UNIVERSITY OF CALIFORNIA

Los Angeles

Cooperative Channel Sensing, Relaying and Computing in UAV and Vehicular Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical and Computer Engineering

by

Enes Krijestorac

2024

ABSTRACT OF THE DISSERTATION

Cooperative Channel Sensing, Relaying and Computing in UAV and Vehicular Networks

by

Enes Krijestorac
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2024
Professor Danijela Cabric, Chair

Mobile devices generate an enormous amount of data traffic to satisfy their computing and communications needs. To meet these demands, mobile network operators frequently need to expand their capacity, which entails significant capital costs and increased energy consumption. Motivated by this, we seek to develop cooperative systems that will bring higher communications speeds and larger computing power to mobile devices without relying on mobile network infrastructure.

In recent years, unmanned aerial vehicle (UAV) technology has garnered interest for its potential use as a communications enabler. Swarms of UAVs can be deployed as temporary relays to meet short term but high intensity communication demands from mobile users. UAV swarms can coordinate their placement to improve the capacity on the fronthaul link between users and UAVs. Algorithms for optimal placement often rely on the knowledge of channel gain across space. Hence, we developed deep learning methods for channel gain prediction across space based on measurements collected by the UAVs and 3D maps of the environment. In line with this, we also developed methods to design UAV flying paths for optimal measurement collection such that the accuracy of channel gain prediction is maximized under constraints on the distance traveled by the UAVs. Additionally, we develop a reinforcement-learning based approach that controls a UAV to directly improve the fronthaul link without relying on channel gain knowledge across space.

With the proliferation of intelligent vehicles, there is an increasing number of computationally demanding computer applications appearing in vehicular environments. Providing the computational resources to meet the demands of such applications is a critical problem. In this work, we consider a cooperative computing paradigm between intelligent vehicles of similar computing power to enable emerging vehicular applications. Vehicles cooperate with each other over vehicle-to-vehicle networks to form vehicular micro clouds that can complete computationally intensive tasks without relying on cloud or edge computing. We developed optimized resource assignment and scheduling algorithms that efficiently use vehicular computing resources for computation in emerging vehicular applications. Our proposed approaches adapt to link quality changes between vehicles and prevent congestion in vehicular networks, even in the presence of incumbent interference.

To Yutong

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Danijela Cabric for giving me the opportunity to work on challenging problems in a variety of different domains. Her continued support and constructive criticism has allowed me to make this far. I would also like to thank the members of my committee Prof. Christina Fragouli, Prof. Gregory Pottie, and Prof. Paulo Tabuada for their helpful comments.

I would like to thank the following collaborators that helped me in my PhD research: Dr. Samer Hanna, Dr. Ghaith Hattab, Ruifu Li and Prof. Shamik Sarkar. My research has been significantly shaped by collaborations with researchers in the industry and I would like to give a special thanks to Dr. Onur Altintas, Dr. Seyhan Ucar and Dr. Takamasa Higuchi.

I am also grateful that I had the opportunity to work on many other interesting research projects while at CORES lab with other collaborators and, in particular, I would like to thank: Samurdhi Karunaratne, Dr. Agon Memedi, Dr. Hazem Sallouha and Tianyi Zhao. My PhD experience has been memorable thanks to other amazing lab mates whom I would like to thank: Yen-Chin Wang, Benjamin Domae, Dr. Veljko Boljanovic, Aditya Wadaskar, Manali Sharma and Ibrahim Pehlivan.

Most of all, my deepest gratitude is towards my family for being supportive and encouraging during graduate studies. My PhD journey has had its fair share of ups and downs, but throughout it my wife Yutong has helped me persist and has brought joy and tranquility into my life. My parents Mersija and Hamed have enabled me to have a chance at the PhD and have supported me throughout it as well. Likewise, I have always been supported by my older brother Elvis.

VITA

2014–2018	B.S. in Electrical Engineering, New York University Abu Dhabi, Abu Dhabi, UAE.
2018–2021	M.S. in Electrical and Computer Engineering, University of California, Los Angeles, Los Angeles, USA.
2018-2023	Graduate Student Researcher and Teaching Assistant, UCLA.
2020	Summer Research Intern, Toyota Motor North America, Mountain View, CA, USA.
2021	Summer Engineering Intern, Qualcomm, San Diego, CA, USA.
2022	Summer Engineering Intern, Apple, Cupertino, CA, USA.

SELECTED PUBLICATIONS

[1] E. Krijestorac, R. Li, D. Cabric, J. McGraw, P. Powers, J. Vitaz, M. Salpukas, D. Eustice, T. Allen, "Machine Learning Assisted Computationally Efficient Target Detection and Tracking in Massive Fully Digital Phased Arrays," in *IEEE Transactions on Radar Systems*, vol. 1, pp. 353 - 367, August 2023.

