distributing an excessively high number of data segments at the beginning of the content distribution cycle using V2I communication which can not be replicated sufficiently using V2V segment exchanges between the nodes. Not imposing this threshold would leave a high percentage of the data segments for the final distribution using V2I communication which should be avoided such that the network is not overloaded.

Therefore, and before estimating n_{as}^{max}, er_{min} is specified given the total number of nodes n_n^b. This is done by first making the assumption that one version of each data segment is initially distributed among the nodes. This single version would then be replicated using segment exchanges between the nodes such that each node would end

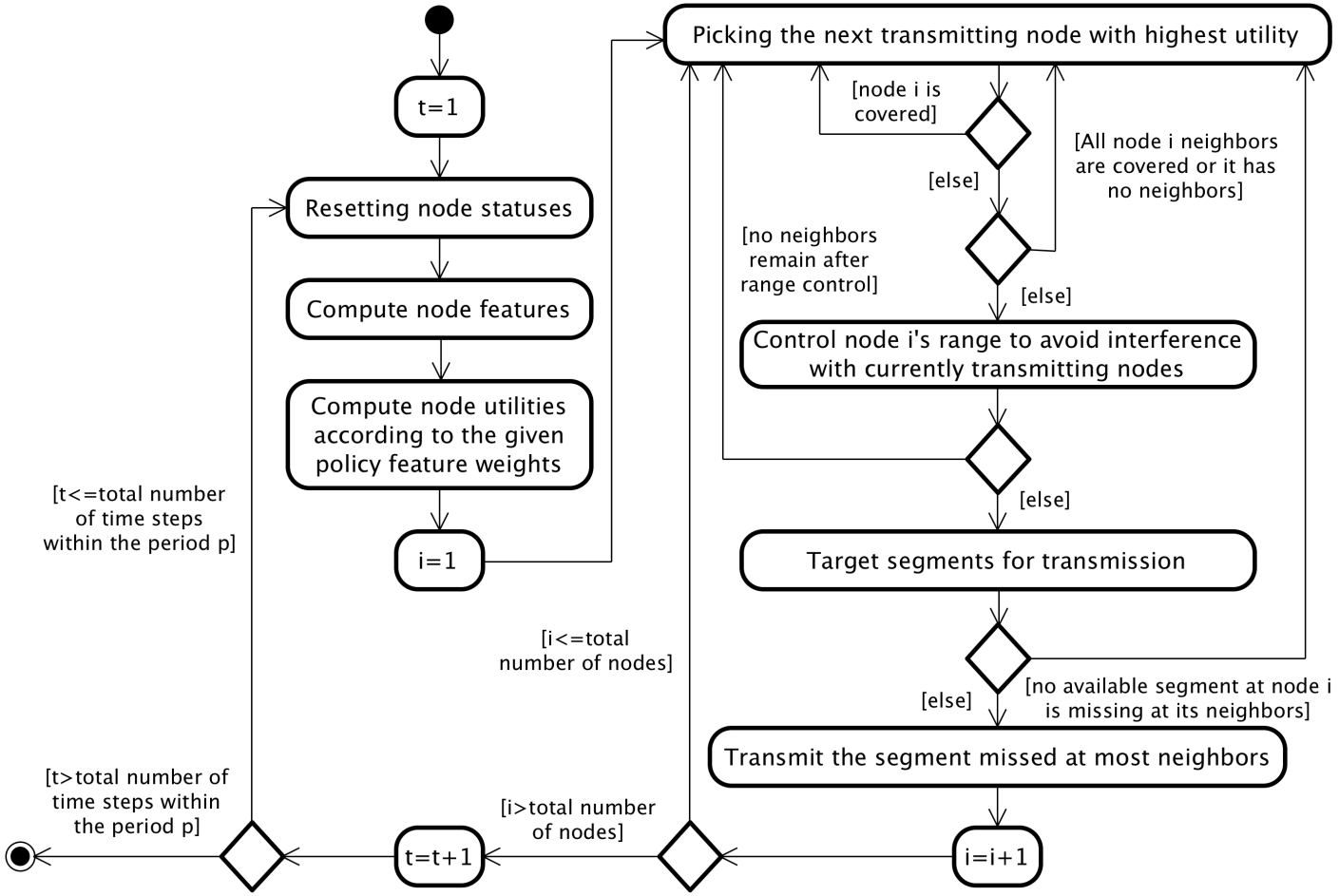


Figure 6.1: Flowchart of the function ($followPolicy$)

up having one copy of each segment with no duplicates at the end of these exchanges. Given this setup, the value of er_{min} can now be specified such that it satisfies the following formula:

$$er_{min} = \lceil 0.9 \times n_n^b - 1 \rceil$$

The intuition behind this formula is to assume that at least 90% of the data segments should be either distributed initially using V2I communication or exchanged between the nodes by the end of the segment exchanges using V2V communication. This leaves at maximum 10% of the segments for the final content distribution using V2I communication. These two distribution stages result eventually in $(1 + er_{min})$ number of segment versions distributed among the nodes by the end of V2V segment exchanges out of the n_n^b versions which should be distributed by the end of the whole distribution period. Given this equation and the fact that $n_n^b = 21$ nodes for the first biggest nodes cluster in the case study adopted, then $er_{min} = 18$.

Given the value of er_{min}, n_{as}^{max} can now be estimated by going over the vector $\mathbf{n}_{as}$ of the different numbers of all data segments. This vector $\mathbf{n}_{as}$ is assumed in the case study adopted to be as follows:

$$\mathbf{n}_{as} = [50 \;\; 100 \;\; ... \;\; 2000]$$

This scanning is done while computing the resulting ratio between the number of exchanged data segments and the number of data segments distributed initially. Therefore, each scan starts by computing the number of data segments for each node as specified by the vector $\mathbf{n}_{ns}$ using the function ($divideData$). Given $\mathbf{n}_{ns}$, the matrix of segment allocations $\mathbf{SA}$ is computed using the function ($allocateData$). Knowing $\mathbf{n}_{ns}$ and $\mathbf{SA}$, the actions matrix $\mathbf{A}$ can be computed using the function ($followPolicy$) assuming that the "naive" policy with the weights vector $\omega = (10, 0, 0, 0)$ is followed. This policy is called "naive" because it is simply giving the highest transmission priority to the node with the highest normalized number of direct neighbors.

By the end of each scan, the number of data segment exchanges is determined as equal to $length(\mathbf{A})$. After that, the ratio between this number and the number of data segments initially distributed is determined. This process repeats under the different numbers of all segments as specified by $\mathbf{n}_{as}$. With this and given the case study under consideration, the chart shown in Figure 6.2 is drawn for the first biggest nodes cluster in order to determine n_{as}^{max} to equal 150 segments given $er_{min} = 18$ as illustrated by the red dashed line. Notice that in Figure 6.2, the ratio between the number of exchanged data segments and the number of data segments distributed initially can not exceed $(n_n^b - 1)$ which means $(21 - 1) = 20$ for the first biggest nodes cluster under consideration. This is given the one version of data segments initially distributed.

Refer to Algorithm C.9 in Appendix C for further details about this operation of estimating the maximum number of data segments.

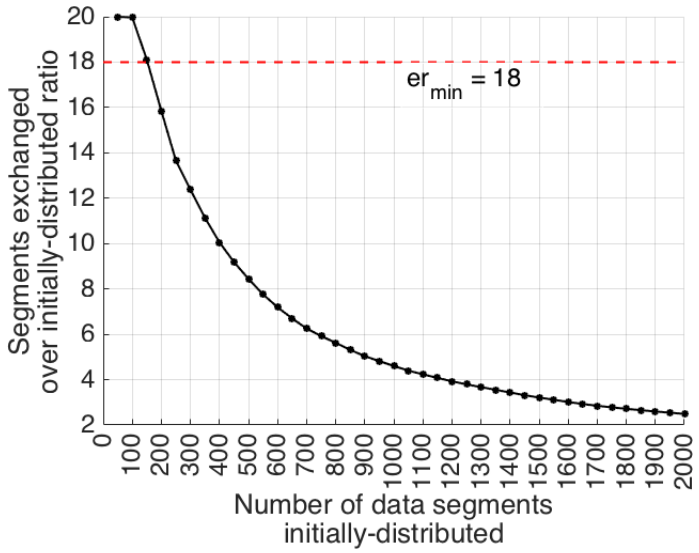


Figure 6.2: Number of data segments initially distributed effect

6.4 Data Segments Division and Allocation

Given n_{as}^{max}, which is found to be 150 segments for the first biggest nodes cluster in the case study adopted, the n_{as}^{max} data segments are divided and allocated to the n_n^b nodes. This process starts by using the function ($argfind$) to find the index k that picks the n_{as}^{max} out of the vector $\mathbf{n}_{as}$. This index k is then used as input to the function ($divideData$) and the resulting $\mathbf{n}_{ns}$ vector of node segment numbers is used afterwards as input to the function ($allocateData$) to compute the matrix **SA** of segment allocations. These allocations represent the segments distributed initially among the system nodes using V2I communication.

Notice that these aforementioned steps are already part of the upcoming bayesian optimization operations as will be shown. This is due to the fact that **SA** values are altered in the iterations of these operations and therefore need to be reset at the beginning of each iteration.

Figure 6.3 shows the resulting segment allocations where the value 1 in the heat map indicates a segment being allocated and the value 0 indicates, otherwise. These allocations are made across the n_n^b nodes of the first biggest cluster of the adopted case study which is 21 nodes. Notice that the number of data segments allocated is about 140 segments which is less than n_{as}^{max} of 150 segments. This is due to the use of the function ($divideData$) which ensures that the smallest unit of data allocated is one segment.

Refer to Algorithm C.10 in Appendix C for further details about this operation of dividing and allocating the data segments.

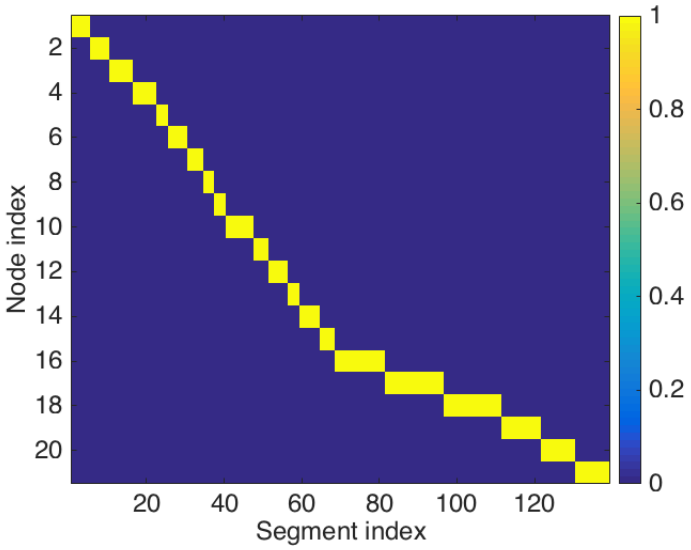


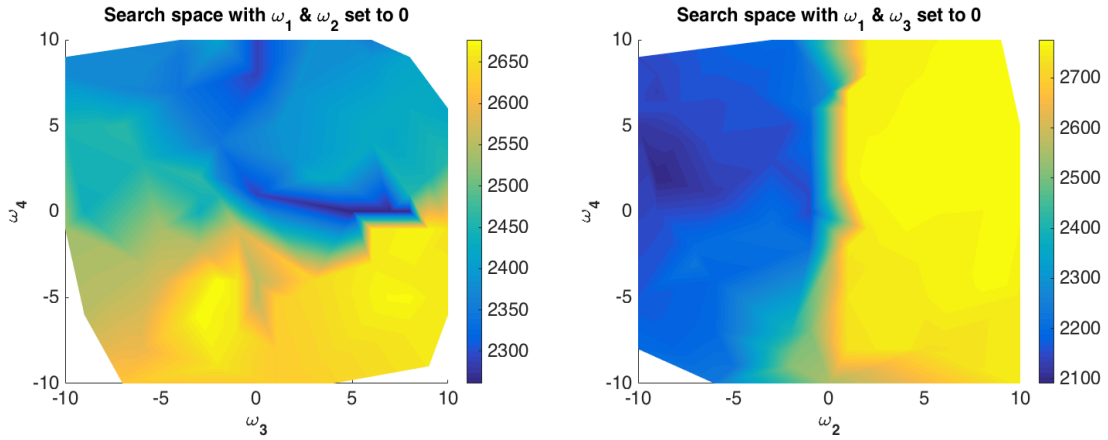
Figure 6.3: Initial data segment allocations

6.5 Exchange Policy Search Space Visualization

Before optimizing the V2V segment exchange policy, the search space for the policy weights is first visualized. This is done by producing a visualization data matrix that generates $n_i = 100$ random points for each search space slice of the 6 slices used such that in each point: n_{as}^{max} segments are allocated between the nodes, a policy weights vector is generated randomly while adhering to the corresponding slice weight restrictions and a matrix $\mathbf{A}$ of actions is computed using the function $(followPolicy)$. The resulting number of data segment exchanges $length(\mathbf{A})$ as well as the randomly-chosen policy weights vector ω are then used for the slice visualizations as shown in Figure 6.4. The weight restrictions used for the 6 slices are as follows:

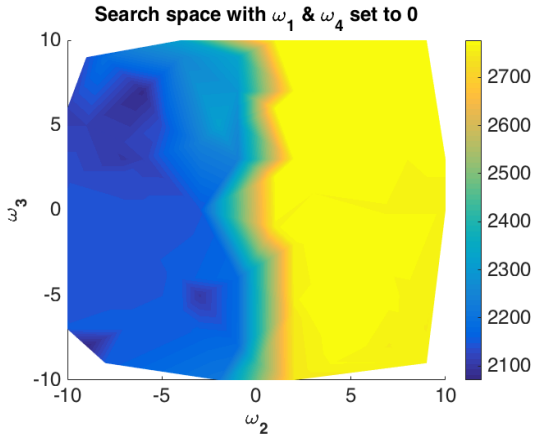
1. Slice 1; where weights vary randomly except for setting $\omega[1]$ & $\omega[2]$ to 0,
2. Slice 2; where weights vary randomly except for setting $\omega[1]$ & $\omega[3]$ to 0,
3. Slice 3; where weights vary randomly except for setting $\omega[1]$ & $\omega[4]$ to 0,
4. Slice 4; where weights vary randomly except for setting $\omega[2]$ & $\omega[3]$ to 0,
5. Slice 5; where weights vary randomly except for setting $\omega[2]$ & $\omega[4]$ to 0, and
6. Slice 6; where weights vary randomly except for setting $\omega[3]$ & $\omega[4]$ to 0.

As it can be seen, Figure 6.4 shows that the optimal weight values are not trivial to find given the many local minima and maxima encountered.

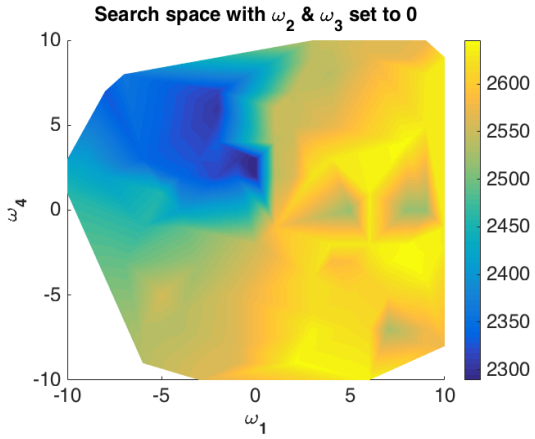


(a) Slice 1

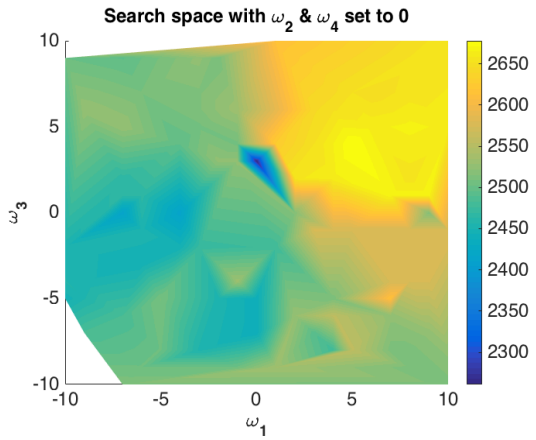
(b) Slice 2



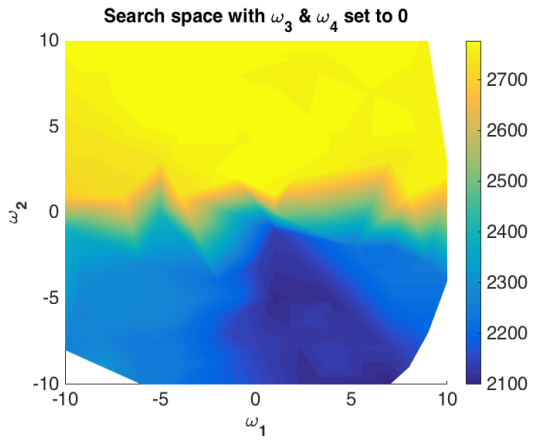
(c) Slice 3



(d) Slice 4



(e) Slice 5



(f) Slice 6

Figure 6.4: Exchange policy search space visualizations

6.6 Exchange Policy Bayesian Optimization

Figure 6.5 shows the flowchart of bayesian optimization. In each iteration i, a model is fit to a set of regression data followed by using it to produce random samples. Assuming a UCB acquisition function, the "best" random policy that has the highest UCB estimate is chosen to experiment with by first dividing and allocating the n_{as}^{max} data segments and then using the function ($followPolicy$) discussed previously. The result of following this "best" policy is added to the regression data to have better estimates in the next iterations until the bayesian optimization algorithm converges to the actual optimal policy.

As it can be seen from Figure 6.5, an initial set of regression data has to be generated before searching for the optimal exchange policy using bayesian optimization. Therefore, this section starts by doing so after which bayesian optimizations under different regression techniques are compared. These techniques are: GP, RF, BNN and batch-based RF. The outcome of this comparison is a bayesian optimization technique that is based on batch-based RF regression which utilizes a batch of the data only instead of the whole data as will be presented.

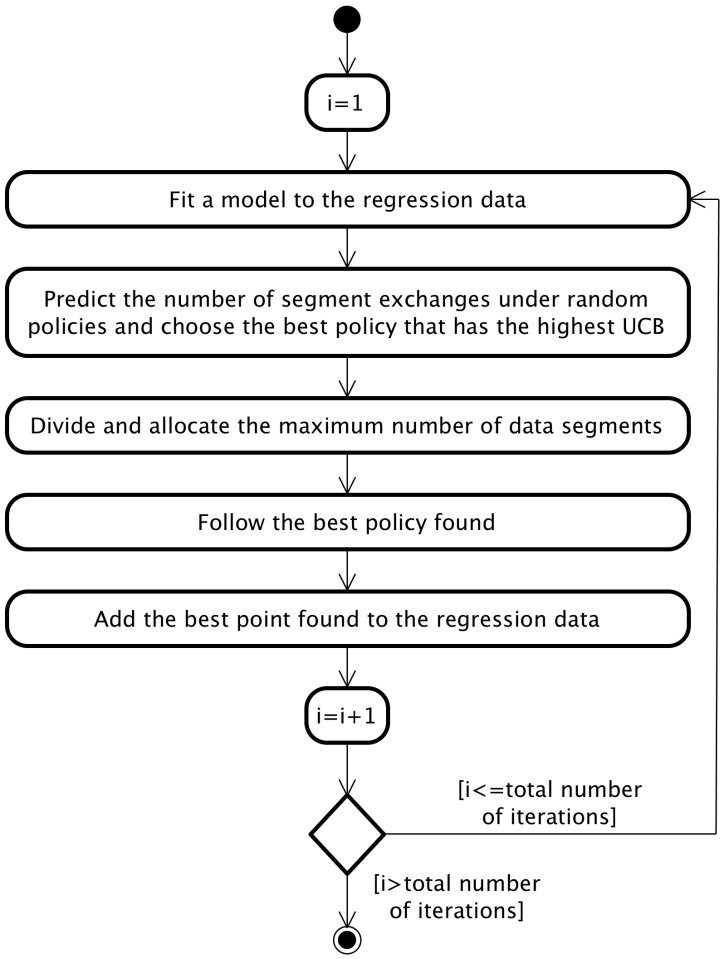


Figure 6.5: Flowchart of bayesian optimization

6.6.1 Initial Regression Data Generation

Before starting bayesian optimization under any regression technique, the initial regression data matrix $\mathbf{DATA}_{reg}$ should be generated. This matrix is used to build the regression models used in the bayesian optimization iterations when drawing random samples.