[2] E. Krijestorac, H. Sallouha, S. Sarkar, D. Cabric, "Agile Radio Map Prediction Using Deep Learning," in ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2023, pp. 1–2. doi:10.1109/ICASSP49357.2023.10096546.

[3] E. Krijestorac, G. Hattab, P. Popovski, D. Cabric, “Multiband Massive IoT: A Learning Approach to Infrastructure Deployment,” in *IEEE Transactions on Wireless Communications*, vol. 21, pp. 10300 - 10314, December 2022.

[4] E. Krijestorac, S. Hanna, D. Cabric, “Spatial Signal Strength Prediction using 3D Maps and Deep Learning,” in ICC 2021 - IEEE International Conference on Communications, June 2021, pp. 1–6. doi:10.1109/ICC42927.2021.9500970.

[5] E. Krijestorac, A. Memedi, T. Higuchi, S. Ucar, O. Altintas, D. Cabric, “Hybrid Vehicular and Cloud Distributed Computing: A Case for Cooperative Perception,” in GLOBECOM 2020 - 2020 IEEE Global Communications Conference, January 2021, pp. 1–6. doi:10.1109/GLOBECOM42002.2020.9322247.

[6] E. Krijestorac, S. Hanna, D. Cabric, “UAV Access Point Placement for Connectivity to a User with Unknown Location Using Deep RL,” in 2019 IEEE Globecom Workshops (GC Wkshps), January 2019, pp. 1–6. doi:10.1109/GCWkshps45667.2019.9024644.

CHAPTER 1

Introduction

1.1 Motivation

The increasing volume of smart devices and IoT modules is generating an enormous amount of data traffic to satisfy computing and communications needs, putting an additional burden on existing network infrastructure. Mobile network operators frequently need to expand the number of active base stations to satisfy the increasing user demand. However, deployment of more base stations entails significant capital cost and increases energy consumption. Motivated by this, we seek to develop cooperative systems that will bring higher communications speeds and larger computing power to mobile devices without relying on expansion of network infrastructure.

Unmanned aerial vehicles (UAVs) or drones have received widespread attention in military and, more recently, civilian applications. Their mobility and low cost of operation makes them suitable for many civilian purposes, such as package delivery, environment mapping and emergency assistance [Ins22].

In recent years, UAV technology has garnered interest from the wireless communications industry and the academic community for their potential use as communication enablers [MSB19b]. UAVs as communications enablers can be used to meet the requirements of 5G and future communications systems in challenging scenarios. In particular, UAV swarms can be effective due to their ability to serve as relays for many users simultaneously and also coordinate their operation to improve the capacity on the fronthaul side (the link between users and UAVs) and backhaul side (the link between UAVs and the core network). Use case scenarios include replacing damaged base stations and events that can increase traffic intensity in certain areas, such as concerts, mass gatherings and traffic jams. In such cases, UAVs are faster to deploy and are more economic than fixed infrastructure. Additionally, UAVs can be used to provide coverage to areas with sporadic traffic such as remote areas with occasional activity or IoT networks that have occasional traffic that they need to offload to the core network. To meet the data rate demands of modern mobile devices, it is essential to optimize the placement of UAVs to maximize the capacity of fronthaul and backhaul links.

With the proliferation of intelligent vehicles, there is an increasing number of computationally demanding computer applications appearing in vehicular environments. Providing the computational resources to meet the demands of such applications is a critical problem. Mobile devices with limited resources have often relied on mobile network infrastructure to offload complex computation to cloud or edge servers with more powerful computing resources. The main difference between cloud and edge computing is the proximity of the servers to mobile users, with cloud servers being located on the Internet, whereas edge servers are located closer to the users and perform various tasks in conjunction with cloud data centers. However, computational offloading to the cloud or edge can face significant difficulties due to reliance on wireless infrastructure, such as high latency and low coverage, which makes them unsuitable for delay-sensitive applications. There is a growing number of intelligent vehicles on the road, each of which has an increasingly rich amount of computational resources as well as the sensing and communication capabilities. These vehicles can cooperate with each other over vehicle-to-vehicle (V2V) networks to from vehicular micro clouds (MCs) [HJD17]. Vehicles in a MC can cooperatively perform computationally intensive tasks to minimize or fully remove the need to offload to cloud or edge servers. However, vehicular computing resources are smaller than those available to cloud or edge servers, therefore it is necessary to effectively manage them to maximize their utility. Furthermore, computational offloading over V2V channels has a unique set of challenges such as fluctuating link capacities and distributed channel access, which must be addressed to enable cooperative computing.

1.2 Challenges and objectives

1.2.1 UAW Swarms for Communications

We focus on improving the fronthaul capacity, which depends on the signal-to-noise ratio (SNR) between ground users and the UAVs. Hence, by optimized placement of UAVs we can improve the SNR and thereby the capacity.

Spatial channel gain prediction Most state-of-the-art solutions for optimal placement of UAVs rely on analytical models of the channel, since analytical models are computationally and mathematically simple while keeping satisfactory accuracy. However, they are accurate only on average across a predefined set of propagation environments for which the models were developed. The rigid format of analytical models does not allow adaptation to the specific environments they are applied to. We address the limitations of algorithms previously developed for this task by proposing a new deep learning-based channel prediction algorithm. Our approach adapts to the environment by relying on the knowledge of the 3D map of the propagation environment and on measurements collected by the UAV. Complimentary to this approach, it is necessary to collect informative measurements of the channel gain between users and UAVs across space. Due to energy and time constraints, UAVs are constrained in how much distance they can cover. Therefore, it is necessary to optimally design their trajectories to collect most informative measurements of the channel gain under constraints on the length of their trajectories.

Simultaneous channel gain learning and placement optimization The traversal of UAVs to collect measurements for channel gain prediction can lead them to a placement that is far away away from a placement that maximizes the capacity of the fronthaul links. After collecting sufficient measurements for channel gain prediction, the UAVs may still need to traverse a significant distance to optimize the capacity. Hence, there is a need to develop approaches for control of UAVs that enable them to simultaneously learn the channel gain across space while simultaneously moving them towards a placement that improves the capacity of the fronthaul links.

1.2.2 Cooperative Computing in Vehicular Micro Clouds

We aim to develop and analyze systems for cooperative computing in a vehicular MC to support emerging computationally-intensive and delay-constrained applications such as co-operative perception [CZX22], augmented reality, driving assistance and navigation.

Resource management and task scheduling for cooperative computing To enable cooperative computing in a vehicular micro cloud for support of emerging applications, we need to address the following challenges. First, we need to develop approaches for optimal use of computing resources and task scheduling in the micro cloud. We assume that the V2V communication is achieved with hardware from one of the IEEE 802.11 standards. Due to the their mobility, achievable data rates between vehicles fluctuate constantly. Our resource assignment and scheduling framework must therefore adapt to the current and future achievable data rates between vehicles. Furthermore, when performing task scheduling it is essential to not exceed the capacity limits of the wireless channel and cause congestion, which can severely delay the delivery of offloaded tasks.

Hybrid cooperative and edge computing Cooperative computing in a vehicular micro cloud can be combined with edge computing to further improve the quality of service of vehicular applications. We assume that communication between edge servers and vehicles is accomplished using cellular networks. In the hybrid computing case, it is also necessary to devise methods for resource assignment and task scheduling under the additional challenges of computational offloading via cellular networks. Furthermore, it is of interest to determine in which scenarios does hybrid offloading offer benefits compared to cooperative or edge computing.

1.3 Contributions

1.3.1 UAW Swarms for Communications

Spatial channel gain prediction We developed novel active prediction approaches which consist of both methods for UAV path planning for optimal measurement collection and methods for prediction of channel gain across space based on the collected measurements. First, we developed an active deep learning channel gain prediction approach that relies on measurements collected by multiple UAVs and a 3D map of the environment, but does not rely on ground user location. The developed approach consists of a deep learning prediction method that also provide probabilistic channel gain prediction across space and a deep reinforcement learning method that designs UAV paths for measurement collection under constraints on the lengths of their trajectories. Second, we developed an active channel gain prediction algorithm using multiple UAVs based on Kriging spatial interpolation. While Kriging interpolation has been extensively used for channel gain prediction, no methods for measurement collection using multiple UAVs have been proposed. This method is suitable for channel gain prediction when the ground user location is available and a 3D map of the environment is not. Furthermore, this method does not require extensive training compared to our proposed deep learning active prediction approach.

Simultaneous channel gain learning and placement optimization Assuming that each UAV is associated to serve a single ground user, we developed a deep reinforcement learning approach that controls the UAV to collect channel measurements while simultaneously moving it towards improving the capacity of the fronthaul link. The proposed algorithm relies on the knowledge of 3D map of environment and does not need the knowledge of the user location. Deep reinforcement learning allows us to use 3D maps alongside channel gain measurements to predict the optimal direction of motion to maximize the fronthaul capacity under constraints on maximum traversed distance by the UAV.