This data generation process starts by setting $\mathbf{DATA}_{reg}$ to the zero matrix $\mathbf{0}^{n_i \times (4+1)}$. The number of iterations n_i used to generate the initial regression data points is set to 100. In each iteration, the index k that gives n_{as}^{max} is computed followed by the vector $\mathbf{n}_{ns}$ of node segments and the matrix $\mathbf{SA}$ of segment allocations. After that, the policy weights vector ω is drawn at random using the function $(randi)$. The resulting actions matrix $\mathbf{A}$ is computed and a new data point is added to the matrix $\mathbf{DATA}_{reg}$ where the columns of indices $1 : (length(\mathbf{DATA}'_{reg}) - 1)$ represent the four $\omega[1:4]$ policy feature weights and the column of index $length(\mathbf{DATA}'_{reg})$ represents the corresponding number of segment exchanges computed as $length(\mathbf{A})$.

Algorithm C.11 in Appendix C gives further details about this operation of generating the initial regression data. Appendix C does also include Algorithms C.12 to C.15 giving further details about the upcoming bayesian optimizations under different regression techniques.

6.6.2 Regression Using Gaussian Processes

The first regression technique used for bayesian optimization is GP. This optimization process starts by creating the GP data matrix $\mathbf{DATA}_{gp}$ and setting it to the zero matrix $\mathbf{0}^{(length(\mathbf{DATA}_{reg})+n_i)\times(4+1)}$ where n_i is the number of iterations used in the bayesian optimization and it is set to 900. The first $length(\mathbf{DATA}_{reg})$ rows of this matrix are assigned the values of the initial regression data matrix $\mathbf{DATA}_{reg}$ as follows:

$$\mathbf{DATA}_{gp}[1 : length(\mathbf{DATA}_{reg}), *] = \mathbf{DATA}_{reg}$$

The process proceeds afterwards with the bayesian optimization iterations. In each iteration, the GP regression model $\mathbf{Mdl}_{gp}$ is fit first with the weights columns $\mathbf{DATA}_{gp}[*, 1 : 4]$ used as the input data and the column of the corresponding number of segment exchanges $\mathbf{DATA}_{gp}[*, 4+1]$ as the output data. This is done using the GP regression fitting function $(fitgp)$ as follows:

$$\mathbf{Mdl}_{gp} = fitgp(\mathbf{DATA}_{gp}[*, 1:4], \mathbf{DATA}_{gp}[*, 4+1])$$

The optimization process continues by inquiring the generated regression model $\mathbf{Mdl}_{gp}$ randomly for a total number of n_{rnd} times while searching for the optimal policy weights vector. The value of n_{rnd} is set to 1000 given the cheap cost of inquiring the model $\mathbf{Mdl}_{gp}$ compared to the actual evaluation cost. In order to store the results of these random inquires, the data matrix $\mathbf{DATA}_{rnd}$ is created and set initially to the zero matrix $\mathbf{0}^{n_{rnd}\times(4+1)}$ before proceeding into the random inquiry iterations. In each one of these iterations, the optimization process starts by

generating a random policy weights vector using the function ($randi$). This random weights vector is, in turn, stored into $\mathbf{DATA}_{rnd}[j, 1 : 4]$ before inquiring the model $\mathbf{Mdl}_{gp}$ using the function ($predict$) as follows:

$$[\hat{\mathbf{DATA}}_{rnd}, \sigma] = predict(\mathbf{Mdl}_{gp}, \mathbf{DATA}_{rnd}[j, 1 : 4])$$

where:

- j is the current random point iteration index,
- $\hat{\mathbf{DATA}}_{rnd}$ is the estimated number of segment exchanges given the random weights vector $\mathbf{DATA}_{rnd}[j, 1 : 4]$, and
- σ is the standard deviation of the estimated number of segment exchanges $\hat{\mathbf{DATA}}_{rnd}$.

Given both $\hat{\mathbf{DATA}}_{rnd}$ and σ, the estimated UCB for the number of segment exchanges can be derived and stored in $\mathbf{DATA}_{rnd}[j, 4 + 1]$ as follows:

$$\mathbf{DATA}_{rnd}[j, 4 + 1] = \parallel \hat{\mathbf{DATA}}_{rnd} + \sigma \parallel$$

After generating all the n_{rnd} random inquiries, the optimization process finds the vector $\mathbf{data}_{rnd}$ of the current-best policy weights and their corresponding UCB of segment exchanges using the function ($argmax$) as follows:

$$\mathbf{data}_{rnd} = \mathbf{DATA}_{rnd}[\underset{j}{argmax}(\mathbf{DATA}_{rnd}[j, 4 + 1]), *]$$

where the function ($argmax$) is used to find the index j that gives the random point with the highest UCB of segment exchanges. Given $\mathbf{data}_{rnd}$, the process proceeds with evaluating the index k yielding n_{as}^{max}, the vector $\mathbf{n}_{ns}$ and the matrix $\mathbf{SA}$. The matrix of actions $\mathbf{A}$ is evaluated afterwards given the current-best weights vector ω defined as follows:

$$\omega' = \mathbf{data}_{rnd}[1, 1 : 4]$$

and a new point is added to the matrix $\mathbf{DATA}_{gp}$ as follows:

$$\mathbf{DATA}_{gp}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \ \ length(\mathbf{A})]$$

where i is the current bayesian optimization iteration index. These bayesian optimization iterations continue until convergence to the optimal policy.

6.6.3 Regression Using Random Forest

Bayesian optimization using RF regression is mostly similar to the previous bayesian optimization using GP regression. However, there are some differences. The first difference is obviously using RF regression instead of GP regression. This is done by first building the forest assumed to be based on the 10 tree regression models $\mathbf{Mdl}_{tr}\{1 : 10\}$. These models are generated using the function ($fittree$) as follows:

$$\mathbf{Mdl}_{tr}\{j\} = fittree(\mathbf{DATA}_{rf}[*, 1:4], \mathbf{DATA}_{rf}[*, 4+1], 'MinLeafSize', j)$$

where j is the current tree index and, similar to before, each tree regression model has the inputs data represented by the weights columns $\mathbf{DATA}_{rf}[*, 1:4]$ and the output data represented by the column of the corresponding number of segment exchanges $\mathbf{DATA}_{rf}[*, 4+1]$ where $\mathbf{DATA}_{rf}$ is the RF data matrix. Notice that the regression trees differ in their minimum leaf sizes which are set to the index j of each tree. This is made in order to have a random forest that would generalize well.

Given the tree models $\mathbf{Mdl}_{tr}\{1:10\}$, the optimization process continues with its n_{rnd} random points where n_{rnd} is set to 1000 as before. In each random point, all of the 10 trees are inquired using the function ($predict$) as follows:

$$\hat{\mathbf{DATA}}_{rnd}[k] = predict(\mathbf{Mdl}_{tr}\{k\}, \mathbf{DATA}_{rnd}[j, 1:4])$$

where:

- k is the current tree index,
- j is the current random point iteration index, and
- $\mathbf{DATA}_{rnd}[j, 1:4]$ is the random weights vector drawn using the function ($randi$) like before.

Given the tree estimations $\hat{\mathbf{DATA}}_{rnd}[1:10]$, their mean μ as well as their standard deviation σ can be derived as follows:

$$\mu = (\sum_{k} \hat{\mathbf{DATA}}_{rnd}[k])/10$$

$$\sigma = (\tfrac{1}{10} \times \sum_{k} (\hat{\mathbf{DATA}}_{rnd}[k])^2 - \mu^2)^{1/2}$$

Given μ and σ, the estimated UCB for the number of segment exchanges given the random weights vector is computed and stored in $\mathbf{DATA}_{rnd}[j, 4+1]$ as follows:

$$\mathbf{DATA}_{rnd}[j, 4+1] = \parallel \mu + \sigma \parallel$$

After generating all the n_{rnd} random inquiries, the optimization process continues like before using GP regression until a new point is added to the matrix $\mathbf{DATA}_{rf}$ as follows:

$$\mathbf{DATA}_{rf}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \ \ length(\mathbf{A})]$$

where i is the current bayesian optimization iteration index. These bayesian optimization iterations continue until convergence to the optimal policy. Notice that n_i is set to 900 iterations like before.

6.6.4 Regression Using Bayesian Neural Network

Bayesian optimization using BNN regression is again very similar to bayesian optimization using GP regression except for the main difference of using BNN regression instead of GP regression. Doing such a regression requires two steps to be taken. The first step is when the neural network model $\mathbf{Mdl}_{nn}$ is built using the function $(fitnn)$ as follows:

$$\mathbf{Mdl}_{nn} = fitnn([10, 10, 10])$$

where the number of hidden layers is assumed to be 3 and each layer is assumed to have 10 neurons. The activation functions for the different layers are specified such that the three hidden layers use the "Hyperbolic Tangent Sigmoid" transfer function whereas the output layer uses the "Linear" transfer function. The "Linear" transfer function is used at the output layer in order to act as a linear summation that will be replaced with a linear regression model in the second step.

After building the model $\mathbf{Mdl}_{nn}$, it is trained using the function $(train)$ as follows:

$$\mathbf{Mdl}_{nn} = train(\mathbf{Mdl}_{nn}, \mathbf{DATA}_{bnn}[*, 1:4], \mathbf{DATA}_{bnn}[*, 4+1])$$

where $\mathbf{DATA}_{bnn}$ is the BNN data matrix such that $\mathbf{DATA}_{bnn}[*, 1:4]$, representing the weight columns, is used as the input data, and $\mathbf{DATA}_{bnn}[*, 4+1]$, representing the column of the corresponding number of segment exchanges, is used as the output data.

In the second step, the first thing made is removing the output layer of the neural network model $\mathbf{Mdl}_{nn}$ such that the 3^{rd} hidden layer becomes disconnected from the output layer (i.e. 4^{th} layer) and the model output becomes directly available at the 3^{rd} hidden layer. This is followed by predicting the outputs of this updated model $\mathbf{Mdl}_{nn}$ given the same input data $\mathbf{DATA}_{bnn}[*, 1:4]$ using the function $(predict)$. The output of this updated model is then used as an input to the linear regression mode $\mathbf{Mdl}_{lin}$ being fit using the function $(fitlinear)$ and the output data $\mathbf{DATA}_{bnn}[*, 4+1]$ as follows:

$$\mathbf{Mdl}_{lin} = fitlinear(predict(\mathbf{Mdl}_{nn}, \mathbf{DATA}_{bnn}[*, 1:4]), \mathbf{DATA}_{bnn}[*, 4+1])$$

Given the trained models $\mathbf{Mdl}_{nn}$ and $\mathbf{Mdl}_{lin}$, the optimization process proceeds with the n_{rnd} random inquiries such as each estimation $\hat{\mathbf{DATA}}_{rnd}$ is computed as follows:

$$\hat{\mathbf{DATA}}_{rnd} = predict(\mathbf{Mdl}_{lin}, predict(\mathbf{Mdl}_{nn}, \mathbf{DATA}_{rnd}[j, 1:4]))$$

where:

- n_{rnd} is set to 1000 points like before,
- j is the current random point iteration index,
- $\mathbf{DATA}_{rnd}[j, 1:4]$ is the random weights vector drawn using the function $(randi)$ like before, and

- the function (*predict*) is used to predict the estimation $\hat{\mathbf{DATA}}_{rnd}$.

Given $\hat{\mathbf{DATA}}_{rnd}$, the estimated UCB for the number of segment exchanges given the random weights vector is computed and stored in $\mathbf{DATA}_{rnd}[j, 4+1]$ as follows:

$$\mathbf{DATA}_{rnd}[j, 4+1] = \| \hat{\mathbf{DATA}}_{rnd} + \mathbf{Mdl}_{lin}.rmse \|$$

where the method ($rmse$) gives the root mean square of the estimation when applied to $\mathbf{Mdl}_{lin}$. This is almost equivalent to the standard deviation used during previous bayesian optimizations.

After generating all the n_{rnd} random inquiries, the optimization process continues like before using GP regression until a new point is added to the matrix $\mathbf{DATA}_{bnn}$ as follows:

$$\mathbf{DATA}_{bnn}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \ \ length(\mathbf{A})]$$

where i is the current bayesian optimization iteration index. These bayesian optimization iterations continue until convergence to the optimal policy. Notice that n_i is set here to 900 iterations like before.

6.6.5 Regression Techniques Comparison

Figures 6.6a to 6.6c show the policy weights as they vary throughout bayesian optimization iterations under the different regression techniques. They have the smallest variation and converge faster under RF regression. They have the most extreme variation and converge much slower under BNN regression. Their variation under GP regression is in between that of RF regression and BNN regression. In all cases, the first 100 iterations represent the weights of the initial regression data generated randomly as discussed before in Subsection 6.6.1. After that, bayesian optimization takes effect for the next 900 iterations.

Overall, the number of segment exchanges increases as the bayesian optimization iterations progress as shown in Figure 6.7a. However, this number tends to fluctuate more under BNN regression. Notice again the random variation in this number for the first 100 iterations representing the initial regression data.

The execution time of the last 900 bayesian optimization iterations under the different regression techniques is shown in Figure 6.7b. The machine used to make these time measurements is a MacBook Pro with a 2.5 GHz Intel Core i5 processor and an 8 GB-1333 MHz DDR3 memory. These measurements represent the average execution time of 10 regression iterations in each bayesian optimization iteration. They are further cleaned from any outliers using an outlier removing window of size 10 and smoothed afterwards using a moving average window of size 10. Mean execution time measurements are used to replace the outliers and smooth the measurements within the window.

In general, the execution time increases as the number of observations increases which corresponds directly to the number of iterations after the first 100 bayesian optimization iterations. This is given the fact that exactly one observation is added to the regression data used in each iteration. This execution time cost varies

significantly depending on which regression technique is being used. In fact, it grows exponentially under GP regression, reaching as high as 3 seconds, while growing linearly under both RF and BNN regression techniques. Using RF regression has resulted in the shortest execution times with the slowest rate of increase as the number of observations increases.

With the above observations in mind, the next subsection proceeds with bayesian optimization using the RF regression technique given the short execution time it has and the fact that it reaches the optimal number of segment exchanges as well as policy weights with fewer fluctuations compared to the other regression techniques. However, the most recent batch of regression data will only be used during each bayesian optimization iteration instead of using the full data size. In addition, the randomly-generated weights will be restricted at each iteration such that only "interesting" weight values are chosen. The motivation for this approach is to make the execution time shorter by processing less data while not missing the optimal weight values. This is driven by the fact that policy weights tend to converge to their optimal value as iterations progress. Such a value can be found in the weight search space represented by the most recent batch of regression data. This fact can be seen by observing the policy weight variations shown in Figures 6.6a to 6.6c as the number of iterations increases and comparing that to the resulting number of segment exchanges as shown in Figure 6.7a for the different regression techniques.

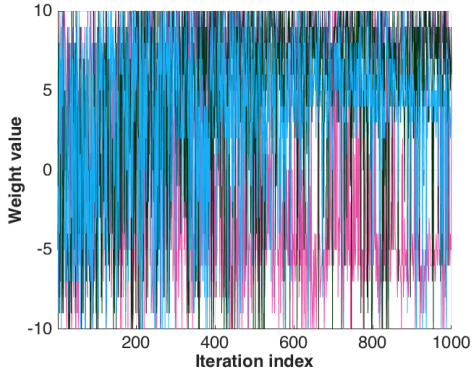
6.6.6 Regression Using Batch-based Random Forest

Given the previous comparison, this subsection proceeds with bayesian optimization using batch-based RF regression. This optimization process is very similar to the bayesian optimization using RF regression except for the amount of data being used in each optimization iteration. In this optimization process, it is assumed that only the most recent 100 points of the batch-based RF data matrix $\mathbf{DATA}_{brf}$ are used for the RF regression made in each bayesian optimization iteration. The function $(fittree)$ is used to build the regression models $\mathbf{Mdl}_{btr}\{1:10\}$ for the 10 batch-based trees used to construct the random forest. In each one of these models, the matrix $\mathbf{DATA}_{brf}[i:i+99, 1:4]$, representing the weight columns of the most recent 100 points, is used as an input data to each model and the matrix $\mathbf{DATA}_{brf}[i:i+99, 4+1]$, representing the column of the corresponding number of segment exchanges, is used as the output data where i is the current bayesian optimization iteration index.

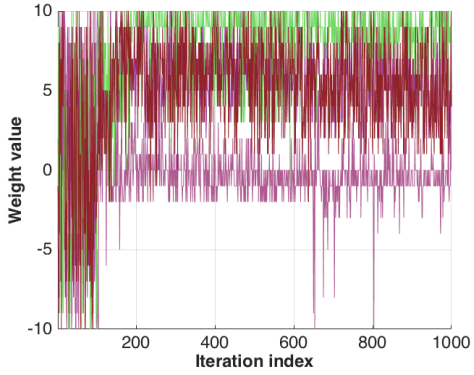
Another difference between this optimization process and the previous bayesian optimization using RF regression is the fact that the random weights vector ω is drawn at random using the function $(randi)$ such that each $\omega[k] : k \in \{1, 2, ..., 4\}$ falls between the minimum weight bound ω_{min} and the maximum weight bound ω_{max}. These bounds are derived as follows:

$$\omega_{min} = \parallel max(\mu_\omega - \sigma_\omega, -10) \parallel$$

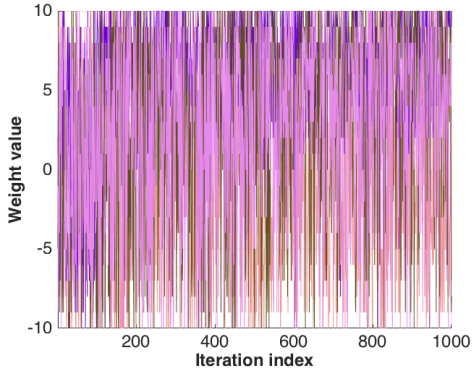
$$\omega_{max} = \parallel min(\mu_\omega + \sigma_\omega, 10) \parallel$$



(a) Under GP regression

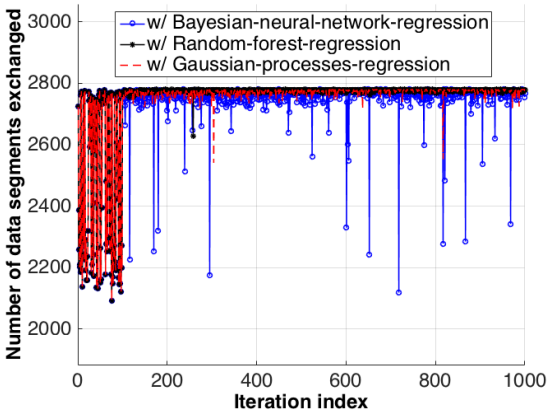


(b) Under RF regression

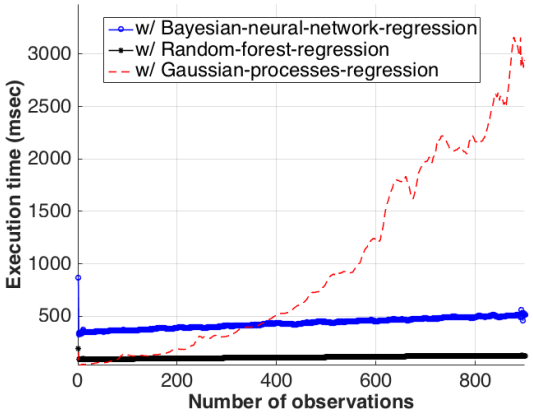


(c) Under BNN regression

Figure 6.6: Policy weights variation



(a) Optimization performances



(b) Optimization execution times

Figure 6.7: Bayesian optimization under GP, RF and BNN regressions

where:

- μ_ω is the policy weights mean within the most recent 100-point data batch and it is computed as follows:

$$\mu_\omega = (\sum_{m=i}^{i+99} \mathbf{DATA}_{brf}[m,k])/100$$

- and σ_ω is the policy weights standard deviation within the most recent 100-point data batch and it is computed given μ_ω as follows:

$$\sigma_\omega = ((\sum_{m=i}^{i+99} (\mathbf{DATA}_{brf}[m,k] - \mu_\omega)^2)/100)^{1/2}$$

Notice that m is the batch point index, i is the current bayesian optimization iteration index and k is the weight index. Also notice that ω_{min} can not be less than -10 and ω_{max} can not be more than 10 according to the adopted case study.

As explained previously, choosing $\omega[k] : k \in \{1, 2, ..., 4\}$ between these ω_{min} and ω_{max} values bound the interesting search region in which the optimal ω can be found. This is given the improvement demonstrated by bayesian optimizations as iterations progress.