1.3.2 Cooperative Computing in Vehicular Micro Clouds

Resource management and task scheduling We first develop approaches based on mixed integer linear programming for assignment of computing resources such that the quality of service of vehicular applications is maximized. The proposed algorithms adapt to the changes in communication links due to vehicular mobility. Moreover, assuming that wireless technology from a 802.11 communication standard is used by vehicles, the proposed resource assignment and scheduling approaches ensure that channel congestion does not occur due to task offloading between the vehicles. Based on our mixed integer programming solutions, we also develop approximate resource assignment and scheduling methods that run in polynomial time and are feasible to use in more challenging cases such as when the number of vehicles participating in cooperative computing is high. Finally, given optimal resource assignment and scheduling methods, we provide answers on what are the feasible gains of cooperative computing compared to non-cooperative computing in terms of quality of service and as a function of different factors including computing task features, traffic characteristics, vehicular computing resources and incumbent wireless transmitter activity.

Hybrid cooperative and edge computing We develop a resource assignment and scheduling algorithm for hybrid processing of computing tasks for cooperative perception that maximizes the rate at which frames are processed, while ensuring that results are delivered within a deadline and also meeting cellular and V2V communication constraints. This approach is developed under simplified models of vehicle-to-vehicle communications compared to the methods we described in the previous paragraph. Using this approach, we quantify the benefits of hybrid computing for cooperative perception and compare them against edge and cooperative computing.

1.4 Thesis organization

The rest of this thesis is organized as follows:

- Chapter 2: We present our active prediction approaches for prediction of channel gain across space. This chapter is based on [KC23], which is an extension of the work published in [KHC21].
- Chapter 3: We present our reinforcement learning based approach for simultaneous channel gain learning and UAV placement optimization. This chapter is a based on our previous publication in [KHC19].
- Chapter 4: We present our work on cooperative computing in vehicular micro clouds for support of emerging vehicular applications. This chapter is based on [KLH23].
- Chapter 5: We present our resource assignment and scheduling algorithm for hybrid computing of tasks in cooperative perception. This chapter is based on our previous publication in [KMH20].
- Chapter 6: We summarize the research contributions and outline future research directions.

CHAPTER 2

Spatial Channel Gain Prediction Using a Swarm of Unmanned Aerial Vehicles

Prediction of wireless CG across space is a necessary tool for many important wireless network design problems. In this chapter, we develop prediction methods that use environment-specific features, namely building maps and CG measurements, to achieve a high prediction accuracy. We assume that measurements are collected using a swarm of coordinated unmanned aerial vehicles (UAVs). We develop novel active prediction approaches which consist of both methods for UAV path planning for optimal measurement collection and methods for prediction of CG across space based on the collected measurements. We propose two active prediction approaches based on DL and Kriging interpolation. The first approach does not rely on the location of the transmitter and utilizes 3D maps to compensate for the lack of it. We utilize DL to incorporate 3D maps into prediction and reinforcement learning for optimal path planning for the UAVs based on DL prediction. The second active prediction approach is based on Kriging interpolation, which requires known transmitter location and cannot utilize 3D maps. We train and evaluate the two proposed approaches in a ray-tracing-based channel simulator. Using simulations, we demonstrate the importance of active prediction compared to prediction based on randomly collected measurements of channel gain. Furthermore, we show that using DL and 3D maps, we can achieve high prediction accuracy even without knowing the transmitter location. We also demonstrate the importance of coordinated path planning for active prediction when using multiples UAVs compared to UAVs collecting measurements independently in a greedy manner.

2.1 Introduction

The use of unmanned aerial vehicles (UAVs) as communication enablers has received a lot of attention in recent years, in part, due to their ability to optimize their placement in order to increase CG to the ground devices they are serving [MSB19a]. Algorithms for optimal placement often rely on the knowledge of the CG across space, which can be obtained via direct measurements or via some type of predictive model. Prediction of the CG across space is also necessary for other important wireless network design problems such as wireless network infrastructure planning [WXC07], network resource allocation and spectrum sharing [ARV12].

Commonly used statistical approaches for CG prediction rely on the assumption that the channel can be modeled based on features that are not environment-specific, such as distance between radio devices, altitude of radio devices and others. Some examples of statistical UAV communications channel models are provided in [KCZ18]. The advantage of using statistical models for spatial channel prediction lies in their computational simplicity and in their suitability for mathematical analysis. However, such prediction approaches lack the ability to adapt to a given environment which limits the accuracy of their prediction. The key reason for this is that the local environment blockage and scattering may cause the channel to sharply differ from the predictions drawn from simple statistical features such as distance and altitude.

In order to circumvent the limitations of statistical approaches, methods that utilize environment adaptive features such as 3D maps or CG measurements can be utilized. These approaches have the advantage of adapting to the propagation characteristics of the current environment. An example of such methods for predicting the wireless channel is ray-tracing. Ray-tracing can be used to accurately simulate the wireless channel for a specific environment. However, it has the disadvantages of requiring a precise 3D map of the environment, exact transmitter location and is highly computationally complex. An alternative set of methods for environment adaptive spatial prediction are spatial interpolation methods often used in geostatistics, such as Kriging interpolation and inverse distance weighting. These methods can be used to predict the CG across an area of interest given a set of sparse measurements [AAG11].

However, majority of current research on environment adaptive CG prediction based on measurements does not consider the methods according to which measurements are collected. For applications such as UAV-enabled communications, it is possible to utilize one or multiple UAVs to collect CG measurements for CG prediction. For other applications such as network planning, it is also possible to use UAVs or other types of vehicles to collect measurements. The design of paths according to which measurements are collected can significantly influence the accuracy of predicted CG. Despite their potential importance, methods for path planning for measurement collection have scarcely been considered in the prior literature. Therefore, the first goal of this chapter is to develop CG prediction approaches that include methods for measurement collection, which we refer to as active CG prediction.

Additionally, spatial interpolation algorithms and ray-tracing CG prediction methods rely on exact transmitter location knowledge. Obtaining the exact transmitter location may not always be possible for several reasons: GPS operation is not always reliable in urban environments, the transmitter equipment may not have localization capabilities, or the transmitter location cannot be shared due to privacy or security reasons. Hence, our second goal is to develop active CG prediction approaches that can operate without the knowledge of the exact transmitter location. At the same time, we seek to achieve higher or similar level of accuracy using our location-free CG prediction methods compared to traditional spatial interpolation methods, which rely on known transmitter location.

Guided by these objectives, we propose two new active channel gain prediction solutions. Our contributions can be summarized as follows:

- First, we developed an active DL CG prediction approach that relies on measurements collected by multiple UAVs and a 3D map of the environment, but does not rely on transmitter location. The 3D maps enable highly accurate CG prediction compared to spatial interpolation using only measurements and without transmitter location. The developed approach consists of a DL prediction method that provides probabilistic CG prediction across space and a deep RL based method that designs UAV paths for measurement collection for multiple UAVs based on the DL predictions. This active approach is trained and evaluated in a ray-tracing-based wireless channel simulator.

- Second, we developed an active CG prediction algorithm using multiple UAVs based on Kriging spatial interpolation. While Kriging interpolation has been extensively used for CG prediction, no methods for active prediction using multiple UAVs have been proposed. This method is suitable for CG prediction when the transmitter location is available and a 3D map of the environment is not. Furthermore, this method does not require extensive training compared to our proposed DL active prediction approach. We also evaluated the proposed active Kriging prediction approach in a ray-tracing-based simulator.

The chapter is organized as follows. In Sec. 2.2, we review and compare our work to the existing literature. In Sec. 2.3, we provide a detailed description of the targeted application scenarios and the modeling assumptions. In Sec. 2.4, we introduce the proposed active DL CG prediction approach. In sections 2.5 and 2.6, we go into more details on its two main components: probabilistic channel gain prediction and reinforcement learning path planner. In Sec. 2.7 we introduce and explain the Kriging based active prediction approach. In Sec. 2.8, we describe the simulation environment, benchmarks and obtained simulation results. Finally, in Sec. 2.9, we summarize the findings of the chapter.

2.2 Related Work

The most common approaches for channel prediction are adopted from the field of spatial interpolation [AAG11, CBS18, HB12, BJF16]. Among the interpolation methods, the most commonly used ones are inverse distance weighting (IDW), gradient plus inverse distance squared (GIDS) and Kriging interpolation. Algorithms based on Kriging interpolation rely on the location of the transmitter, while IDW or GIDS based algorithms normally do not, which comes at the cost of lower prediction accuracy compared to Kriging. Other stand-alone approaches based on Gaussian modeling of shadowing component of CG were proposed in [LKG17] and [MM12], but these approaches also rely on knowing the transmitter location. More recently, in [ZFW20], thin plate splines (TPS) interpolation method and coupled block-term tensor decomposition methods were used for CG prediction. These methods also do not rely on transmitter location. However, the spatial interpolation based approaches in [AAG11, CBS18, HB12, BJF16] and in [LKG17, MM12, ZFW20] do not consider optimal measurement collection methods. Furthermore, these methods are not able to incorporate complex inputs such as topography maps or building maps into their prediction.