After generating all the $n_{rnd} = 1000$ random inquiries, the optimization process continues in the same exact way like before using RF regression until a new point is added to the matrix $\mathbf{DATA}_{brf}$ as follows:

$$\mathbf{DATA}_{brf}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \;\; length(\mathbf{A})]$$

where i is the current bayesian optimization iteration index. These bayesian optimization iterations continue until convergence to the optimal policy. Notice that n_i is set here to 900 iterations like before.

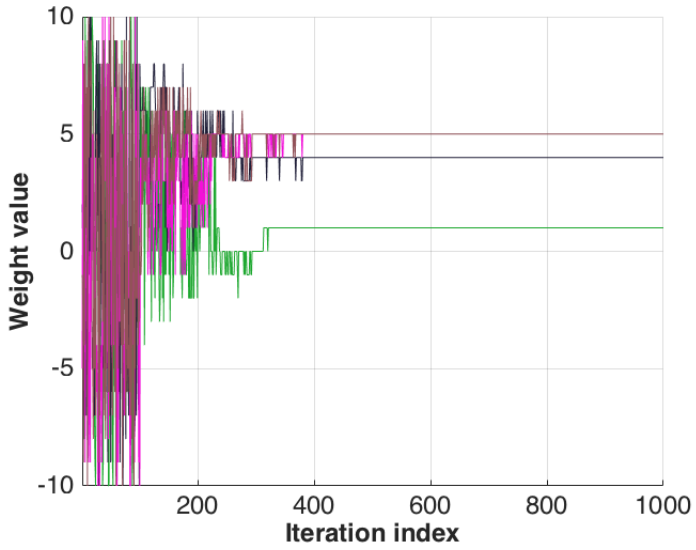


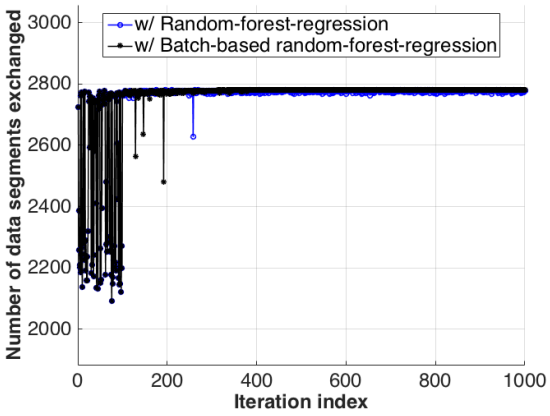
Figure 6.8: Policy weights variation under batch-based RF regression

Figure 6.8 shows the policy weight variation as iterations progress. Compared to Figure 6.6b, it can be seen that the variation is much smaller and the policy weights eventually converge to the fixed optimal values. This convergence is also demonstrated by the number of segment exchanges shown in Figure 6.9a. Compared to the conventional RF regression using the full data size, the batch-based RF regression has managed to converge faster with fewer fluctuations.

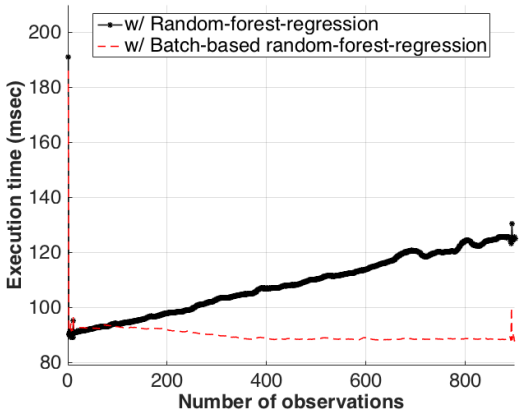
In terms of the execution time, the batch-based RF regression has a time cost that does not increase with the number of observations as shown in Figure 6.9b. This is mainly attributed to the fact that the size of the regression data is actually fixed and does not grow as iterations progress. This is contrary to the case of the RF regression technique discussed previously which uses the full size of the continuously-growing data.

6.7 Routing Policy Results

Using bayesian optimization with batch-based RF regression, the optimal policy weight values are determined. Figure 6.10 shows snapshots of the resulting segment exchanges made between the nodes of the biggest cluster under consideration at the peak hour of the day (i.e. 5:00 PM). While Figure 6.10a provides a snapshot of the whole Region of Waterloo, Figure 6.10b provides a snapshot that zooms into the downtown area of Kitchener, ON which is also part of the Region. The red circles show the interference-free broadcasting ranges of the corresponding transmitting nodes. These nodes are represented by the red dots whereas the receiving nodes are represented by the green dots. The red lines between the red and green dots highlight the fact that nodes are exchanging data segments. However, the black dots represent inactive nodes which are neither transmitting nor receiving.

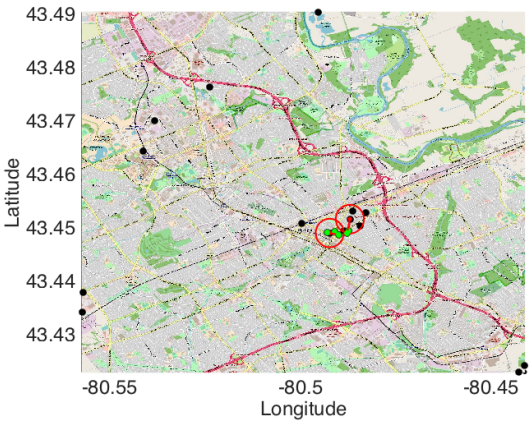


(a) Optimization performance

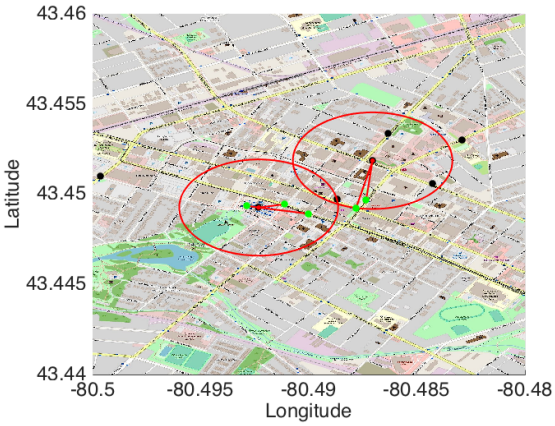


(b) Optimization execution time

Figure 6.9: Bayesian optimization under batch-based RF regression

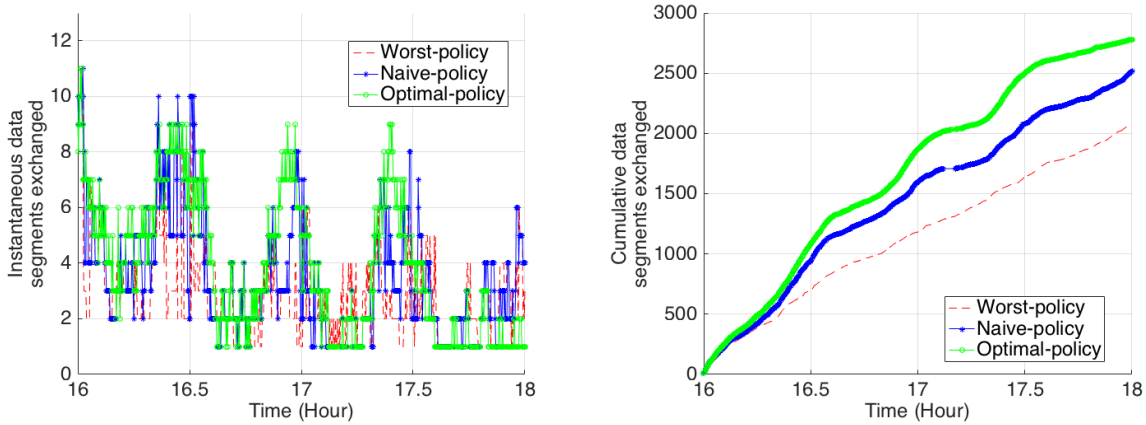


(a) Regular snapshot



(b) Zoomed-in snapshot

Figure 6.10: Data exchanges under optimal policy at 5:00 PM



(a) Instantaneous exchanges

(b) Cumulative exchanges

Figure 6.11: Data segments exchanged vs. time

Figure 6.11a shows the instantaneous number of data segments exchanged between the nodes of the biggest cluster under consideration. On the other hand, Figure 6.11b shows the cumulative number of data segments exchanged. Both figures do so under:

- the worst policy found so far during the bayesian optimization iterations,
- the naive policy which naively prioritizes the nodes with the highest number of direct neighbors, and
- the optimal policy found using bayesian optimization with batch-based RF regression.

The period under consideration is between 4:00 PM and 6:00 PM which is the most busy period of the day given the adopted case study.

Figure 6.12 shows the segment allocations at the end of data exchanges within the period under consideration (i.e. at 6:00 PM) for the worst, naive and optimal policies. This is for the biggest cluster under consideration which has $n_n^b = 21$ nodes. Notice that the number of data segments initially distributed is about 140 segments given that $n_{as}^{max} = 150$ segments as determined before. Also notice that the initial segment allocations has also been determined and shown before in Figure 6.3.

Figure 6.12 does also show the corresponding percentages of categories which segments belong to under the different policies. As discussed before, these segments can be categorized as "Initially-distributed" at the beginning of the content distribution period using V2I communication, "Exchanged" using V2V communication or "Finally-distributed" at the end of the distribution period using V2I communication.

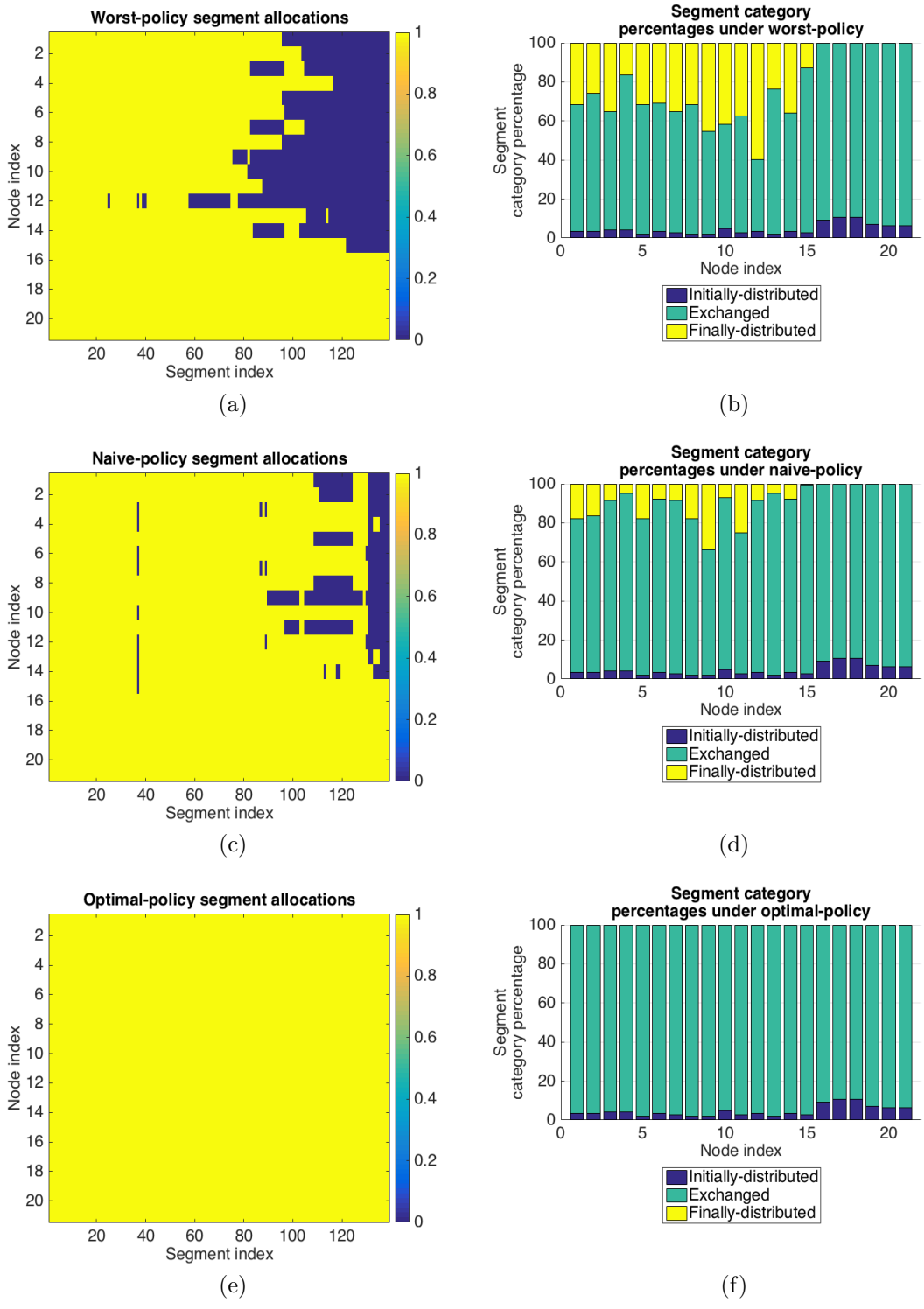


Figure 6.12: Data segments by the end of exchanges

6.8 Discussion

The optimal policy has demonstrated a superior performance compared to the other policies. As shown by the instantaneous segment exchanges, the optimal policy might refrain from exchanging data at certain times while achieving a higher cumulative number of segment exchanges in the long run compared to the other policies. In fact, it has managed to exchange more than 90% of the segments and left non for the final V2I distribution taking place at the end of the content distribution period. On the other hand, the naive policy has managed to exchange about 80% of the segments while leaving almost 10% for the final V2I distribution. Even worse, the worst policy has managed to exchange about 70% of the segments while leaving approximately 20% for the final V2I distribution. Notice that less than 10% of the segments have initially been distributed between the nodes.

In general, it can be noticed that all policies have managed to exchange high percentages of segments to some extent. This is mainly attributed to the approach followed throughout bayesian optimization. This approach has only allowed four policy feature weights to be adjusted while following the same "black-box" or simulation procedure when deciding on how broadcasting ranges should be set or which segments should be chosen for transmission. If the bayesian optimization iterations had more parameters to tune, then the difference between the different policies would be more significant.

Figure 6.13 shows the potential of the optimal policy found using bayesian optimization with batch-based RF regression in terms of the total data sizes which can be distributed under different data rates. These sizes are given for the amount of data initially distributed using V2I communication, the amount of data after the V2V segment exchanges and the amount of data after the final V2I distribution. They are computed by multiplying the number of segments found using the optimal policy by the corresponding data rate value. This is done for the segments under the different categories by the end of the content distribution period. Notice that the data rates are chosen intentionally between 3 and 27 Mbps which are the minimum and maximum data rates of the conventional DSRC V2V communication technology. Also notice the overlap between the amount of data after the V2V segment exchanges and the amount of data after the final V2I distribution. This is attributed to the fact that the optimal policy has left no segments for the final V2I distribution as discussed previously.

Figure 6.13 shows that the optimal policy has managed to distribute more than 50 GB of data starting with approximately 2.5 GB of data distributed initially using V2I communication. This is assuming the average DSRC data rate of 15 Mbps. Even under the minimum DSRC data rate of 3 Mbps, this policy can still distribute as high as 10 GB starting with only 500 MB of data distributed initially. This gain is quite significant and shows that the network load can be reduced by as much as 95% using the proposed content distribution system. Notice that this offloading percentage is the guaranteed minimum and does not depend on the data rate assumption as long as it is maintained.

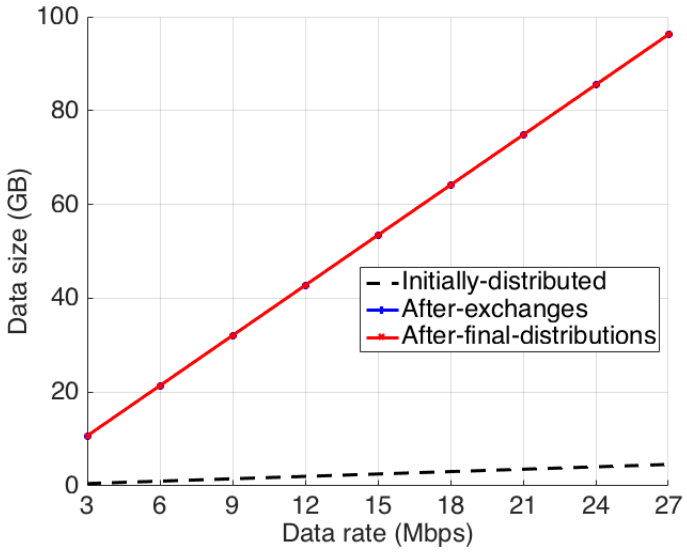


Figure 6.13: Data category size vs. data rate under optimal policy

6.9 Summary

In this chapter, offline operations related to content routing have been discussed as part of the proposed content distribution system. This routing includes the initial V2I segment distributions, the V2V segment exchanges and the final V2I segment distributions. The functions used throughout these operations have been explained first including: $divideData$, $allocateData$, $computeFeatures$, $controlRange$, $targetSegments$, $transmitData$ and $followPolicy$ functions. The offline operations are discussed afterwards including: maximum number of data segments estimation, data segments division and allocation, V2V segment exchange policy optimization and the production of the corresponding action tables. Bayesian optimization based on batch-based RF regression has been used to search for the optimal V2V exchange policy after visualizing the search space and comparing the optimization under different regression techniques including: GP, RF, BNN and batch-based RF. This comparison has been made in terms of the total number of segments exchanged as well as the optimization execution time. This chapter has ended with a comparison of the different policies found against the optimal policy and showed the significant network offloading potential of the proposed content distribution system.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

The majority of content over outdoor wireless networks is public. This ranges from videos at sites such as Youtube or Netflix, pictures at sites such as Instagram to other types of data at social media sites such as Facebook or Twitter. Such popular content consumes large amounts of data, if not the majority of it, which current outdoor wireless networks struggle to support. In fact, this data demand is expected to grow even further with services generated by outdoor things like vehicles including infotainment and sensing services.

Public transportation systems including vehicles and stops have the networking potential to address this huge demand for popular content data in outdoor environments. They can provide a reliable and scalable networking solution for popular content distribution given the distinctive attributes they have including:

- the highly predictable mobility patterns of vehicles,
- the publicly available locations of vehicles and stops,
- the same administrative entity which enforces content routing cooperation, and
- the uniform location distribution of vehicles and stops across the city space as well as throughout the day. This distribution meets the mobility demand of riders who constitute the main consumers of popular content in outdoor environments. Notice that the number of these consumers is multiples of that of the consumers riding regular vehicles who are not driving.

All of these attributes allow for efficient content routing policies to be designed in a gradual, reliable and scalable manner.

After having a reliable ad-hoc network formed by public transportation vehicles and stops, regular vehicles would be incentivized to be part of a reliable and secure ad-hoc network. These vehicles would otherwise struggle with the data-selfishness among themselves. This selfishness is what prohibited VANETs from growing despite the many field trials conducted to show their promising performance.

With public transportation vehicles and stops having ad-hoc network capabilities, regular vehicles would be free from the network effect resulting from them waiting for each other to start owning such capabilities. The resulting huge ad-hoc network of public transportation vehicles/stops and regular vehicles would pave the way for services improving the safety and quality of transportation as a whole.

All of the above are motivations for the popular content distribution system proposed in this study. A system that uses different AI-based techniques to exploit the previously mentioned characteristics of public transportation systems such as a reliable and scalable popular content distribution system is provided. This system would offload significantly the wireless networks in outdoor environments in addition to boosting the adoption of ad-hoc network capabilities among regular vehicles.

This work starts by providing a background and a survey of previous literature related to content distribution. After that, an overview of the proposed content distribution system is provided. This includes the content distribution procedure as well as the databases connected to it. This procedure has the two main sets of operations, namely: offline operations and online operations. Offline operations are explained in details such that each operation is either related to analyzing the mobility of public transportation vehicles and stops or related to routing the popular content among the public transportation vehicles and stops. The experimental approach followed when designing the recommender as part of the online operations is also explained. Notice that, the Grand River Transit bus service offered throughout the Region of Waterloo, Ontario, Canada is adopted as the case study.

For the offline operations related to the mobility analysis, both mobility data preprocessing and processing operations are discussed. The preprocessing operations include: collecting, sorting, cleaning and synthesizing the data. The processing operations include: optimizing stop selections, computing system inter-vehicle/stop connectivities and system vehicle/stop clustering. The clustering operation is presented after evaluating the networking potential of public transportation vehicles and stops.

For the offline operations related to content routing design, the functions used throughout these operations are explained first. These operations include: estimating the maximum number of distributable data segments, dividing and allocating the data segments among the system vehicles and stops and optimizing the V2V segment exchange tables. These data segments as a whole form the popular content(s) being distributed. Bayesian optimization has been chosen as the method of optimizing the exchange tables of these segments over a parameterized routing policy space. The ultimate objective is to distribute as many data segments as possible using V2V segment exchanges while doing so in an interference-free and collision-free manner. Notice that the content distribution is done in three phases: initial V2I distribution, V2V exchanges and final V2I distribution.

Different regression techniques have been implemented and compared throughout bayesian optimizations including: GP, RF, BNN and batch-based RF. The batch-based RF regression has been found to be the fastest while reaching the optimal number of data segment exchanges. Therefore, it has been chosen to be part of the proposed content distribution system.

The recommender used in the online operations has been chosen after experimenting extensively different designs. This experimentation has been conducted under different consumer and network scenarios including: varying consumer group interest distributions, varying unknown consumer interest ratios and varying ad-hoc network capacities. The designs differ in whether they interact with the consumers or not and the nature of this interaction (i.e. Greedy, ϵ-greedy, Decaying ϵ-greedy or UPB based). They also differ in whether they measure consumer-consumer interest-profile similarities in order to make the consumer interactions more efficient or not. In addition, they differ in whether they exploit the existing geographical groups of consumers in order to make more personalized recommendations or not. The final recommender design chosen for the proposed content distribution system is the UPB collaborative and group-based recommender due to its superior performance under the varying scenarios mentioned as proven experimentally.

At the end, the proposed content distribution system has a superior performance in terms of offloading the outdoor wireless network. In fact, it has managed to offload as high as 95% of the wireless network load. Notice that this is still quite conservative given the fact that the 300-meter broadcasting range has been assumed for the transmitting vehicles and stops. In addition, the content distributed by the proposed system would in fact reach more than one consumer due to the large number of riders usually using the public transportation vehicles and stops simultaneously. This means offloading the outdoor wireless network even further and in proportion to the number consumers receiving the same popular content.

With the proposed system, a reliable and scalable ad-hoc network can be formed in a manner that contains gradually the biggest cluster of vehicles and stops until all public transportation system vehicles and stops are included. This network would exploit the public transportation infrastructure in order to significantly offload the wireless network in outdoor environments while reliably providing the backbone for other vehicle categories (e.g. regular vehicles) to join the ad-hoc network system, contribute to its services and benefit from them.

7.2 Future Directions

Future research directions can be divided into the following:

7.2.1 Mobility Analysis Improvements

The analysis proposed is now based on synthesizing vehicle mobility traces without accounting for stops, decelerations or accelerations at traffic intersections, traffic lights or traffic jams. These events affect the locations of vehicles throughout the day and therefore the routing decisions made to distribute data segments. More realistic mobility models incorporating these events are to be included. In fact, the current mobility generation assumes constant vehicle velocities between successive stops which is not completely realistic. More realistic mobility simulations

incorporating acceleration, stopping and deceleration throughout mobility traces are going to be implemented.

Furthermore, the proposed system is currently based on many assumptions which are set by the system designer and therefore an automated way for computing them is to be developed. Examples of these assumptions include: the total number of day periods and their durations, the total number of system nodes, the minimum contact duration thresholds and the maximum number of hops within a cluster.

Finally, clusters are chosen to be the biggest under the minimum contact duration thresholds which is greedy and might not be the best decision to make. This is attributed to the fact that connectivities between nodes within these biggest clusters might be restricted by connectivities occurring simultaneously at nearby previous biggest clusters.

7.2.2 Content Recommender Design Improvements

In order to make better recommendations, more consumer features are going to be included like their age, gender, future trip times and locations. Moreover, some prior beliefs are to be incorporated when doing the UPB estimations. This can be done by assuming that the popularity level indicated by the known consumer interests still holds among the unknown remaining consumer interests given the same service.

7.2.3 Content Routing Design Improvements

More sophisticated directional antennas are going to be considered which would extend the broadcasting range and enhance the data rate significantly.

The popular content is going to be distributed using V2I communication at the beginning of the content distribution cycle while exploiting the vacant frequency bands of the spectrum. These bands are going to be inquired from realistic spectrum databases such as those of TV white space.

Routing gaps are going to be identified between the system nodes and incentives are to be designed for nodes to participate in filling these gaps. For example, existing hotspots at nearby cafes and stores can be incentivized to fill the routing gaps by paying for them and/or distributing advertisements on their behalf. However, notice that public transportation vehicles usually wait at stops for predetermined durations of time while it is not necessarily the case with hotspots. Such waits constitute a significant chance for content distribution that should be exploited.

The way the content is chosen within the same service is also going to be made while taking into account the recency and data size of the different content segments.

Further features measuring the V2I communication potential of nodes are going to be incorporated. They can be different at different nodes at different times of the day and at different parts of the city. Their interactions are also going to be included which would lead to more samples needed and therefore the longer execution time resulting from that during the bayesian optimization is going to be considered. Notice that the difference in performance between policies within this higher dimensionality

feature space is expected to be more significant compared to what has been observed during the current study.

Shorter periods are also going to be considered in order to come up with better exchange policies. Moreover, broadcasting ranges would be more flexible in addition to the way segments are chosen. In other words, the current proposed system does not need to choose greedily the maximum interference-free broadcasting range or the least popular segment among the neighbors to be transmitted.

The policy weights are going to be chosen more flexibly from a set of continuous numbers and need not be chosen from a restricted set of discrete numbers. Notice that this would also lead to a longer execution time.

Dynamic programming is going to be used to come up with better policies by dividing a day period into smaller intervals. The optimization objective is going to be more explicit by maximizing the number of bits exchanged instead of just the number of segments. This is motivated by the fact that not all segments carry the same amount of bits and therefore maximizing them might not be the best bayesian optimization objective. In order to measure the number of bits exchanged, realistic channel models and measurements are going to be used.

Finally, the current optimization approach is going to be enhanced with the ability to react in real time to vehicle schedule irregularities. The focus would be on making the optimization even faster despite the fact that more features are going to be incorporated.

7.2.4 Other Improvements

Other improvements are diverse. Starting with the ability to sort content at the consumer handset in a way that meets their specific interests. In addition, a technique to discard distributed content at the system vehicles and stops as well as the consumer handsets is going to be developed in order to prevent buffer overflows.

The integration between the proposed popular content distribution system and another private real time content distribution system is going to be investigated. Such a private content distribution system is going to be designed while exploiting the predictable mobility patterns of public transportation vehicles.

Transport protocols which are compatible with the proposed content distribution system are going to be developed in a way that also exploits the predictable mobility patterns of public transportation vehicles.

Finally, the matching between the maximum number of distributable data segments and their services content is going to be investigated. This matching is going to be fair and takes into account the number of content segments for each service as well as their sizes and priorities as specified by the content recommender.

Bibliography

[1] “Statistical Report on Internet Development in China”, China Internet Network Information Center, January 2018.

[2] Coleman Bazelon and Giulia McHenry, “Substantial Licensed Spectrum Deficit (2015-2019): Updating the FCC’s Mobile Data Demand Projections”, The Brattle Group, June 23rd, 2015.

[3] “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022”, White paper, Cisco public, February 2019.

[4] “The Mobile Internet Phenomena Report”, Sandvine, February 2019

[5] IHS Economics & IHS Technology (2017), “The 5G Economy: How 5G Technology Will Contribute to the Global Economy”, p.4, January 2017.

[6] Wisitpongphan, N.; Fan Bai; Mudalige, P.; Tonguz, O.K., "On the Routing Problem in Disconnected Vehicular Ad-hoc Networks," in INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE, pp. 2291 - 2295, 6-12 May 2007.

[7] Rubinstein, M.G.; Ben Abdesslem, F.; Dias de Amorim, M.; Cavalcanti, S.R.; Dos Santos Alves, R.; Costa, L.H.M.K.; Duarte, O.C.M.B.; Campista, M.E.M., "Measuring the capacity of in-car to in-car vehicular networks," in Communications Magazine, IEEE, vol.47, no.11, pp.128-136, November 2009.

[8] Sandesh Uppoor and Marco Fiore. 2012. Insights on metropolitan-scale vehicular mobility from a networking perspective. In Proceedings of the 4th ACM international workshop on Hot topics in planet-scale measurement (HotPlanet ’12). ACM, New York, NY, USA, 39-44.

[9] S. Uppoor and M. Fiore, "Large-scale urban vehicular mobility for networking research," 2011 IEEE Vehicular Networking Conference (VNC), Amsterdam, 2011, pp. 62-69.

[10] S. Uppoor, O. Trullols-Cruces, M. Fiore and J. M. Barcelo-Ordinas, "Generation and Analysis of a Large-Scale Urban Vehicular Mobility Dataset," in IEEE Transactions on Mobile Computing, vol. 13, no. 5, pp. 1061-1075, May 2014.

[11] S. Uppoor and M. Fiore, "Characterizing Pervasive Vehicular Access to the Cellular RAN Infrastructure: An Urban Case Study," in IEEE Transactions on Vehicular Technology, vol. 64, no. 6, pp. 2603-2614, June 2015.

[12] H. Zhu, M. Li, L. Fu, G. Xue, Y. Zhu and L. M. Ni, "Impact of Traffic Influxes: Revealing Exponential Intercontact Time in Urban VANETs," in IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 8, pp. 1258-1266, Aug. 2011.

[13] H. Zhu, L. Fu, G. Xue, Y. Zhu, M. Li and L. M. Ni, "Recognizing Exponential Inter-Contact Time in VANETs," 2010 Proceedings IEEE INFOCOM, San Diego, CA, 2010.

[14] K. Zhao, M. P. Chinnasamy and S. Tarkoma, "Automatic City Region Analysis for Urban Routing," 2015 IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, NJ, 2015, pp. 1136-1142.

[15] Park MH., Park HS., Cho SB. (2008) Restaurant Recommendation for Group of People in Mobile Environments Using Probabilistic Multi-criteria Decision Making. In: Lee S., Choo H., Ha S., Shin I.C. (eds) Computer-Human Interaction. APCHI 2008. Lecture Notes in Computer Science, vol 5068. Springer, Berlin, Heidelberg.

[16] Yu Zhiwen, Zhou Xingshe and Zhang Daqing, "An adaptive in-vehicle multimedia recommender for group users," 2005 IEEE 61st Vehicular Technology Conference, Stockholm, 2005, pp. 2800-2804 Vol. 5.

[17] Yong Ge, Hui Xiong, Alexander Tuzhilin, Keli Xiao, Marco Gruteser, and Michael Pazzani. 2010. An energy-efficient mobile recommender system. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '10). ACM, New York, NY, USA, 899-908.

[18] Tuukka Ruotsalo, Krister Haav, Antony Stoyanov, Sylvain Roche, Elena Fani, Romina Deliai, Eetu Mäkelä, Tomi Kauppinen, Eero Hyvönen, SMARTMUSEUM: A mobile recommender system for the Web of Data, Journal of Web Semantics, Volume 20, 2013, Pages 50-67, ISSN 1570-8268.

[19] M. Kenteris, D. Gavalas and A. Mpitziopoulos, "A mobile tourism recommender system," The IEEE symposium on Computers and Communications, Riccione, 2010, pp. 840-845.

[20] Wan-Shiou Yang, San-Yih Hwang, iTravel: A recommender system in mobile peer-to-peer environment, Journal of Systems and Software, Volume 86, Issue 1, 2013, Pages 12-20, ISSN 0164-1212.

[21] T. Li, M. Zhao, A. Liu and C. Huang, "On Selecting Vehicles as Recommenders for Vehicular Social Networks," in IEEE Access, vol. 5, pp. 5539-5555, 2017.

[22] Wan-Shiou Yang, Hung-Chi Cheng, Jia-Ben Dia, A location-aware recommender system for mobile shopping environments, Expert Systems with Applications, Volume 34, Issue 1, 2008, Pages 437-445, ISSN 0957-4174.

[23] Linda Briesemeister and Günter Hommel. 2000. Role-based multicast in highly mobile but sparsely connected ad hoc networks. In Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '00). IEEE Press, Piscataway, NJ, USA, 45-50.

[24] Meng Guo; Ammar, M.H.; Zegura, E.W., "V3: a vehicle-to-vehicle live video streaming architecture," Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on, pp.171,180, 8-12 March 2005.

[25] Bachir, A.; Benslimane, A., "A multicast protocol in ad hoc networks inter - vehicle geocast," Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual, vol.4, pp.2456,2460, 22-25 April 2003.

[26] Briesemeister, L.; Schafers, L.; Hommel, G., "Disseminating messages among highly mobile hosts based on inter-vehicle communication," Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE, pp.522,527, 2000.

[27] Maria Kihl, Mihail Sichitiu, Ted Ekeroth, and Michael Rozenberg. 2007. Reliable Geographical Multicast Routing in Vehicular Ad-Hoc Networks. In Proceedings of the 5th international conference on Wired/Wireless Internet Communications (WWIC '07), Fernando Boavida, Edmundo Monteiro, Saverio Mascolo, and Yevgeni Koucheryavy (Eds.). Springer-Verlag, Berlin, Heidelberg, 315-325.

[28] Kosch, T.; Schwingenschlogl, C.; Li Ai, "Information dissemination in multihop inter-vehicle networks," Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on, pp.685,690, 2002.

[29] Korkmaz, G.; Ekici, E.; Ozguner, F., "A cross-layer multihop data delivery protocol with fairness guarantees for vehicular networks," Vehicular Technology, IEEE Transactions on, vol.55, no.3, pp.865,875, May 2006.

[30] Gökhan Korkmaz, Eylem Ekici, Füsun Özgüner, Supporting real-time traffic in multihop vehicle-to-infrastructure networks, Transportation Research Part C: Emerging Technologies, Volume 18, Issue 3, June 2010, Pages 376-392, ISSN 0968-090X.

[31] Tonguz, O.K.; Wisitpongphan, N.; Fan Bai, "DV-CAST: A distributed vehicular broadcast protocol for vehicular ad hoc networks," Wireless Communications, IEEE, vol.17, no.2, pp.47,57, April 2010.

[32] Eichler, S.; Schroth, C.; Kosch, T.; Strassberger, M., "Strategies for Context-Adaptive Message Dissemination in Vehicular Ad Hoc Networks," Mobile and

Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on, pp.1,9, 17-21 July 2006.

[33] Briesemeister, L.; Schafers, L.; Hommel, G., "Disseminating messages among highly mobile hosts based on inter-vehicle communication," Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE, pp.522,527, 2000.

[34] Schwartz, R.S.; Ohazulike, A.E.; Scholten, H., "Achieving Data Utility Fairness in Periodic Dissemination for VANETs," Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th, pp.1,5, 6-9 May 2012.

[35] Min-Te Sun; Wu-chi Feng; Lai, Ten-Hwang; Yamada, K.; Okada, H.; Fujimura, K., "GPS-based message broadcasting for inter-vehicle communication," Parallel Processing, 2000. Proceedings. 2000 International Conference on, pp.279,286, 2000.

[36] Qing Xu; Segupta, R.; Jiang, D.; Chrysler, D., "Design and analysis of highway safety communication protocol in 5.9 GHz dedicated short range communication spectrum," Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual, vol.4, pp.2451,2455, 22-25 April 2003.

[37] Nasri, A.; Fathy, M.; Hajisheykhi, R., "A Cross Layered Scheme for Broadcasting at Intersections in Vehicular Ad Hoc Networks," Future Networks, 2009 International Conference on, pp.13,17, 7-9 March 2009.

[38] Osafune, T.; Lan Lin; Lenardi, M., "Multi-Hop Vehicular Broadcast (MHVB)," ITS Telecommunications Proceedings, 2006 6th International Conference on, pp.757,760, June 2006.

[39] Mariyasagayam, M.N.; Osafune, T.; Lenardi, M., "Enhanced Multi-Hop Vehicular Broadcast (MHVB) for Active Safety Applications," Telecommunications, 2007. ITST '07. 7th International Conference on ITS, pp.1,6, 6-8 June 2007.

[40] Yuanguo Bi; Cai, L.X.; Xuemin Shen; Hai Zhao, "Efficient and Reliable Broadcast in Intervehicle Communication Networks: A Cross-Layer Approach," Vehicular Technology, IEEE Transactions on, vol.59, no.5, pp.2404,2417, Jun 2010.

[41] Durresi, M.; Durresi, A.; Barolli, L., "Emergency Broadcast Protocol for Inter-Vehicle Communications," Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on, vol.2, pp.402,406, 22-22 July 2005.

[42] Gökhan Korkmaz, Eylem Ekici, Füsun Özgüner, and Ümit Özgüner. 2004. Urban multi-hop broadcast protocol for inter-vehicle communication systems. In Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks (VANET '04). ACM, New York, NY, USA, 76-85.

[43] Di Di; Zhang Dongxia; Limin Sun; Jiangchuan Liu; Li Juanjuan, "A game based routing algorithm for congestion control of multimedia transmission in VANETs," Wireless Communications and Signal Processing (WCSP), 2011 International Conference on , pp.1,6, 9-11 Nov. 2011.

[44] W. Saad, Zhu Han, A. Hjorungnes, D. Niyato, and E. Hossain. 2011. Coalition Formation Games for Distributed Cooperation Among Roadside Units in Vehicular Networks. IEEE J.Sel. A. Commun.29, 1 (January 2011), 48-60.

[45] Shrestha, B.; Niyato, D.; Zhu Han; Hossain, E., "Wireless Access in Vehicular Environments Using BitTorrent and Bargaining," Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, pp.1,5, Nov. 30 2008-Dec. 4 2008.

[46] Tianyu Wang; Lingyang Song; Zhu Han, "Coalitional Graph Games for Popular Content Distribution in Cognitive Radio VANETs," Vehicular Technology, IEEE Transactions on, vol.62, no.8, pp.4010,4019, Oct. 2013.

[47] Tianyu Wang; Lingyang Song; Zhu Han; Bingli Jiao, "Dynamic Popular Content Distribution in Vehicular Networks using Coalition Formation Games," Selected Areas in Communications, IEEE Journal on, vol.31, no.9, pp.538,547, September 2013.

[48] Wikipedia, “Grand River Transit”. Available: https://en.wikipedia.org/wiki/Grand_River_Transit

[49] Transit - GRT GTFS Static & GTFS-realtime Data Feed. Available: https://www.grt.ca/en/about-grt/open-data.aspx

[50] K. E. Suleiman and O. Basir, "Analyzing public transportation mobility data for networking purposes," 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, 2017, pp. 1-6.