DL based approaches have been developed for spatial gain prediction. These approaches are usually developed to outperform spatial interpolation methods in terms of prediction accuracy or to outperform ray tracing approaches in terms of computational complexity, such as in [LYK21, KSS23], where spatial gain prediction is performed assuming a known 3D map of the environment and location of the transmitter. In [HXS20], generative adversarial neural networks are used for spatial prediction based on measurements, while the authors in [TR20] use deep completion auto-encoders for the same task. The approaches in [HXS20] and [TR20] do not assume to know the user location. In our prior work [KHC21], we also utilized 3D maps and signal strength measurements for probabilistic CG prediction using DL and this chapter builds upon that work. However, like the approaches in [HXS20, TR20], we have not explored optimal CG measurement collection strategies.

Optimal measurement collection methods for spatial CG prediction have rarely been considered in the prior literature. In [AMB22], the authors consider the problem of cellular base station gain prediction based on crowd-sourced measurements from end-user devices. In this case, the objective is to devise a strategy for optimal selection and utilization of crowd-sourced measurements but not to directly control where the measurements are collected. Optimal path planning for measurement collection using UAVs has been considered in [SRC22]. However, in this work, path planning for measurement collection using only a single UAV is considered and machine-learning-based path planning is not considered. Given that the UAV technology is currently mature and widely accessible, it is reasonable to deploy swarms of UAVs for CG prediction for one or more transmitters. Therefore, we develop path planning methods for multiple coordinated UAVs. Furthermore, it is necessary to consider learning-based methods for path planning for measurement collection due to the recent interest in deep-learning-based CG predictors. Since deep neural networks are black box models, it is difficult to design optimal analytical approaches for path planning, hence learning-based approaches can be used to better accomplish this task. In [LLW23], reinforcement learning was applied for path planning for measurement collection for TPS interpolation. However, this approach was developed for control of a single UAV, so it would not be applicable to coordinated control of multiple UAVs. Furthermore, since it was trained and evaluated for prediction based on TPS interpolation, it is not clear if it would extend to DL CG prediction approaches.

2.3 System model and objectives

2.3.1 Environment and transmitters

We consider a rectangular urban area of interest (AoI) of width w, length l and height h, for which we have a database of major buildings and objects that can be used to construct a 3D map of the environment, which is denoted by $\mathbf{M}$. We discretize the AoI into a uniformly-

Table 2.1: A summary of important notation used in the chapter and their meanings.

Variables/Parameters	
Environment and transmitters	
l, w, h	Dimensions of the AoI
$\mathbf{M}$	3D Map of the AoI
d	Grid spacing
$\mathcal{Q}$	Discrete set of locations in AoI
$\mathbf{q}_j$	A location in $\mathcal{Q}$
h_p	Altitude where CG is predicted
$\mathcal{Q}_P$	Set of locations in $\mathcal{Q}$ where CG is predicted
$\mathbf{x}_k$	Vector of CGs at locations$\mathcal{Q}_P$ for transmitter k
$\mathbf{z}$	Binary vector denoting outdoor locations in$\mathcal{Q}_P$
UAV swarm and CG measurements	
N	Number of UAVs
h_{UAV}	Altitude where UAVs are moving
t	Time index
$\mathbf{P}_t$	Locations of all UAVs at time t
$\mathbf{p}_{t,n}$	Location of UAV n at time t
$\mathcal{V}_{t_1:t_2}$	Set of locations visited by UAVs between t_1 and t_2
$\mathbf{y}_{k,t_1:t_2}$	Vector of CG measurements collected from t_1 to t_2
T	Number of steps UAVs move to collect measurements
U	Number of possible motion actions per UAV
$\mathbf{u}_t$	Vector of motion actions of all N UAVs at time t
Reinforcement learning	
$\mathcal{S}$	Set of states in an MDP
$\mathcal{A}$	Set of actions in an MDP
$\mathbf{s}_t$	State in an MDP
$\mathbf{a}_t$	Action in an MDP
r	Reward function in an MDP
γ	Discount factor in an MDP
π	Policy function in an MDP
ϵ	Exploration probability
M	Multi-step learning length
τ_{DQN}	Target update period

spaced 3D grid of locations $\mathcal{Q} = \left\{\mathbf{q}_1, \ldots, \mathbf{q}_{|\mathcal{Q}|}\right\}$ with spacing d. Each location $\mathbf{q}_j \in \mathcal{Q}$ is coordinate vector $\mathbf{q}_j = [q_{j,x}, q_{j,y}, q_{j,z}]^T$. The number of locations in $\mathcal{Q}$ is $|\mathcal{Q}| = \frac{l}{d} \times \frac{w}{d} \times \frac{h}{d}$.