Appendix A:
Mobility Analysis Algorithms

Algorithm A.1 Collecting data

1. **Inputs:** $\mathbf{F}^t, \mathbf{F}^{st}, \mathbf{F}^s$
2. **Output:** $\mathbf{F}^c$
3. **for** $i = 1 : length(\mathbf{F}^{st})$ **do**
4. $$\begin{aligned}\mathbf{F}^c[i,*] \;=\; [&\mathbf{F}^t[\underset{j}{argfind}(\mathbf{f}_3^t[j] = \mathbf{f}_1^{st}[i]), \{6,1,5\}] \;\ldots \\ &\mathbf{F}^{st}[i, 1:5] \;\ldots \\ &\mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}_1^s[j] = \mathbf{f}_4^{st}[i]), \{5,6\}] \;\ldots \\ &\mathbf{F}^t[\underset{j}{argfind}(\mathbf{f}_3^t[j] = \mathbf{f}_1^{st}[i]), \{7,2\}]]\end{aligned} \;;$$
5. **end for**

Algorithm A.2 Re-sorting same-block and same-arrival-time row data with switching directions when occurring at the beginning of a trip

1. **Input:** $\mathbf{F}^c$
2. **Output:** $\mathbf{F}^c$
3. **for** $i = 2 : length(\mathbf{F}^c) - 1$ **do**
4. **if** $(\mathbf{f}_3^c[i] \neq \mathbf{f}_3^c[i-1]) \wedge (\mathbf{f}_3^c[i] \neq \mathbf{f}_3^c[i+1])$ **then**
5. %Extracting same-block data:
6. $\mathbf{F}^{bl} = \mathbf{F}^c[\underset{j}{argfind}(\mathbf{f}_1^c[j] = \mathbf{f}_1^c[i]), *]$;
7. %Extracting same-block and same-arrival-time data:
8. $\mathbf{F}_{st}^{bl} = \mathbf{F}^{bl}[\underset{j}{argfind}(\mathbf{f}_5^{bl}[j] = \mathbf{f}_5^c[i]), *]$;
9. %Re-sorting data:
10. **if** $\mathbf{f}_3^c[i] = 0$ **then**
11. $\mathbf{F}_{st}^{bl} \leftarrow sort(\mathbf{F}_{st}^{bl}, 3)$;
12. **else if**
13. $\mathbf{F}_{st}^{bl} \leftarrow sort(\mathbf{F}_{st}^{bl}, -3)$;
14. **end if**
15. $\mathbf{F}^{bl}[\underset{j}{argfind}(\mathbf{f}_5^{bl}[j] = \mathbf{f}_5^c[i]), *] = \mathbf{F}_{st}^{bl}$;
16. $\mathbf{F}^c[\underset{j}{argfind}(\mathbf{f}_1^c[j] = \mathbf{f}_1^c[i]), *] = \mathbf{F}^{bl}$;
17. **end if**
18. **end for**

Algorithm A.3 Re-sorting same-block and same-arrival-time row data with switching directions when occurring at the end of a trip

1. **Input:** $\mathbf{F}^c$
2. **Output:** $\mathbf{F}^c$
3. **for** $i = 2 : length(\mathbf{F}^c) - 1$ **do**
4. **if** $(\mathbf{f}_3^c[i] \neq \mathbf{f}_3^c[i-1]) \wedge (\mathbf{f}_3^c[i] \neq \mathbf{f}_3^c[i+1])$ **then**
5. %Extracting same-block data:
6. $\mathbf{F}^{bl} = \mathbf{F}^c[\underset{j}{argfind}(\mathbf{f}_1^c[j] = \mathbf{f}_1^c[i]), *];$
7. %Extracting same-block and same-arrival-time data:
8. $\mathbf{F}_{st}^{bl} = \mathbf{F}^{bl}[\underset{j}{argfind}(\mathbf{f}_5^{bl}[j] = \mathbf{f}_5^c[i]), *];$
9. %Re-sorting data:
10. **if** $\mathbf{f}_3^c[i] = 0$ **then**
11. $\mathbf{F}_{st}^{bl} \leftarrow sort(\mathbf{F}_{st}^{bl}, -3);$
12. **else if**
13. $\mathbf{F}_{st}^{bl} \leftarrow sort(\mathbf{F}_{st}^{bl}, 3);$
14. **end if**
15. $\mathbf{F}^{bl}[\underset{j}{argfind}(\mathbf{f}_5^{bl}[j] = \mathbf{f}_5^c[i]), *] = \mathbf{F}_{st}^{bl};$
16. $\mathbf{F}^c[\underset{j}{argfind}(\mathbf{f}_1^c[j] = \mathbf{f}_1^c[i]), *] = \mathbf{F}^{bl};$
17. **end if**
18. **end for**

Algorithm A.4 Computing velocities

1. **Input:** $\mathbf{F}^c$
2. **Output:** $\mathbf{F}^c$
3. %Adding the velocity feature column:
4. $\mathbf{f}^c_{12} = \mathbf{0}^{1\times length(\mathbf{F}^c)}$;
5. $\mathbf{F}^c \leftarrow [\mathbf{F}^c \;\; \mathbf{f}^c_{12}]$;
6. %Computing velocities:
7. **for** $i = 2 : length(\mathbf{F}^c)$ **do**
8. **if** $\mathbf{f}^c_1[i] = \mathbf{f}^c_1[i-1]$ **then**
9. $d = ...$
10. $distance([\mathbf{f}^c_9[i-1] \;\; \mathbf{f}^c_{10}[i-1]], [\mathbf{f}^c_9[i] \;\; \mathbf{f}^c_{10}[i]])$;
11. $T_{travel} = \mathbf{f}^c_5[i] - \mathbf{f}^c_6[i-1]$;
12. **if** $T_{travel} = 0$ **then**
13. $\mathbf{f}^c_{12}[i]$ is NaN;
14. **else if**
15. $\mathbf{f}^c_{12}[i] = d/T_{travel}$;
16. **end if**
17. **end if**
18. **end for**

Algorithm A.5 Replacing velocity NaN-values

1. **Input:** $\mathbf{F}^c$
2. **Output:** $\mathbf{F}^c$
3. $\mathbf{id}_{bl} = unique(\mathbf{f}_1^c)$;
4. **for** $i = 1 : length(\mathbf{id}_{bl})$ **do**
5. %Extracting same-block data:
6. $\mathbf{F}^{bl} = \mathbf{F}^c[\underset{j}{argfind}(\mathbf{f}_1^c[j] = \mathbf{id}_{bl}[i]), *]$;
7. %Replacing velocity NaN-values within the block:
8. **while** $\exists k : \mathbf{f}_{12}^{bl}[k]\ is\ NaN$ **do**
9. **for** $j = 2 : length(\mathbf{F}^{bl})\ - 1$ **do**
10. **if** $(\mathbf{f}_{12}^{bl}[j]\ is\ NaN) \wedge (\mathbf{f}_{12}^{bl}[j+1]\ is\ not\ NaN)$ **then**
11. $T_{wait} = \mathbf{f}_6^{bl}[j] - \mathbf{f}_5^{bl}[j]$;
12. $\mathbf{f}_5^{bl}[j] = \mathbf{f}_6^{bl}[j-1] + (\mathbf{f}_5^{bl}[j+1] - \mathbf{f}_6^{bl}[j-1])/2$;
13. $\mathbf{f}_6^{bl}[j] = min(\mathbf{f}_5^{bl}[j] + T_{wait}, \mathbf{f}_5^{bl}[j+1])$;
14. $d = distance([\mathbf{f}_9^{bl}[j]\ \ \mathbf{f}_{10}^{bl}[j]], [\mathbf{f}_9^{bl}[j-1]\ \ \mathbf{f}_{10}^{bl}[j-1]])$;
15. $T_{travel} = \mathbf{f}_5^{bl}[j] - \mathbf{f}_6^{bl}[j-1]$;
16. $\mathbf{f}_{12}^{bl}[j] = d/T_{travel}$;
17. $d = distance([\mathbf{f}_9^{bl}[j]\ \ \mathbf{f}_{10}^{bl}[j]], [\mathbf{f}_9^{bl}[j+1]\ \ \mathbf{f}_{10}^{bl}[j+1]])$;
18. $T_{travel} = \mathbf{f}_5^{bl}[j+1] - \mathbf{f}_6^{bl}[j]$;
19. **if** $T_{travel} = 0$ **then**
20. $\mathbf{f}_{12}^{bl}[j+1]\ is\ NaN$;
21. **else if**
22. $\mathbf{f}_{12}^{bl}[j+1] = d/T_{travel}$;
23. **end if**
24. **end if**
25. **end for**

26. %Replacing velocity NaN-values

27. %at the end of the block:

28. **if** $\mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl})]$ *is* NaN **then**

29. $\mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl})] = \mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl}) - 1]$;

30. $d = distance([\mathbf{f}_{9}^{bl}[length(\mathbf{f}_{9}^{bl})]\ \ \mathbf{f}_{10}^{bl}[length(\mathbf{f}_{10}^{bl})]], ...$

31. $[\mathbf{f}_{9}^{bl}[length(\mathbf{f}_{9}^{bl}) - 1]\ \ \mathbf{f}_{10}^{bl}[length(\mathbf{f}_{10}^{bl}) - 1]])$;

32. $T_{travel} = d/\mathbf{f}_{12}^{bl}[length(\mathbf{f}_{12}^{bl})]$;

33. $T_{wait} = \mathbf{f}_{6}^{bl}[length(\mathbf{f}_{6}^{bl})] - \mathbf{f}_{5}^{bl}[length(\mathbf{f}_{5}^{bl})]$;

34. $\mathbf{f}_{5}^{bl}[length(\mathbf{f}_{5}^{bl})] = \mathbf{f}_{6}^{bl}[length(\mathbf{f}_{6}^{bl}) - 1] + T_{travel}$;

35. $\mathbf{f}_{6}^{bl}[length(\mathbf{f}_{6}^{bl})] = \mathbf{f}_{5}^{bl}[length(\mathbf{f}_{5}^{bl})] + T_{wait}$;

36. **end if**

37. **end while**

38. $\mathbf{F}^{c}[\underset{j}{argfind}(\mathbf{f}_{1}^{c}[j] = \mathbf{id}_{bl}[i]), *] = \mathbf{F}^{bl}$;

39. **end for**

Algorithm A.6 Replacing high velocity values

1. **Inputs:** $\mathbf{F}^c, ve_{max}$
2. **Output:** $\mathbf{F}^c$
3. %Detecting high velocity row indices:
4. $\mathbf{id}^{hve}_{rows} = \underset{j}{argfind}(\mathbf{f}^c_{12}[j] > ve_{max})$;
5. %Replacing high velocity values:
6. **for** $i = 1 : length(\mathbf{id}^{hve}_{rows})$ **do**
7. $\quad j = \mathbf{id}^{hve}_{rows}[i]$;
8. $\quad \mathbf{f}^c_{12}[j] = ve_{max}$;
9. $\quad d = distance([\mathbf{f}^c_9[j] \;\; \mathbf{f}^c_{10}[j]], [\mathbf{f}^c_9[j-1] \;\; \mathbf{f}^c_{10}[j-1]])$;
10. $\quad T_{travel} = d/\mathbf{f}^c_{12}[j]$;
11. $\quad T_{wait} = \mathbf{f}^c_6[j] - \mathbf{f}^c_5[j]$;
12. $\quad \mathbf{f}^c_5[j] = \mathbf{f}^c_6[j-1] + T_{travel}$;
13. $\quad \mathbf{f}^c_6[j] = min(\mathbf{f}^c_5[j] + T_{wait}, \mathbf{f}^c_5[j+1])$;
14. $\quad d = distance([\mathbf{f}^c_9[j] \;\; \mathbf{f}^c_{10}[j]], [\mathbf{f}^c_9[j+1] \;\; \mathbf{f}^c_{10}[j+1]])$;
15. $\quad T_{travel} = \mathbf{f}^c_5[j+1] - \mathbf{f}^c_6[j]$;
16. $\quad \mathbf{f}^c_{12}[j+1] = d/T_{travel}$;
17. **end for**

Algorithm A.7 Modifying data

1. **Input:** $\mathbf{F}^c$
2. **Output:** $\mathbf{F}^m$
3. $\mathbf{F}^m = [\mathbf{f}_1^c \ \mathbf{f}_{11}^c \ \mathbf{f}_4^c \ \mathbf{f}_5^c \ \mathbf{f}_6^c \ \mathbf{f}_9^c \ \mathbf{f}_{10}^c]$;
4. **for** $i = 1 : length(\mathbf{F}^m)$ **do**
5. **if** $\mathbf{f}_4^m[i] \neq \mathbf{f}_5^m[i]$ **then**
6. $\mathbf{row}_{extra} = \mathbf{F}^m[i, *]$;
7. $\mathbf{row}_{extra}[1, 4] = \mathbf{f}_5^m[i]$;
8. $\mathbf{F}^m[i + 1 : length(\mathbf{F}^m) + 1, *] = ...$
9. $(\mathbf{row}_{extra}, \mathbf{F}^m[i + 1 : length(\mathbf{F}^m), *])$;
10. **end if**
11. **end for**

Algorithm A.8 Synthesizing trip data while matching the map

1. **Inputs:** $\mathbf{F}^m, n_T, n_{T_{day}}, \mathbf{F}^{sh}$
2. **Outputs:** $\mathbf{LATS}_{tr}, \mathbf{LONS}_{tr}$
3. %Extracting the vector of trip IDs:
4. $\mathbf{id}_{tr} = unique(\mathbf{f}_3^m)$;
5. %Synthesizing trip trajectories:
6. **for** $i = 1 : length(\mathbf{id}_{tr})$ **do**
7. $\quad \mathbf{TR}_{tr} = \mathbf{F}^m[\underset{j}{argfind}(\mathbf{f}_3^m[j] = \mathbf{id}_{tr}[i]), *]$;
8. $\quad \mathbf{Mdl}_{lat} = fitlinear(\mathbf{TR}_{tr}[*, 4], \mathbf{TR}_{tr}[*, 5])$;
9. $\quad \mathbf{Mdl}_{lon} = fitlinear(\mathbf{TR}_{tr}[*, 4], \mathbf{TR}_{tr}[*, 6])$;
10. $\quad$ **for** $t = 1 : n_T$ **do**
11. $\quad\quad$ **if** $(t/n_{T_{day}} \geq min(\mathbf{TR}_{tr}[*, 4]) \wedge (t/n_{T_{day}} \leq max(\mathbf{TR}_{tr}[*, 4])$ **then**
12. $\quad\quad\quad \mathbf{mdl}_{result} = [\mathbf{Mdl}_{lat}(t/n_{T_{day}})\ \ \mathbf{Mdl}_{lon}(t/n_{T_{day}})]$;
13. $\quad\quad\quad$ %Matching trajectories to map shapes:
14. $\quad\quad\quad \mathbf{TR}_{sh} = \mathbf{F}^{sh}[\underset{j}{argfind}(\mathbf{f}_1^{sh}[j] = \mathbf{TR}_{tr}[1, 2]), 1 : 3]$;
15. $\quad\quad\quad$ **for** $j = 1 : length(\mathbf{TR}_{sh})$ **do**
16. $\quad\quad\quad\quad \mathbf{d}[j] = ...$
17. $\quad\quad\quad\quad distance([\mathbf{mdl}_{result}[1]\ \ \mathbf{mdl}_{result}[2]], [\mathbf{TR}_{sh}[j, 2]\ \ \mathbf{TR}_{sh}[j, 3]])$;
18. $\quad\quad\quad$ **end for**
19. $\quad\quad\quad \mathbf{LATS}_{tr}[i, t] = \mathbf{TR}_{sh}[\underset{j}{argmin}(\mathbf{d}[j]), 2]$;
20. $\quad\quad\quad \mathbf{LONS}_{tr}[i, t] = \mathbf{TR}_{sh}[\underset{j}{argmin}(\mathbf{d}[j]), 3]$;
21. $\quad\quad$ **end if**
22. $\quad$ **end for**
23. **end for**

Algorithm A.9 Synthesizing block data

1. **Inputs:** $\mathbf{F}^m, \mathbf{LATS}_{tr}, \mathbf{LONS}_{tr}$
2. **Outputs:** $\mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}$
3. %Extracting the vector of block IDs:
4. $\mathbf{id}_{bl} = unique(\mathbf{f}_1^m)$;
5. %Extracting the vector of trip IDs:
6. $\mathbf{id}_{tr} = unique(\mathbf{f}_3^m)$;
7. %Synthesizing block trajectories:
8. **for** $i = 1 : length(\mathbf{id}_{bl})$ **do**
9. $\quad \mathbf{id}_{tr}^{bl} = unique(\mathbf{F}^m[\underset{j}{argfind}(\mathbf{f}_1^m[j] = \mathbf{id}_{bl}[i]), 3])$;
10. $\quad id_{tr}^{bl}|_{first} = \underset{j}{argfind}(\mathbf{id}_{tr}[j] = \mathbf{id}_{tr}^{bl}[1])$;
11. $\quad id_{tr}^{bl}|_{last} = \underset{j}{argfind}(\mathbf{id}_{tr}[j] = \mathbf{id}_{tr}^{bl}[length(\mathbf{id}_{tr}^{bl})])$;
12. $\quad \mathbf{LATS}_{bl}[i, *] = \sum_{j=id_{tr}^{bl}|_{first}}^{id_{tr}^{bl}|_{last}} \mathbf{LATS}_{tr}[j, *]$;
13. $\quad \mathbf{LONS}_{bl}[i, *] = \sum_{j=id_{tr}^{bl}|_{first}}^{id_{tr}^{bl}|_{last}} \mathbf{LONS}_{tr}[j, *]$;
14. **end for**

Algorithm A.10 Filling gaps between same-block trips

1. **Inputs:** $\mathbf{id}_{bl}, \mathbf{F}^m, n_T, n_{T_{day}}, \mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}$
2. **Outputs:** $\mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}$
3. **for** $i = 1 : length(\mathbf{id}_{bl})$ **do**
4. $\quad \mathbf{F}^m_{bl} = \mathbf{F}^m[\underset{j}{argfind}(\mathbf{f}^m_1[j] = \mathbf{id}_{bl}[i]), *]$;
5. $\quad t^{bl}_{start} = \mathbf{F}^m_{bl}[1, 4]$; $t^{bl}_{end} = \mathbf{F}^m_{bl}[length(\mathbf{F}^m_{bl}), 4]$;
6. $\quad$ %Filling the block gaps:
7. $\quad$ **for** $t = 1 : n_T$ **do**
8. $\quad\quad$ **if** $(t^{bl}_{start} \leq t/n_{T_{day}} \leq t^{bl}_{end}) \wedge (\mathbf{LATS}_{bl}[i, t] = 0)$ **then**
9. $\quad\quad\quad n_{T_{gap}} = 0$;
10. $\quad\quad\quad$ **while** $\mathbf{LATS}_{bl}[i, t + n_{T_{gap}}] = 0$ **do**
11. $\quad\quad\quad\quad n_{T_{gap}} \leftarrow n_{T_{gap}} + 1$;
12. $\quad\quad\quad$ **end while**
13. $\quad\quad\quad \mathbf{TR}^{lats}_{gap} = ([(t-1)/n_{T_{day}}\ \ \mathbf{LATS}_{bl}[i, t-1]], ...$
14. $\quad\quad\quad [(t + n_{T_{gap}})/n_{T_{day}}\ \ \mathbf{LATS}_{bl}[i, t + n_{T_{gap}}]])$;
15. $\quad\quad\quad \mathbf{Mdl}_{lat} = fitlinear(\mathbf{TR}^{lats}_{gap}[*, 1], \mathbf{TR}^{lats}_{gap}[*, 2])$;
16. $\quad\quad\quad \mathbf{LATS}_{bl}[i, t : t + n_{T_{gap}}] = \mathbf{Mdl}_{lat}((t : t + n_{T_{gap}})/n_{T_{day}})$;
17. $\quad\quad\quad \mathbf{TR}^{lons}_{gap} = ([(t-1)/n_{T_{day}}\ \ \mathbf{LONS}_{bl}[i, t-1]], ...$
18. $\quad\quad\quad [(t + n_{T_{gap}})/n_{T_{day}}\ \ \mathbf{LONS}_{bl}[i, t + n_{T_{gap}}]])$;
19. $\quad\quad\quad \mathbf{Mdl}_{lon} = fitlinear(\mathbf{TR}^{lons}_{gap}[*, 1], \mathbf{TR}^{lons}_{gap}[*, 2])$;
20. $\quad\quad\quad \mathbf{LONS}_{bl}[i, t : t + n_{T_{gap}}] = \mathbf{Mdl}_{lon}((t : t + n_{T_{gap}})/n_{T_{day}})$;
21. $\quad\quad$ **end if**
22. $\quad$ **end for**
23. **end for**

Algorithm A.11 Correcting same-block trip single-step overlaps

1. **Inputs:** $\mathbf{id}_{bl}, \mathbf{F}^m, n_T, n_{T_{day}}, \mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}, ...$
2. $\mathbf{LATS}_{tr}, \mathbf{LONS}_{tr}$
3. **Outputs:** $\mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}$
4. **for** $i = 1 : length(\mathbf{id}_{bl})$ **do**
5. $\quad \mathbf{F}^m_{bl}$=$\mathbf{F}^m[\underset{j}{argfind}(\mathbf{f}^m_1[j] = \mathbf{id}_{bl}[i]), *]$;
6. $\quad t^{bl}_{start} = \mathbf{F}^m_{bl}[1, 4]$; $t^{bl}_{end} = \mathbf{F}^m_{bl}[length(\mathbf{F}^m_{bl}), 4]$;
7. $\quad$%Correcting overlaps:
8. $\quad$**for** $t = 1 : n_T$ **do**
9. $\quad\quad$**if** $(t^{bl}_{start} \leq t/n_{T_{day}} \leq t^{bl}_{end})$ **then**
10. $\quad\quad\quad$**if** $(\mathbf{LATS}_{bl}[i, t] > max(\mathbf{LATS}_{tr}))$...
11. $\quad\quad\quad\vee(\mathbf{LONS}_{bl}[i, t] < min(\mathbf{LONS}_{tr}))$ **then**
12. $\quad\quad\quad\quad$**if** $((\mathbf{LATS}_{bl}[i, t-1] \leq max(\mathbf{LATS}_{tr}))$...
13. $\quad\quad\quad\quad\wedge(\mathbf{LATS}_{bl}[i, t+1] \leq max(\mathbf{LATS}_{tr})))$...
14. $\quad\quad\quad\quad\vee((\mathbf{LONS}_{bl}[i, t-1] \geq min(\mathbf{LONS}_{tr}))$...
15. $\quad\quad\quad\quad\wedge(\mathbf{LONS}_{bl}[i, t+1] \geq min(\mathbf{LONS}_{tr})))$ **then**
16. $\quad\quad\quad\quad\quad\mathbf{LATS}_{bl}[i, t] \leftarrow \mathbf{LATS}_{bl}[i, t]/2$; $\mathbf{LONS}_{bl}[i, t] \leftarrow \mathbf{LONS}_{bl}[i, t]/2$;
17. $\quad\quad\quad\quad$**end if**
18. $\quad\quad\quad$**end if**
19. $\quad\quad$**end if**
20. $\quad$**end for**
21. **end for**

Algorithm A.12 Converting stop-coordinates data

1. **Inputs:** $\mathbf{F}^{s}, r_{earth}$
2. **Output:** $\mathbf{F}^{cs}$
3. $[\mathbf{f}_1^{cs}\ \ \mathbf{f}_2^{cs}\ \ \mathbf{f}_3^{cs}] = unique(\mathbf{F}^{s}[*, \{1, 5, 6\}])$;
4. $\mathbf{f}_4^{cs} = r_{earth} \cdot cos(\mathbf{f}_2^{cs}) \cdot cos(\mathbf{f}_3^{cs})$;
5. $\mathbf{f}_5^{cs} = r_{earth} \cdot cos(\mathbf{f}_2^{cs}) \cdot sin(\mathbf{f}_3^{cs})$;
6. $\mathbf{f}_6^{cs} = r_{earth} \cdot sin(\mathbf{f}_2^{cs})$;
7. $\mathbf{F}^{cs} = [\mathbf{f}_1^{cs}\ \ \mathbf{f}_2^{cs}\ \ ...\ \ \mathbf{f}_6^{cs}]$;

Algorithm A.13 Refining stop selections

1. **Inputs:** $\mathbf{F}^{cs}, r_{br}$
2. **Output:** $\mathbf{id}_{rs}$
3. %Extracting the stop cluster IDs:
4. $\mathbf{id}_{sc} = cluster(\mathbf{F}^{cs}[*, 4:6], 'linkage', 'complete', ...$
5. $'criterion', 'distance', 'cutoff', r_{br})$;
6. %Refining stops by selecting stop cluster medoids:
7. **for** $i = 1 : length(unique(\mathbf{id}_{sc}))$ **do**
8. $\quad \mathbf{id}_{sc_i} = \mathbf{F}^{cs}[\underset{j}{argfind}(\mathbf{id}_{sc}[j] = i), 1]$;
9. $\quad \mathbf{CO}_{sc_i} = \mathbf{F}^{cs}[\underset{j}{argfind}(\mathbf{id}_{sc}[j] = i), 2:3]$;
10. $\quad$ **for** $j = 1 : length(\mathbf{id}_{sc_i})$ **do**
11. $\quad\quad \mathbf{d}_{sc_i}[j] = distance(\mathbf{CO}_{sc_i}[j, *], \sum_{k} \mathbf{CO}_{sc_i}[k, *]/length(\mathbf{CO}_{sc_i}))$;
12. $\quad$ **end for**
13. $\quad \mathbf{id}_{rs}[i] = \mathbf{id}_{sc_i}[\underset{j}{argmin}(\mathbf{d}_{sc_i}[j])]$;
14. **end for**

Algorithm A.14 Optimizing stop selections

1. **Inputs:** $\mathbf{id}_{rs}, n_T, \mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}, \mathbf{F}^s, r_{br}, n_{os}$
2. **Output:** $\mathbf{id}_{os}$
3. %Initializing refined stop popularities:
4. $\mathbf{pop}_{stops}[1 : length(\mathbf{id}_{rs})] = \mathbf{0}^{1\times length(\mathbf{id}_{rs})}$;
5. %Computing refined stop popularities:
6. **for** $i = 1 : length(\mathbf{id}_{rs})$ **do**
7. **for** $t = 1 : n_T$ **do**
8. **if** $\exists k : \mathbf{LATS}_{bl}[k, t] \neq 0$ **then**
9. $\mathbf{CO}_{bl} = [\mathbf{LATS}_{bl}[\underset{j}{argfind}(\mathbf{LATS}_{bl}[j, t] \neq 0), t]$ ...
10. $\mathbf{LONS}_{bl}[\underset{j}{argfind}(\mathbf{LONS}_{bl}[j, t] \neq 0), t]]$;
11. $\mathbf{co}_{s_i} = \mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}^s_1[j] = \mathbf{id}_{rs}[i]), 5 : 6]$;
12. **for** $j = 1 : length(\mathbf{CO}_{bl})$ **do**
13. **if** $distance(\mathbf{CO}_{bl}[j, *], \mathbf{co}_{s_i}) \leq r_{br}$ **then**
14. $\mathbf{pop}_{stops}[i] \leftarrow \mathbf{pop}_{stops}[i] + 1$;
15. **end if**
16. **end for**
17. **end if**
18. **end for**
19. **end for**
20. %Selecting the first n_{os} most popular refined stops as the optimal stops set:
21. $[\sim, \mathbf{id}_{os}] = sort(\mathbf{pop}_{stops}, -1)$;
22. $\mathbf{id}_{os} = \mathbf{id}_{rs}[\mathbf{id}_{os}]$;
23. $\mathbf{id}_{os} \leftarrow \mathbf{id}_{os}[1 : n_{os}]$;

Algorithm A.15 Adding optimal stops data to synthetic block data

1. **Inputs:** $\mathbf{id}_{bl}, \mathbf{LATS}_{bl}, \mathbf{LONS}_{bl}, \mathbf{id}_{os}, \mathbf{F}^s, n_T$
2. **Outputs:** $\mathbf{LATS}, \mathbf{LONS}$
3. $\mathbf{LATS}[1 : length(\mathbf{id}_{bl}), *] = \mathbf{LATS}_{bl}$;
4. $\mathbf{LONS}[1 : length(\mathbf{id}_{bl}), *] = \mathbf{LONS}_{bl}$;
5. **for** $i = 1 : length(\mathbf{id}_{os})$ **do**
6. $\quad \mathbf{LATS}[length(\mathbf{id}_{bl}) + i, *] = ...$
7. $\quad repmat(\mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}_1^s[j] = \mathbf{id}_{os}[i]), 5], 1, n_T)$;
8. $\quad \mathbf{LONS}[length(\mathbf{id}_{bl}) + i, *] = ...$
9. $\quad repmat(\mathbf{F}^s[\underset{j}{argfind}(\mathbf{f}_1^s[j] = \mathbf{id}_{os}[i]), 6], 1, n_T)$;
10. **end for**

Algorithm A.16 Computing connectivities

1. **Inputs:** $n_T, n_n, \mathbf{LATS}, \mathbf{LONS}, r_{br}$
2. **Output:** $\mathbf{C}$
3. %Initializing the connectivities matrix:
4. $\mathbf{C} = \mathbf{0}^{n_T \times n_n \times n_n}$;
5. %Computing node connectivities:
6. **for** $t = 1 : n_T$ **do**
7. **for** $i = 1 : n_n - 1$ **do**
8. **if** $\mathbf{LATS}[i, t] \neq 0$ **then**
9. **for** $j = i + 1 : n_n$ **do**
10. **if** $\mathbf{LATS}[j, t] \neq 0$ **then**
11. $d = distance([\mathbf{LATS}[i, t] \ \ \mathbf{LONS}[i, t]], ...$
12. $[\mathbf{LATS}[j, t] \ \ \mathbf{LONS}[j, t]])$;
13. **if** $d \leq r_{br}$ **then**
14. $\mathbf{C}[t, i, j] = 1$;
15. $\mathbf{C}[t, j, i] = \mathbf{C}[t, i, j]$;
16. **end if**
17. **end if**
18. **end for**
19. **end if**
20. **end for**
21. **end for**

Algorithm A.17 Computing continuous contact durations

Inputs: $n_p, n_n, \mathbf{t}^{start}, \mathbf{t}^{end}, \mathbf{C}, \triangle cd$

Output: CD

for $p = 1 : n_p$ **do**

  %Initializing the continuous contact

  %durations matrix of period p:

  $\mathbf{CD}\{p\} = \emptyset$;

  %Computing continuous contact durations:

  for $i = 1 : n_n$ **do**

    %Initializing contact duration counters:

    $\mathbf{cd} = \mathbf{0}^{1 \times n_n}$;

    for $t = \mathbf{t}^{start}[p] : \mathbf{t}^{end}[p]$ **do**

      for $j = 1 : n_n$ **do**

        if $\mathbf{C}[t, i, j] = 1$ **then**

          $\mathbf{cd}[j] \leftarrow \mathbf{cd}[j] + \triangle cd$;

        else if

          if $(t \neq \mathbf{t}^{start}[p]) \wedge (\mathbf{C}[t-1, i, j] = 1)$ **then**

            %Adding a duration and resetting its counter:

            $\mathbf{CD}\{p\} \leftarrow (\mathbf{CD}\{p\}, \mathbf{cd}[j])$;

            $\mathbf{cd}[j] = 0$;

          end if

        end if

      end for

    end for

  end for

end for

Algorithm A.18 Extracting the c^{th} biggest cluster ($extractCluster$)

1. **Inputs:** $c, n_n, t^{start}, t^{end}, \mathbf{C}, \triangle cd, cd_{min}, n_{h_{max}}$
2. **Outputs:** $\mathbf{ID}_{nc}^{b}\{c\}$
3. %Initializing node clusters:
4. **for** $i = 1 : n_n$ **do**
5. $\quad \mathbf{c}_p = \sum_{t=t^{start}}^{t^{end}} \mathbf{C}[t, i, *]$;
6. $\quad \mathbf{ID}_{nc}^{prvs}\{i\} = \underset{j}{argfind}(\mathbf{c}_p[j] \times \triangle cd \geq cd_{min})$;
7. **end for**
8. %Extending clusters to include multi-hop nodes:
9. $\mathbf{ID}_{nc}^{nxt} = \mathbf{ID}_{nc}^{prvs}$;
10. **for** $h = 1 : n_{h_{max}}/2 - 1$ **do**
11. $\quad$ **for** $i = 1 : n_n$ **do**
12. $\quad\quad$ **for** $j = 1 : length(\mathbf{ID}_{nc}^{prvs}\{i\})$ **do**
13. $\quad\quad\quad \mathbf{ID}_{nc}^{nxt}\{i\} \leftarrow \mathbf{ID}_{nc}^{nxt}\{i\} \cup \mathbf{ID}_{nc}^{prvs}\{\mathbf{ID}_{nc}^{prvs}\{i\}[j]\}$;
14. $\quad\quad$ **end for**
15. $\quad$ **end for**
16. $\quad \mathbf{ID}_{nc}^{prvs} = \mathbf{ID}_{nc}^{nxt}$;
17. **end for**
18. %Extracting the c^{th} biggest cluster:
19. $[\mathbf{ID}_{nc}, \mathbf{id}_{nc}, \sim] = unique(\mathbf{ID}_{nc}^{nxt})$;
20. **for** $i = 1 : length(\mathbf{id}_{nc})$ **do**
21. $\quad \mathbf{sizes}_{nc}[i] = length(\mathbf{ID}_{nc}\{i\})$;
22. **end for**
23. $\mathbf{ID}_{nc}^{b}\{c\} = \mathbf{ID}_{nc}\{\underset{i}{argmax}(\mathbf{sizes}_{nc}[i])\}$;

Algorithm A.19 Extracting next biggest clusters

1. **Inputs:** $\mathbf{cd}_{min}, t^{start}, t^{end}, \mathbf{C}, \mathbf{ID}^b_{nc}, c, n_n, \triangle cd, n_{h_{max}}$
2. **Outputs:** $\mathbf{ID}^b_{nc}$
3. **for** $i = 2 : length(\mathbf{cd}_{min})$ **do**
4. %Removing connectivities
5. %of the previous biggest nodes cluster:
6. $n_{T_{period}} = t^{end} - t^{start} + 1$;
7. **for** $t = 1 : n_{T_{period}}$ **do**
8. $\mathbf{C}[t, \mathbf{ID}^b_{nc}\{c\}, *] = \mathbf{0}^{length(\mathbf{ID}^b_{nc}\{c\}) \times n_n}$;
9. $\mathbf{C}[t, *, \mathbf{ID}^b_{nc}\{c\}] = \mathbf{0}^{n_n \times length(\mathbf{ID}^b_{nc}\{c\})}$;
10. **end for**
11. %Extracting the next biggest nodes cluster:
12. $cd_{min} = \mathbf{cd}_{min}[i]$;
13. $c = c + 1$;
14. $\mathbf{ID}^b_{nc}\{c\} = ...$
15. $extractCluster(c, n_n, t^{start}, t^{end}, \mathbf{C}, \triangle cd, cd_{min}, n_{h_{max}})$;
16. **end for**

Appendix B: Content Recommender Design Algorithms

Algorithm B.1 Generating consumer true interests

1. **Inputs:** $n_c, n_s, n_g, \sigma_i^g$
2. **Output:** $\mathbf{INT}_{true}$
3. $\mathbf{INT}_{true} = \mathbf{0}^{n_c \times n_s}$;
4. **for** $g = 1 : n_g$ **do**
5. $\quad id_c^f = n_c/n_g \times (g-1) + 1$;
6. $\quad id_c^l = n_c/n_g \times g$;
7. $\quad id_s^f = n_s/n_g \times (g-1) + 1$;
8. $\quad id_s^l = n_s/n_g \times g$;
9. $\quad$ **for** $j = id_c^f : id_c^l$ **do**
10. $\quad\quad \mathbf{id}_{\mathbf{INT}_{true}} = ...$
11. $\quad\quad \lfloor normrnd(\parallel (id_s^f + id_s^l)/2 \parallel, \sigma_i^g, (1, n_s/n_g)) \rfloor$;
12. $\quad\quad \mathbf{id}_{\mathbf{INT}_{true}}[\underset{i}{arg find}(\mathbf{id}_{\mathbf{INT}_{true}}[i] < 1)] = \phi$;
13. $\quad\quad \mathbf{id}_{\mathbf{INT}_{true}}[\underset{i}{arg find}(\mathbf{id}_{\mathbf{INT}_{true}}[i] > n_s)] = \phi$;
14. $\quad\quad \mathbf{INT}_{true}[j, \mathbf{id}_{\mathbf{INT}_{true}}] = \mathbf{1}^{1 \times length(\mathbf{id}_{\mathbf{INT}_{true}})}$;
15. $\quad$ **end for**
16. **end for**

Algorithm B.2 Confirming that at least one service is truly liked per consumer

1. **Inputs:** $n_c, \mathbf{INT}_{true}, n_g, n_s$
2. **Output:** $\mathbf{INT}_{true}$
3. **for** $i = 1 : n_c$ **do**
4. **if** $\forall j, \mathbf{INT}_{true}[i, j] = 0$ **then**
5. $\mathbf{INT}_{true}[i, ...$
6. $randi(\lfloor i/(\frac{n_c}{n_g})\rfloor \times (\frac{n_s}{n_g}) + 1, (\lfloor i/(\frac{n_c}{n_g})\rfloor + 1) \times (\frac{n_s}{n_g}))] = 1;$
7. **end if**
8. **end for**

Algorithm B.3 Determining consumer available interests

1. **Inputs:** $\mathbf{INT}_{true}, n_c, n_s, ratio_{ui}$
2. **Output:** $\mathbf{INT}_{avail}$
3. $\mathbf{INT}_{avail} = \mathbf{INT}_{true};$
4. **for** $i = 1 : n_c$ **do**
5. **for** $j = 1 : n_s$ **do**
6. **if** $randn \leq ratio_{ui}$ **then**
7. $\mathbf{INT}_{avail}[i, j]\ is\ NaN;$
8. **end if**
9. **end for**
10. **end for**

Algorithm B.4 Confirming that at least one service is known to be liked per consumer

1. **Inputs:** $n_c, n_s, \mathbf{INT}_{avail}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{INT}_{avail}$
3. **for** $i = 1 : n_c$ **do**
4. **for** $j = 1 : n_s$ **do**
5. **if** $(\forall k : \mathbf{INT}_{avail}[i,k]\ is\ not\ NaN, \mathbf{INT}_{avail}[i,k] = 0)...$
6. $\wedge(\mathbf{INT}_{avail}[i,j]\ is\ NaN) \wedge (\mathbf{INT}_{true}[i,j] = 1)$ **then**
7. $\mathbf{INT}_{avail}[i,j] = \mathbf{INT}_{true}[i,j]$;
8. **break for**
9. **end if**
10. **end for**
11. **end for**

Algorithm B.5 Non-interactive non-collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_s, n_c, \mathbf{INT}_{avail}, cap_{net}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $s = 1 : n_s$ **do**
5. $\mathbf{pop}^{avail}_{services}[s] = \sum_{i:\mathbf{INT}_{avail}[i,s]\ is\ not\ NaN} \mathbf{INT}_{avail}[i,s]$;
6. **end for**
7. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{avail}_{services}, -1)$;
8. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$;
9. **end for**

Algorithm B.6 Greedy non-collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_s, n_c, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $s = 1 : n_s$ **do**
5. $\mathbf{pop}^{avail}_{services}[s] = \sum_{i:\mathbf{INT}_{avail}[i,s] \text{ is not } NaN} \mathbf{INT}_{avail}[i, s]$;
6. **end for**
7. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{avail}_{services}, -1)$;
8. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$;
9. **for** $i = 1 : n_c$ **do**
10. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
11. **end for**
12. **end for**

Algorithm B.7 ϵ-greedy non-collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_s, n_c, \mathbf{INT}_{avail}, ratio_{ui}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $s = 1 : n_s$ **do**
5. $\mathbf{pop}^{avail}_{services}[s] = \sum_{i:\mathbf{INT}_{avail}[i,s] \text{ is not } NaN} \mathbf{INT}_{avail}[i, s]$;
6. **end for**
7. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{avail}_{services}, -1)$;
8. $\epsilon = ratio_{ui}$;
9. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1 - \epsilon) \rfloor]$;
10. $\mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds}\ \ \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1 - \epsilon) \rfloor + 1, n_s), ...$
11. $1, cap_{net} - \lfloor cap_{net} \times (1 - \epsilon) \rfloor))]$;
12. **for** $i = 1 : n_c$ **do**
13. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
14. **end for**
15. **end for**

Algorithm B.8 Decaying ϵ-greedy non-collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_s, n_c, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $s = 1 : n_s$ **do**
5. $\mathbf{pop}^{avail}_{services}[s] = \sum_{i:\mathbf{INT}_{avail}[i,s] \; is \; not \; NaN} \mathbf{INT}_{avail}[i, s];$
6. **end for**
7. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{avail}_{services}, -1);$
8. $\epsilon = 1 - ...$
9. $length(\underset{(i,s)}{argfind}(\mathbf{INT}_{avail}[i, s] \; is \; not \; NaN))/(n_c \times n_s);$
10. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1 - \epsilon) \rfloor];$
11. $\mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds} \;\; \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1 - \epsilon) \rfloor + 1, n_s), ...$
12. $1, cap_{net} - \lfloor cap_{net} \times (1 - \epsilon) \rfloor))];$
13. **for** $i = 1 : n_c$ **do**
14. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}];$
15. **end for**
16. **end for**

Algorithm B.9 Upper-Popularity-Bound non-collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_s, n_c, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $s = 1 : n_s$ **do**
5. $\mathbf{pop}^{avail}_{services}[s] = \sum_{i:\mathbf{INT}_{avail}[i,s] \ is \ not \ NaN} \mathbf{INT}_{avail}[i, s];$
6. $\mathbf{u}_{avail}[s] = ...$
7. $\mathbf{pop}^{avail}_{services}[s] + length(\underset{i}{argfind}(\mathbf{INT}_{avail}[i, s] \ is \ NaN));$
8. **end for**
9. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{u}_{avail}, -1);$
10. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}];$
11. **for** $i = 1 : n_c$ **do**
12. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}];$
13. **end for**
14. **end for**

Algorithm B.10 Generating Non-Group-Based interest Recommendations using consumers collaboration (generateNGBRecommendations)

1. **Inputs:** $n_c, n_s, \mathbf{INT}_{avail}$
2. **Output:** $\mathbf{INT}_{rcmnd}$
3. $\mathbf{J} = \mathbf{0}^{n_c \times n_c}$;
4. **for** $i = 1 : n_c$ **do**
5. **for** $j = 1 : n_c$ **do**
6. **if** $i = j$ **then**
7. **continue for**
8. **else if**
9. $\mathbf{J}[i, j] = ...$
10. $length(\underset{s}{argfind}(\mathbf{INT}_{avail}[i, s] = \mathbf{INT}_{avail}[j, s]))/n_s$;
11. **end if**
12. **end for**
13. **end for**
14. $\mathbf{INT}_{rcmnd} = \mathbf{INT}_{avail}$;
15. **for** $i = 1 : n_c$ **do**
16. **for** $s = 1 : n_s$ **do**
17. **if** $\mathbf{INT}_{rcmnd}[i, s]$ is NaN **then**
18. $\mathbf{INT}_{rcmnd}[i, s] \leftarrow \mathbf{INT}_{rcmnd}[\underset{j}{argmax}(\mathbf{J}[i, j]), s]$;
19. **end if**
20. **end for**
21. **end for**