Throughout this chapter, we assume that the goal is to estimate the CG in the AoI for a single transmitter k. The transmitter is assumed to be stationary with a location $\mathbf{w}_{TX}$. The location $\mathbf{w}_{TX}$ could be known or unknown, and we will propose approaches that will handle both of these cases. Furthermore, we assume that CG is predicted for a set of points in $\mathcal{Q}_P = \left\{\tilde{\mathbf{q}}_1, \ldots, \tilde{\mathbf{q}}_{|\mathcal{Q}_P|}\right\} \subset \mathcal{Q}$, which have an altitude h_P. The number of locations in $\mathcal{Q}_P$ is $|\mathcal{Q}_{\mathcal{P}}| = \frac{l}{d} \times \frac{w}{d}$. For the purposes of this chapter, the CG is predicted for a constant altitude to reduce the training time and computational complexity of the presented CG prediction algorithms. However, in applications such as UAV communications, for example, 3D CG prediction may be necessary. Therefore, the active CG prediction algorithms that will be presented in later sections can naturally be extended to 3D.

The time-averaged narrow-band CG in logarithmic scale for a particular transmitter k can be modeled as a function of space $\psi_k(\cdot)$. The CG function $\psi_k(\cdot)$ evaluated across locations $\mathcal{Q}_P$ for user k are stacked into a vector $\mathbf{x}_k \in \mathbb{R}^{|\mathcal{Q}_P|}$. We further define a utility binary vector variable $\mathbf{z} \in \mathbb{Z}_2^{|\mathcal{Q}_P|}$, where $\mathbb{Z}_2^R$ denotes the set of all binary integer vectors of size R. $[\mathbf{z}]_j = 0$ if the location $\tilde{\mathbf{q}}_j \in \mathcal{Q}_P$ is obstructed by a building, i.e. it is indoor, and $[\mathbf{z}]_j = 1$ otherwise.

2.3.2 UAV swarm and channel gain measurements

We assume that N UAVs are deployed to predict the CG in the AoI. We also assume that these UAVs are constrained to move and collect CG measurements at a constant altitude h_{UAV}. This assumption is made for the purposes of this chapter, to reduce the training time and computational complexity of the presented path planning algorithms. However, in practice, the UAVs would have the ability to move vertically depending on the local flight regulations. Therefore, the path planning algorithms that we will discuss in later sections can be extended to 3D mobility.

Furthermore, let us denote the placement of UAVs at time t, rounded to the nearest grid point in $\mathcal{Q}$, by $\mathbf{P}_t \in \mathcal{Q}_{\text{UAV}}^N$. Similarly, we denote the location of each individual UAV, n, by $\mathbf{p}_{t,n} \in \mathcal{Q}_{\text{UAV}} \subset \mathcal{Q}$. We ignore UAV localization errors in our system model. Moreover, we assume that the control of UAVs can be centralized and performed at one of the UAVs or at a nearby edge server. This also implies that UAVs can communicate within the UAV swarm, either directly or through message relaying within the swarm.

We assume that, as UAVs move, they estimate the CG for the transmitter k via the use of pilot signals. We assume that time is discretized into time steps, where in each time step, UAVs move by a certain distance and estimate the CG at new locations. The set of new locations that have been visited by at least one UAV between time t_1 and up to and including time t_2 is denoted by $\mathcal{V}_{t_1:t_2} = \left\{\mathbf{v}_1, \ldots, \mathbf{v}_{|\mathcal{V}_{t_1:t_2}|}\right\}$. At these locations, the set of measurements of CG are obtained via some measurement model $\mathbf{y}_{k,t_1:t_2} = C(\mathcal{V}_{t_1:t_2}, \mathbf{x}_k)$. Here, we denote the set of CG measurements obtained at locations $\mathcal{V}_{t_1:t_2}$ by $\mathbf{y}_{k,t_1:t_2}$. In this chapter, we assume a perfect CG measurement model, where $\mathbf{y}_{k,t_1:t_2}$ measurements are equal to the true channel gains at $\mathcal{V}_{t_1:t_2}$, which we denote by $\mathbf{x}_{k,t_1:t_2}$. The main variables and parameters introduced in this section are summarized in Table 2.1. Some of the parameters and variables in Table 2.1 are introduced in later sections.