Algorithm B.11 Greedy collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_c, n_s, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. $\quad \mathbf{INT}_{rcmnd} = ...$
5. $\quad generateNGBRecommendations(n_c, n_s, \mathbf{INT}_{avail})$;
6. $\quad$ **for** $s = 1 : n_s$ **do**
7. $\quad\quad \mathbf{pop}^{rcmnd}_{services}[s] = \sum_{i:\mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i, s]$;
8. $\quad$ **end for**
9. $\quad [\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{rcmnd}_{services}, -1)$;
10. $\quad \mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$;
11. $\quad$ **for** $i = 1 : n_c$ **do**
12. $\quad\quad \mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
13. $\quad$ **end for**
14. **end for**

Algorithm B.12 ϵ-greedy collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_c, n_s, \mathbf{INT}_{avail}, ratio_{ui}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. $\quad \mathbf{INT}_{rcmnd} = ...$
5. $\quad generateNGBRecommendations(n_c, n_s, \mathbf{INT}_{avail})$;
6. $\quad$ **for** $s = 1 : n_s$ **do**
7. $\quad\quad \mathbf{pop}^{rcmnd}_{services}[s] = \sum_{i:\mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i, s]$;
8. $\quad$ **end for**
9. $\quad [\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{rcmnd}_{services}, -1)$;
10. $\quad \epsilon = ratio_{ui}$;
11. $\quad \mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1 - \epsilon) \rfloor]$;
12. $\quad \mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds}\ \ \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1 - \epsilon) \rfloor + 1, n_s), ...$
13. $\quad 1, cap_{net} - \lfloor cap_{net} \times (1 - \epsilon) \rfloor))]$;
14. $\quad$ **for** $i = 1 : n_c$ **do**
15. $\quad\quad \mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
16. $\quad$ **end for**
17. **end for**

Algorithm B.13 Decaying ϵ-greedy collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_c, n_s, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. $\quad \mathbf{INT}_{rcmnd} = ...$
5. $\quad generateNGBRecommendations(n_c, n_s, \mathbf{INT}_{avail})$;
6. $\quad$ **for** $s = 1 : n_s$ **do**
7. $\quad\quad \mathbf{pop}^{rcmnd}_{services}[s] = \sum_{i:\mathbf{INT}_{rcmnd}[i,s] \ is \ not \ NaN} \mathbf{INT}_{rcmnd}[i, s]$;
8. $\quad$ **end for**
9. $\quad [\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}^{rcmnd}_{services}, -1)$;
10. $\quad \epsilon = 1 - ...$
11. $\quad length(\underset{(i,s)}{argfind}(\mathbf{INT}_{avail}[i, s] \ is \ not \ NaN))/(n_c \times n_s)$;
12. $\quad \mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1 - \epsilon) \rfloor]$;
13. $\quad \mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds} \ \ \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1 - \epsilon) \rfloor + 1, n_s), ...$
14. $\quad 1, cap_{net} - \lfloor cap_{net} \times (1 - \epsilon) \rfloor))]$;
15. $\quad$ **for** $i = 1 : n_c$ **do**
16. $\quad\quad \mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
17. $\quad$ **end for**
18. **end for**

Algorithm B.14 Upper-Popularity-Bound collaborative non-group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_c, n_s, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. $\quad \mathbf{INT}_{rcmnd} = ...$
5. $\quad generateNGBRecommendations(n_c, n_s, \mathbf{INT}_{avail})$;
6. $\quad$ **for** $s = 1 : n_s$ **do**
7. $\quad\quad \mathbf{pop}^{rcmnd}_{services}[s] = \sum_{i:\mathbf{INT}_{rcmnd}[i,s] \ is \ not \ NaN} \mathbf{INT}_{rcmnd}[i, s]$;
8. $\quad\quad \mathbf{u}_{rcmnd}[s] = ...$
9. $\quad\quad \mathbf{pop}^{rcmnd}_{services}[s] + length(\underset{i}{argfind}(\mathbf{INT}_{rcmnd}[i, s] \ is \ NaN))$;
10. $\quad$ **end for**
11. $\quad [\sim, \mathbf{id}_{mps}] = sort(\mathbf{u}_{rcmnd}, -1)$;
12. $\quad \mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}]$;
13. $\quad$ **for** $i = 1 : n_c$ **do**
14. $\quad\quad \mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
15. $\quad$ **end for**
16. **end for**

Algorithm B.15 Generating Group-Based interest Recommendations using consumers collaboration (generateGBRecommendations)

1. **Inputs:** $n_c, id_c^f, id_c^l, n_s, \mathbf{INT}_{avail}$
2. **Output:** $\mathbf{INT}_{rcmnd}$
3. $\mathbf{J} = \mathbf{0}^{n_c \times n_c}$;
4. **for** $i = id_c^f : id_c^l$ **do**
5. **for** $j = id_c^f : id_c^l$ **do**
6. **if** $i = j$ **then**
7. **continue for**
8. **else if**
9. $\mathbf{J}[i, j] = ...$
10. $length(\underset{s}{argfind}(\mathbf{INT}_{avail}[i, s] = \mathbf{INT}_{avail}[j, s]))/n_s$;
11. **end if**
12. **end for**
13. **end for**
14. $\mathbf{INT}_{rcmnd} = \mathbf{INT}_{avail}$;
15. **for** $i = id_c^f : id_c^l$ **do**
16. **for** $s = 1 : n_s$ **do**
17. **if** $\mathbf{INT}_{rcmnd}[i, s]$ is NaN **then**
18. $\mathbf{INT}_{rcmnd}[i, s] \leftarrow \mathbf{INT}_{rcmnd}[\underset{j:j \in \{id_c^f : id_c^l\}}{argmax} (\mathbf{J}[i, j]), s]$;
19. **end if**
20. **end for**
21. **end for**

Algorithm B.16 Greedy collaborative group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_g, n_c, n_s, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $g = 1 : n_g$ **do**
5. $id_c^f = n_c/n_g \times (g-1) + 1;$
6. $id_c^l = n_c/n_g \times g;$
7. $\mathbf{INT}_{rcmnd} = ...$
8. $generateGBRecommendations(n_c, id_c^f, id_c^l, n_s, \mathbf{INT}_{avail});$
9. **for** $s = 1 : n_s$ **do**
10. $\mathbf{pop}_{services}^{rcmnd}[s] = ...$
11. $\sum_{i \in \{id_c^f : id_c^l\} : \mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i, s];$
12. **end for**
13. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}_{services}^{rcmnd}, -1);$
14. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}];$
15. **for** $i = id_c^f : id_c^l$ **do**
16. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}];$
17. **end for**
18. **end for**
19. **end for**

Algorithm B.17 ϵ-greedy collaborative group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_g, n_c, n_s, \mathbf{INT}_{avail}, ratio_{ui}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $g = 1 : n_g$ **do**
5. $id_c^f = n_c/n_g \times (g-1) + 1$;
6. $id_c^l = n_c/n_g \times g$;
7. $\mathbf{INT}_{rcmnd} = ...$
8. $generateGBRecommendations(n_c, id_c^f, id_c^l, n_s, \mathbf{INT}_{avail})$;
9. **for** $s = 1 : n_s$ **do**
10. $\mathbf{pop}_{services}^{rcmnd}[s] = ...$
11. $\sum_{i \in \{id_c^f : id_c^l\} : \mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i, s]$;
12. **end for**
13. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}_{services}^{rcmnd}, -1)$;
14. $\epsilon = ratio_{ui}$;
15. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1-\epsilon) \rfloor]$;
16. $\mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds}\ \ \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1-\epsilon) \rfloor + 1, n_s), ...$
17. $1, cap_{net} - \lfloor cap_{net} \times (1-\epsilon) \rfloor))]$;
18. **for** $i = id_c^f : id_c^l$ **do**
19. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}]$;
20. **end for**
21. **end for**
22. **end for**

Algorithm B.18 Decaying ϵ-greedy collaborative group-based recommender

Inputs: $n_{T_{rcmnd}}, n_g, n_c, n_s, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$

Output: $\mathbf{id}_{ds}$

for $t = 1 : n_{T_{rcmnd}}$ **do**

  for $g = 1 : n_g$ **do**

    $id_c^f = n_c/n_g \times (g-1) + 1;$

    $id_c^l = n_c/n_g \times g;$

    $\mathbf{INT}_{rcmnd} = ...$

    $generateGBRecommendations(n_c, id_c^f, id_c^l, n_s, \mathbf{INT}_{avail});$

    for $s = 1 : n_s$ **do**

      $\mathbf{pop}_{services}^{rcmnd}[s] = ...$

      $$\sum_{i \in \{id_c^f : id_c^l\} : \mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i, s];$$

    end for

    $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{pop}_{services}^{rcmnd}, -1);$

    $\epsilon = 1 - ...$

    $$length(\underset{(i \in \{id_c^f : id_c^l\}, s)}{argfind} (\mathbf{INT}_{avail}[i, s] \text{ is not } NaN))/(n_c/n_g \times n_s);$$

    $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : \lfloor cap_{net} \times (1-\epsilon) \rfloor];$

    $\mathbf{id}_{ds} \leftarrow [\mathbf{id}_{ds}\ \ \mathbf{id}_{mps}(randi((\lfloor cap_{net} \times (1-\epsilon) \rfloor + 1, n_s), ...$

    $1, cap_{net} - \lfloor cap_{net} \times (1-\epsilon) \rfloor))];$

    for $i = id_c^f : id_c^l$ **do**

      $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}];$

    end for

  end for

end for

Algorithm B.19 Upper-Popularity-Bound collaborative group-based recommender

1. **Inputs:** $n_{T_{rcmnd}}, n_g, n_c, n_s, \mathbf{INT}_{avail}, cap_{net}, \mathbf{INT}_{true}$
2. **Output:** $\mathbf{id}_{ds}$
3. **for** $t = 1 : n_{T_{rcmnd}}$ **do**
4. **for** $g = 1 : n_g$ **do**
5. $id_c^f = n_c/n_g \times (g-1) + 1;$
6. $id_c^l = n_c/n_g \times g;$
7. $\mathbf{INT}_{rcmnd} = ...$
8. $generateGBRecommendations(n_c, id_c^f, id_c^l, n_s, \mathbf{INT}_{avail});$
9. **for** $s = 1 : n_s$ **do**
10. $\mathbf{pop}_{services}^{rcmnd}[s] = ...$
11. $\sum_{i \in \{id_c^f : id_c^l\} : \mathbf{INT}_{rcmnd}[i,s] \text{ is not } NaN} \mathbf{INT}_{rcmnd}[i, s];$
12. $\mathbf{u}_{rcmnd}[s] = ...$
13. $\mathbf{pop}_{services}^{rcmnd}[s] + length(\underset{i \in \{id_c^f : id_c^l\}}{argfind}(\mathbf{INT}_{rcmnd}[i, s] \ is \ NaN));$
14. **end for**
15. $[\sim, \mathbf{id}_{mps}] = sort(\mathbf{u}_{rcmnd}, -1);$
16. $\mathbf{id}_{ds} = \mathbf{id}_{mps}[1 : cap_{net}];$
17. **for** $i = id_c^f : id_c^l$ **do**
18. $\mathbf{INT}_{avail}[i, \mathbf{id}_{ds}] = \mathbf{INT}_{true}[i, \mathbf{id}_{ds}];$
19. **end for**
20. **end for**
21. **end for**

Appendix C:
Content Routing Design Algorithms

Algorithm C.1 Extracting biggest cluster connectivities

1. **Inputs:** $t^{start}, t^{end}, \mathbf{C}, \mathbf{ID}^b_{nc}, c$
2. **Output:** $\mathbf{C}_b\{c\}$
3. $n_{T_{period}} = t^{end} - t^{start} + 1;$
4. **for** $t = 1 : n_{T_{period}}$ **do**
5. $\quad \mathbf{C}_b\{c\}[t, *, *] = \mathbf{C}[t + t^{start} - 1, \mathbf{ID}^b_{nc}\{c\}, \mathbf{ID}^b_{nc}\{c\}];$
6. **end for**

Algorithm C.2 Dividing data segments between nodes (*divideData*)

1. **Inputs:** $\mathbf{ID}^b_{nc}, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k$
2. **Output:** $\mathbf{n}_{ns}$
3. $n^b_n = length(\mathbf{ID}^b_{nc}\{c\});$
4. $n_{T_{period}} = t^{end} - t^{start} + 1;$
5. **for** $i = 1 : n^b_n$ **do**
6. $\quad \mathbf{n}_{ns}[i] = \sum_{t=1}^{n_{T_{period}}} \sum_{j=1}^{n^b_n} \mathbf{C}_b\{c\}[t, i, j];$
7. **end for**
8. $\mathbf{n}_{ns} \leftarrow \lfloor \mathbf{n}_{as}[k] \times \mathbf{n}_{ns} / (\sum_{i=1}^{n^b_n} \mathbf{n}_{ns}[i]) \rfloor;$

Algorithm C.3 Allocating data segments (*allocateData*)

1. **Inputs:** $n_n^b, \mathbf{n}_{ns}$
2. **Output: SA**
3. $\mathbf{SA} = \mathbf{0}^{n_n^b \times \sum_{i=1}^{n_n^b} \mathbf{n}_{ns}[i]}$;
4. **for** $i = 1 : n_n^b$ **do**
5. **if** $i = 1$ **then**
6. $\mathbf{SA}[i, 1 : \mathbf{n}_{ns}[i]] = \mathbf{1}^{1 \times length(\mathbf{n}_{ns}[i])}$;
7. $sa_l = \mathbf{n}_{ns}[i]$;
8. **else if**
9. $\mathbf{SA}[i, sa_l + 1 : sa_l + \mathbf{n}_{ns}[i]] = \mathbf{1}^{1 \times length(\mathbf{n}_{ns}[i])}$;
10. $sa_l \leftarrow sa_l + \mathbf{n}_{ns}[i]$;
11. **end if**
12. **end for**

Algorithm C.4 Computing node features (*computeFeatures*)

1. **Inputs:** $n_n^b, \mathbf{C}_b, c, t, \mathbf{SA}, \mathbf{n}_{ns}$
2. **Output:** $\mathbf{F}^n$
3. $\mathbf{F}^n = [\mathbf{f}_1^n \ \mathbf{f}_2^n \ \mathbf{f}_3^n \ \mathbf{f}_4^n]^{n_n^b \times 4} = \mathbf{0}^{n_n^b \times 4}$;
4. **for** $i = 1 : n_n^b$ **do**
5. $\quad \mathbf{f}_1^n[i] = \sum_{j=1}^{n_n^b} \mathbf{C}_b\{c\}[t, i, j]/n_n^b$;
6. $\quad$ **if** $\mathbf{f}_1^n[i] \neq 0$ **then**
7. $\quad\quad \mathbf{id}_s^n = \underset{j}{argfind}(\mathbf{SA}[i, j] = 1)$;
8. $\quad\quad \mathbf{id}_{ne}^n = \underset{j}{argfind}(\mathbf{C}_b\{c\}[t, i, j] = 1)$;
9. $\quad\quad \mathbf{id}_{nene}^n = \emptyset$;
10. $\quad\quad$ **for** $j = 1 : length(\mathbf{id}_{ne}^n)$ **do**
11. $\quad\quad\quad \mathbf{id}_{ms}^{ne} = \underset{k}{argfind}(\mathbf{SA}[\mathbf{id}_{ne}^n[j], k] = 0)$;
12. $\quad\quad\quad$ **if** $\mathbf{id}_{ms}^{ne} \neq \emptyset$ **then**
13. $\quad\quad\quad\quad \mathbf{f}_2^n[i] \leftarrow (\mathbf{f}_2^n[i] \times (j-1) + length(\mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne})/length(\mathbf{id}_{ms}^{ne}))/j$;
14. $\quad\quad\quad$ **end if**
15. $\quad\quad\quad$ **if** $j = 1$ **then**
16. $\quad\quad\quad\quad \mathbf{id}_{cms}^{ne} = \mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne}$;
17. $\quad\quad\quad$ **else if**
18. $\quad\quad\quad\quad \mathbf{id}_{cms}^{ne} \leftarrow \mathbf{id}_{cms}^{ne} \cap (\mathbf{id}_s^n \cap \mathbf{id}_{ms}^{ne})$;
19. $\quad\quad\quad$ **end if**
20. $\quad\quad\quad \mathbf{id}_{nene}^n \leftarrow \mathbf{id}_{nene}^n \cup \underset{k}{argfind}(\mathbf{C}_b\{c\}[t, \mathbf{id}_{ne}^n[j], k] = 1)$;
21. $\quad\quad$ **end for**
22. $\quad\quad \mathbf{f}_3^n[i] = length(\mathbf{id}_{cms}^{ne})/(\sum_{j=1}^{n_n^b} \mathbf{n}_{ns}[j])$;

23. $\mathbf{id}^n_{nene} \leftarrow \mathbf{id}^n_{nene} - (\mathbf{id}^n_{ne} \cup \{i\})$;

24. $\mathbf{f}^n_4[i] = 1 - length(\mathbf{id}^n_{nene})/n^b_n$;

25. **end if**

26. **end for**

Algorithm C.5 Controlling the broadcasting range of the transmitting node ($controlRange$)

---

1. **Inputs:** $\mathbf{id}^{tx}_{ne}, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}^b_{nc}, c, id^{tx}, t, t^{start}, \mathbf{STAT}^n$

2. **Outputs:** $\mathbf{id}^{tx}_{ne}, \mathbf{d}^{tx}_{ne}$

3. **for** $j = 1 : length(\mathbf{id}^{tx}_{ne})$ **do**

4. $\mathbf{d}^{tx}_{ne}[j] = distance(...$

5. $[\mathbf{LATS}[\mathbf{ID}^b_{nc}\{c\}[id^{tx}], t + t^{start} - 1]$ ...

6. $\mathbf{LONS}[\mathbf{ID}^b_{nc}\{c\}[id^{tx}], t + t^{start} - 1]], ...$

7. $[\mathbf{LATS}[\mathbf{ID}^b_{nc}\{c\}[\mathbf{id}^{tx}_{ne}[j]], t + t^{start} - 1]$ ...

8. $\mathbf{LONS}[\mathbf{ID}^b_{nc}\{c\}[\mathbf{id}^{tx}_{ne}[j]], t + t^{start} - 1]]);$

9. **end for**

10. $\mathbf{id}^{tx}_{ene} = \phi$;

11. **for** $j = 1 : length(\mathbf{id}^{tx}_{ne})$ **do**

12. **if** $(\mathbf{stat}^n_1[\mathbf{id}^{tx}_{ne}[j]] = 1) \vee (\mathbf{stat}^n_3[\mathbf{id}^{tx}_{ne}[j]] = 1)$ **then**

13. $\mathbf{id}^{tx}_{ene} \leftarrow \mathbf{id}^{tx}_{ene} \cup \underset{k}{argfind}(\mathbf{d}^{tx}_{ne}[k] \geq \mathbf{d}^{tx}_{ne}[j])$;

14. **end if**

15. **end for**

16. $\mathbf{id}^{tx}_{ne}[\mathbf{id}^{tx}_{ene}] = \phi$; $\mathbf{d}^{tx}_{ne}[\mathbf{id}^{tx}_{ene}] = \phi$;

Algorithm C.6 Targeting data segments for transmission ($targetSegments$)

1. **Inputs:** $\mathbf{SA}, id^{tx}, \mathbf{id}_{ne}^{tx}, \mathbf{STAT}^n$
2. **Output:** $\mathbf{id}_{ts}$
3. $\mathbf{id}_{ts} = \emptyset$;
4. $\mathbf{id}_s^{tx} = \underset{j}{argfind}(\mathbf{SA}[id^{tx}, j] = 1)$;
5. **for** $j = 1 : length(\mathbf{id}_{ne}^{tx})$ **do**
6. **if** $\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 0$ **then**
7. $\mathbf{id}_{ms}^{ne} = \underset{k}{argfind}(\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], k] = 0)$;
8. $\mathbf{id}_{ts} \leftarrow \mathbf{id}_{ts} \cup (\mathbf{id}_s^{tx} \cap \mathbf{id}_{ms}^{ne})$;
9. **end if**
10. **end for**

Algorithm C.7 Transmitting data (*transmitData*)

1. **Inputs:** $\mathbf{STAT}^n, id^{tx}, \mathbf{id}_{ne}^{tx}, \mathbf{SA}, \mathbf{id}_{ts}, \mathbf{A}, ...$
2. $t, t^{start}, \mathbf{d}_{ne}^{tx}, n_n^b$
3. **Outputs:** $\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n$
4. $\mathbf{stat}_1^n[id^{tx}] = 1;$
5. $\mathbf{pop}_{segs} = \sum_{i \in \{\mathbf{id}_{ne}^{tx} : \mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}] = \mathbf{0}\}} \mathbf{SA}[i, *];$
6. $id_{cs} = \mathbf{id}_{ts}[\underset{i \in \mathbf{id}_{ts}}{argmin}(\mathbf{pop}_{segs}[i])];$
7. **for** $j = 1 : length(\mathbf{id}_{ne}^{tx})$ **do**
8. **if** $(\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 0) \wedge (\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], id_{cs}] = 0)$ **then**
9. $\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}[j]] = 1;$
10. $\mathbf{stat}_3^n[\mathbf{id}_{ne}^{tx}[j]] = 1;$
11. $\mathbf{SA}[\mathbf{id}_{ne}^{tx}[j], id_{cs}] = 1;$
12. $\mathbf{A} \leftarrow (\mathbf{A}, [(t + t^{start} - 1) \;\; id^{tx} \;\; max(\mathbf{d}_{ne}^{tx}) \;\; id_{cs} \;\; \mathbf{0}^{1 \times n_n^b}]);$
13. $\mathbf{A}[length(\mathbf{A}), 4 + \mathbf{id}_{ne}^{tx}[j]] = 1;$
14. **end if**
15. **end for**
16. $\mathbf{stat}_2^n[\mathbf{id}_{ne}^{tx}] = \mathbf{1}^{length(\mathbf{id}_{ne}^{tx}) \times 1};$

Algorithm C.8 Following the segments exchange policy ($followPolicy$)

1. **Inputs:** $t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^b$
2. **Output:** $\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n$
3. $\mathbf{A} = \emptyset$; $n_{T_{period}} = t^{end} - t^{start} + 1$;
4. **for** $t = 1 : n_{T_{period}}$ **do**
5. $\quad \mathbf{STAT}^n = \mathbf{0}^{n_n^b \times 3}$;
6. $\quad \mathbf{F}^n = computeFeatures(n_n^b, \mathbf{C}_b, c, t, \mathbf{SA}, \mathbf{n}_{ns})$;
7. $\quad \mathbf{ut} = \mathbf{F}^n \times \omega$;
8. $\quad [\sim, \mathbf{id}_{tx}^{ord}] = sort(\mathbf{ut}, -1)$;
9. $\quad$ **for** $i = 1 : n_n^b$ **do**
10. $\quad\quad id^{tx} = \mathbf{id}_{tx}^{ord}[i]$;
11. $\quad\quad$ **if** $\mathbf{stat}_2^n[id^{tx}] = 1$ **then**
12. $\quad\quad\quad$ **continue for**
13. $\quad\quad$ **else if**
14. $\quad\quad\quad \mathbf{id}_{ne}^{tx} = \underset{j}{argfind}(\mathbf{C}_b\{c\}[t, id^{tx}, j] = 1)$;
15. $\quad\quad\quad$ **if** $(\forall j \in \mathbf{id}_{ne}^{tx}, \mathbf{stat}_2^n[j] = 1) \vee (\mathbf{id}_{ne}^{tx} = \emptyset)$ **then**
16. $\quad\quad\quad\quad$ **continue for**
17. $\quad\quad\quad$ **else if**
18. $\quad\quad\quad\quad [\mathbf{id}_{ne}^{tx}, \mathbf{d}_{ne}^{tx}] = ...$
19. $\quad\quad\quad\quad controlRange(\mathbf{id}_{ne}^{tx}, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^b, c, id^{tx}, t, t^{start}, \mathbf{STAT}^n)$;
20. $\quad\quad\quad\quad$ **if** $\mathbf{id}_{ne}^{tx} = \emptyset$ **then**
21. $\quad\quad\quad\quad\quad$ **continue for**
22. $\quad\quad\quad\quad$ **else if**
23. $\quad\quad\quad\quad\quad \mathbf{id}_{ts} = targetSegments(\mathbf{SA}, id^{tx}, \mathbf{id}_{ne}^{tx}, \mathbf{STAT}^n)$;

24. **if** $\mathbf{id}_{ts} = \emptyset$ **then**
25. **continue for**
26. **else if**
27. $[\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$
28. $transmitData(\mathbf{STAT}^n, id^{tx}, \mathbf{id}_{ne}^{tx}, \mathbf{SA}, \mathbf{id}_{ts}, \mathbf{A}, t, t^{start}, \mathbf{d}_{ne}^{tx}, n_n^b)$;
29. **end if**
30. **end if**
31. **end if**
32. **end if**
33. **end for**
34. **end for**

Algorithm C.9 Estimating the maximum number of distributable data segments given a minimum exchange ratio threshold

1. **Inputs:** $\mathbf{n}_{as}, \mathbf{ID}_{nc}^b, c, t^{start}, t^{end}, \mathbf{C}_b, n_n^b, ...$
2. $\mathbf{LATS}, \mathbf{LONS}, er_{min}$
3. **Output:** n_{as}^{max}
4. **for** $k = 1 : length(\mathbf{n}_{as})$ **do**
5. $\mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^b, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k)$;
6. $\mathbf{SA} = allocateData(n_n^b, \mathbf{n}_{ns})$;
7. %Specifying the naive policy weights:
8. $\omega = (10, 0, 0, 0)$;
9. $[\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$
10. $followPolicy(t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, ...$
11. $\mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^b)$;
12. **end for**

Algorithm C.10 Dividing and allocating the n_{as}^{max} data segments

1. **Inputs:** $\mathbf{n}_{as}, n_{as}^{max}, \mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, \mathbf{C}_b, n_n^b$
2. **Output:** $\mathbf{n}_{ns}, \mathbf{SA}$
3. $k = \underset{j}{argfind}(\mathbf{n}_{as}[j] = n_{as}^{max})$;
4. $\mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k)$;
5. $\mathbf{SA} = allocateData(n_n^b, \mathbf{n}_{ns})$;

Algorithm C.11 Generating initial regression data

1. **Inputs:** $n_i, \mathbf{n}_{as}, n_{as}^{max}, \mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, ...$
2. $\mathbf{C}_b, n_n^b, \mathbf{LATS}, \mathbf{LONS}$
3. **Output:** $\mathbf{DATA}_{reg}$
4. $\mathbf{DATA}_{reg} = \mathbf{0}^{n_i \times (4+1)}$;
5. **for** $i = 1 : n_i$ **do**
6. $\quad k = \underset{j}{argfind}(\mathbf{n}_{as}[j] = n_{as}^{max})$;
7. $\quad \mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k)$;
8. $\quad \mathbf{SA} = allocateData(n_n^b, \mathbf{n}_{ns})$;
9. $\quad$ %Specifying the policy weights randomly:
10. $\quad \omega = randi((-10, 10), 4, 1)$;
11. $\quad [\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$
12. $\quad followPolicy(t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^{b})$;
13. $\quad \mathbf{DATA}_{reg}[i, *] = [\omega' \ \ length(\mathbf{A})]$;
14. **end for**

Algorithm C.12 Bayesian optimization using Gaussian Processes regression

1. **Inputs:** $\mathbf{DATA}_{reg}, n_i, n_{rnd}, \mathbf{n}_{as}, n_{as}^{max}, \mathbf{ID}_{nc}^{b}, ...$
2. $c, t^{start}, t^{end}, \mathbf{C}_b, n_n^b, \mathbf{LATS}, \mathbf{LONS}$
3. **Output:** $\mathbf{DATA}_{gp}$
4. $\mathbf{DATA}_{gp} = \mathbf{0}^{(length(\mathbf{DATA}_{reg})+n_i)\times(4+1)}$;
5. $\mathbf{DATA}_{gp}[1 : length(\mathbf{DATA}_{reg}), *] = \mathbf{DATA}_{reg}$;
6. **for** $i = 1 : n_i$ **do**
7. $\quad \mathbf{Mdl}_{gp} = fitgp(\mathbf{DATA}_{gp}[*, 1:4], \mathbf{DATA}_{gp}[*, 4+1])$;
8. $\quad$ %Using UCB:
9. $\quad \mathbf{DATA}_{rnd} = \mathbf{0}^{n_{rnd}\times(4+1)}$;
10. $\quad$ **for** $j = 1 : n_{rnd}$ **do**
11. $\quad\quad$ %Specifying policy weights randomly:
12. $\quad\quad \omega = randi((-10, 10), 4, 1)$;
13. $\quad\quad \mathbf{DATA}_{rnd}[j, 1:4] = \omega'$;
14. $\quad\quad [\hat{\mathbf{DATA}}_{rnd}, \sigma] = predict(\mathbf{Mdl}_{gp}, \mathbf{DATA}_{rnd}[j, 1:4])$;
15. $\quad\quad \mathbf{DATA}_{rnd}[j, 4+1] = \parallel \hat{\mathbf{DATA}}_{rnd} + \sigma \parallel$;
16. $\quad$ **end for**
17. $\quad \mathbf{data}_{rnd} = \mathbf{DATA}_{rnd}[\underset{j}{argmax}(\mathbf{DATA}_{rnd}[j, 4+1]), *]$;
18. $\quad k = \underset{j}{argfind}(\mathbf{n}_{as}[j] = n_{as}^{max})$;
19. $\quad \mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k)$;
20. $\quad \mathbf{SA} = allocateData(n_n^b, \mathbf{n}_{ns})$;
21. $\quad$ %Specifying the current best policy weights:
22. $\quad \omega = \mathbf{data}_{rnd}[1, 1:4]'$;

23. $[\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$

24. $followPolicy(t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^b)$;

25. $\mathbf{DATA}_{gp}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \ \ length(\mathbf{A})]$;

26. **end for**

Algorithm C.13 Bayesian optimization using Random Forest regression

1. **Inputs:** $\mathbf{DATA}_{reg}, n_i, n_{rnd}, \mathbf{n}_{as}, n_{as}^{max}, \mathbf{ID}_{nc}^b, ...$

2. $c, t^{start}, t^{end}, \mathbf{C}_b, n_n^b, \mathbf{LATS}, \mathbf{LONS}$

3. **Output:** $\mathbf{DATA}_{rf}$

4. $\mathbf{DATA}_{rf} = \mathbf{0}^{(length(\mathbf{DATA}_{reg})+n_i)\times(4+1)}$;

5. $\mathbf{DATA}_{rf}[1 : length(\mathbf{DATA}_{reg}), *] = \mathbf{DATA}_{reg}$;

6. **for** $i = 1 : n_i$ **do**

7. **for** $j = 1 : 10$ **do**

8. $\mathbf{Mdl}_{tr}\{j\} = fittree(\mathbf{DATA}_{rf}[*, 1 : 4], \mathbf{DATA}_{rf}[*, 4 + 1], ...$

9. $'MinLeafSize', j)$;

10. **end for**

11. %Using UCB:

12. $\mathbf{DATA}_{rnd} = \mathbf{0}^{n_{rnd}\times(4+1)}$;

13. **for** $j = 1 : n_{rnd}$ **do**

14. %Specifying policy weights randomly:

15. $\omega = randi((-10, 10), 4, 1)$;

16. $\mathbf{DATA}_{rnd}[j, 1 : 4] = \omega'$;

17. **for** $k = 1 : 10$ **do**

18. $\hat{\mathbf{DATA}}_{rnd}[k] = predict(\mathbf{Mdl}_{tr}\{k\}, \mathbf{DATA}_{rnd}[j, 1 : 4])$;

19. **end for**

---

20. $\mu = (\sum_k \hat{\mathbf{DATA}}_{rnd}[k])/10$;

21. $\sigma = (\frac{1}{10} \times \sum_k (\hat{\mathbf{DATA}}_{rnd}[k])^2 - \mu^2)^{1/2}$;

22. $\mathbf{DATA}_{rnd}[j, 4+1] = \| \mu + \sigma \|$;

23. **end for**

24. $\mathbf{data}_{rnd} = \mathbf{DATA}_{rnd}[\underset{j}{argmax}(\mathbf{DATA}_{rnd}[j, 4+1]), *]$;

25. $k = \underset{j}{argfind}(\mathbf{n}_{as}[j] = n_{as}^{max})$;

26. $\mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^b, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k)$;

27. $\mathbf{SA} = allocateData(n_n^b, \mathbf{n}_{ns})$;

28. %Specifying the current best policy weights:

29. $\omega = \mathbf{data}_{rnd}[1, 1:4]'$;

30. $[\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$

31. $followPolicy(t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^b)$;

32. $\mathbf{DATA}_{rf}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \ \ length(\mathbf{A})]$;

33. **end for**

Algorithm C.14 Bayesian optimization using Bayesian Neural Network regression

1. **Inputs:** $\mathbf{DATA}_{reg}, n_i, n_{rnd}, \mathbf{n}_{as}, n_{as}^{max}, \mathbf{ID}_{nc}^b, ...$
2. $c, t^{start}, t^{end}, \mathbf{C}_b, n_n^b, \mathbf{LATS}, \mathbf{LONS}$
3. **Output:** $\mathbf{DATA}_{bnn}$
4. $\mathbf{DATA}_{bnn} = \mathbf{0}^{(length(\mathbf{DATA}_{reg})+n_i)\times(4+1)}$;
5. $\mathbf{DATA}_{bnn}[1 : length(\mathbf{DATA}_{reg}), *] = \mathbf{DATA}_{reg}$;
6. **for** $i = 1 : n_i$ **do**
7. $\mathbf{Mdl}_{nn} = fitnn([10, 10, 10])$;

8. $\mathbf{Mdl}_{nn}.layers\{1\}.transferFcn =' HyperbolicTangentSigmoid';$

9. $\mathbf{Mdl}_{nn}.layers\{2\}.transferFcn =' HyperbolicTangentSigmoid';$

10. $\mathbf{Mdl}_{nn}.layers\{3\}.transferFcn =' HyperbolicTangentSigmoid';$

11. $\mathbf{Mdl}_{nn}.layers\{4\}.transferFcn =' Linear';$

12. $\mathbf{Mdl}_{nn} = train(\mathbf{Mdl}_{nn}, \mathbf{DATA}_{bnn}[*, 1:4], \mathbf{DATA}_{bnn}[*, 4+1]);$

13. $\mathbf{Mdl}_{nn}.layerConnect(4,3) = 0;$

14. $\mathbf{Mdl}_{nn}.outputConnect(1,4) = 0;$

15. $\mathbf{Mdl}_{nn}.outputConnect(1,3) = 1;$

16. $\mathbf{Mdl}_{lin} = fitlinear(predict(\mathbf{Mdl}_{nn}, \mathbf{DATA}_{bnn}[*, 1:4]), \mathbf{DATA}_{bnn}[*, 4+1]);$

17. %Using UCB:

18. $\mathbf{DATA}_{rnd} = \mathbf{0}^{n_{rnd} \times (4+1)};$

19. **for** $j = 1 : n_{rnd}$ **do**

20. %Specifying policy weights randomly:

21. $\omega = randi((-10, 10), 4, 1);$

22. $\mathbf{DATA}_{rnd}[j, 1:4] = \omega';$

23. $\hat{\mathbf{DATA}}_{rnd} = predict(\mathbf{Mdl}_{lin}, predict(\mathbf{Mdl}_{nn}, \mathbf{DATA}_{rnd}[j, 1:4]));$

24. $\mathbf{DATA}_{rnd}[j, 4+1] =\parallel \hat{\mathbf{DATA}}_{rnd} + \mathbf{Mdl}_{lin}.rmse \parallel;$

25. **end for**

26. $\mathbf{data}_{rnd} = \mathbf{DATA}_{rnd}[\underset{j}{argmax}(\mathbf{DATA}_{rnd}[j, 4+1]), *];$

27. $k = \underset{j}{argfind}(\mathbf{n}_{as}[j] = n_{as}^{max});$

28. $\mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, \mathbf{C}_{b}, \mathbf{n}_{as}, k);$

29. $\mathbf{SA} = allocateData(n_{n}^{b}, \mathbf{n}_{ns});$

30. %Specifying the current best policy weights:

31. $\omega = \mathbf{data}_{rnd}[1, 1:4]'$;

32. $[\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$

33. $followPolicy(t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^b)$;

34. $\mathbf{DATA}_{bnn}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \ \ length(\mathbf{A})]$;

35. **end for**

Algorithm C.15 Bayesian optimization using batch-based Random Forest regression

1. **Inputs:** $\mathbf{DATA}_{reg}, n_i, n_{rnd}, \mathbf{n}_{as}, n_{as}^{max}, \mathbf{ID}_{nc}^{b}, ...$
2. $c, t^{start}, t^{end}, \mathbf{C}_b, n_n^b, \mathbf{LATS}, \mathbf{LONS}$
3. **Output:** $\mathbf{DATA}_{brf}$
4. $\mathbf{DATA}_{brf} = \mathbf{0}^{(length(\mathbf{DATA}_{reg})+n_i)\times(4+1)}$;
5. $\mathbf{DATA}_{brf}[1 : length(\mathbf{DATA}_{reg}), *] = \mathbf{DATA}_{reg}$;
6. **for** $i = 1 : n_i$ **do**
7. **for** $j = 1 : 10$ **do**
8. $\mathbf{Mdl}_{btr}\{j\} = ...$
9. $fittree(\mathbf{DATA}_{brf}[i : i + 99, 1 : 4], \mathbf{DATA}_{brf}[i : i + 99, 4 + 1], ...$
10. $'MinLeafSize', j)$;
11. **end for**
12. %Using UCB:
13. $\mathbf{DATA}_{rnd} = \mathbf{0}^{n_{rnd}\times(4+1)}$;
14. **for** $j = 1 : n_{rnd}$ **do**
15. %Specifying policy weights randomly within batch ω constraints:
16. **for** $k = 1 : length(\omega)$ **do**
17. $\mu_\omega = (\sum_{m=i}^{i+99} \mathbf{DATA}_{brf}[m, k])/100$;
18. $\sigma_\omega = ((\sum_{m=i}^{i+99} (\mathbf{DATA}_{brf}[m, k] - \mu_\omega)^2)/100)^{1/2}$;
19. $\omega_{min} =\| \ max(\mu_\omega - \sigma_\omega, -10) \ \|$;
20. $\omega_{max} =\| \ min(\mu_\omega + \sigma_\omega, 10) \ \|$;
21. $\omega[k] = randi((\omega_{min}, \omega_{max}), 1, 1)$;
22. **end for**
23. $\mathbf{DATA}_{rnd}[j, 1 : 4] = \omega'$;

24. **for** $k = 1 : 10$ **do**

25. $\quad \hat{\mathbf{DATA}}_{rnd}[k] = predict(\mathbf{Mdl}_{btr}\{k\}, \mathbf{DATA}_{rnd}[j, 1:4])$;

26. **end for**

27. $\mu = (\sum_{k} \hat{\mathbf{DATA}}_{rnd}[k])/10$;

28. $\sigma = (\frac{1}{10} \times \sum_{k} (\hat{\mathbf{DATA}}_{rnd}[k])^2 - \mu^2)^{1/2}$;

29. $\mathbf{DATA}_{rnd}[j, 4+1] = \| \mu + \sigma \|$;

30. **end for**

31. $\mathbf{data}_{rnd} = \mathbf{DATA}_{rnd}[\underset{j}{argmax}(\mathbf{DATA}_{rnd}[j, 4+1]), *]$;

32. $k = \underset{j}{argfind}(\mathbf{n}_{as}[j] = n_{as}^{max})$;

33. $\mathbf{n}_{ns} = divideData(\mathbf{ID}_{nc}^{b}, c, t^{start}, t^{end}, \mathbf{C}_b, \mathbf{n}_{as}, k)$;

34. $\mathbf{SA} = allocateData(n_n^b, \mathbf{n}_{ns})$;

35. %Specifying the current best policy weights:

36. $\omega = \mathbf{data}_{rnd}[1, 1:4]'$;

37. $[\mathbf{SA}, \mathbf{A}, \mathbf{STAT}^n] = ...$

38. $followPolicy(t^{start}, t^{end}, n_n^b, \mathbf{C}_b, c, \mathbf{SA}, \mathbf{n}_{ns}, \omega, \mathbf{LATS}, \mathbf{LONS}, \mathbf{ID}_{nc}^{b})$;

39. $\mathbf{DATA}_{brf}[length(\mathbf{DATA}_{reg}) + i, *] = [\omega' \;\; length(\mathbf{A})]$;

40. **end for**