2.3.3 Objectives

The main objective of this chapter is to devise active methods for prediction of CG $\mathbf{x}_k$, as illustrated in Fig. 2.1. We seek to develop methods that adapt to the local environment using two kinds of input features: (1) measurements $\mathbf{y}_{k,1:t}$ collected by the UAVs up to time time t; (2) 3D map $\mathbf{M}$ of the AoI. Given that we seek to develop approaches for prediction of $\mathbf{x}_k$ that rely on CG measurements, optimally controlling the UAVs to collect measurements that provide maximum amount of information about $\mathbf{x}_k$ is just as important as optimally utilizing the measurements to predict $\mathbf{x}_k$. Therefore, we seek to develop active CG prediction approaches that consist of methods for control of UAVs to collect measurements and methods that can predict $\mathbf{x}_k$.

2.4 Deep Learning Based Active Channel Gain Prediction

In this section, we explain our DL based approach for active CG prediction. Our proposed approach consists of two key parts: a deep-learning algorithm to provide a posterior probability prediction of $\mathbf{x}_k$, $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$, and a multi-agent deep reinforcement learning algorithm to control the UAVs to collect measurements based on $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$, 3D map $\mathbf{M}$ and UAV locations. We propose an iterative procedure where in each time slot, the UAVs move and measure the CG at new locations. At the end of each time slot, the measurements from all UAVs are combined to update $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$.

The posterior $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$ represents the belief on what the true CG $\mathbf{x}_k$ is and it is used in two ways in our proposed approach. First, the predicted CG can be obtained as $\hat{\mathbf{x}}_k = \int_{\mathbf{x}_k} \mathbf{x}_k p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$. Second, the posterior distribution $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$ is used to estimate the uncertainty $\mathrm{Var}(\hat{\mathbf{x}}_k)$ of predicted CG. While the posterior probability $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$ can be used to estimate the uncertainty of CG prediction across space, optimally collecting measurements to feed into the DL model is still a challenging problem. For example, it may not be optimal to simply move UAVs towards locations with high CG uncertainty. This is the case because it is not possible to determine how a deep neural network uses the input measurements to arrive at the final prediction, which is why deep neural networks are often referred to as black box models. Furthermore, it is not clear how can multiple UAVs coordinate their movement to optimally collect CG measurements. Therefore, we utilize reinforcement learning to learn optimal path planning algorithms for UAVs that will result in optimal set of CG measurements being collected.

We assume that the UAVs are allotted a fixed number of steps T to estimate $\mathbf{x}_k$. However, an alternative approach is possible where UAVs can perform early stopping of measurement collection and CG prediction based on the posterior probability $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$.

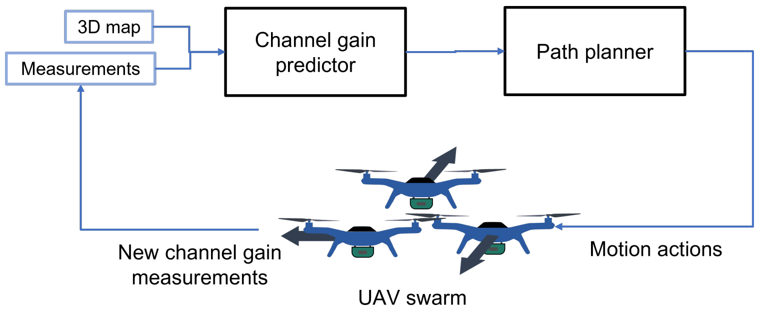


Figure 2.1: Active CG prediction using multiple UAVs

The proposed active DL CG approach is illustrated in Fig. 2.2 and its two main components are explained in sections 2.5 and 2.6. Our proposed DL approach for probabilistic CG prediction is explained in Sec. 2.5 and our proposed reinforcement learning approach for path planning for measurement collection is explained in Sec. 2.6.

2.5 Deep learning based probabilistic channel gain predictor

The purpose of the proposed DL CG predictor is to output a posterior distribution $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$ given measurements $\mathbf{y}_{k,1:t}$ collected by the UAVs and a 3D map $\mathbf{M}$ of the AoI. Deep neural networks are known to be universal function approximators, which is why we use their capabilities to learn the complicated relationship between the 3D map $\mathbf{M}$, CG measurements $\mathbf{y}_{k,1:t}$ and the CG $\mathbf{x}_k$. To design and train the predictor, we must select a probability distribution that will be used as the posterior $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M})$. We selected the Gaussian distribution as the posterior $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M}) = \mathcal{N}(\mu_k, \mathbf{\Sigma}_k)$ due to its similarity to the Gudmundons model of channel gain shadowing [Gud91] and because it performed the best in terms of prediction accuracy amongst the alternatives that we tried. Two DNN models are used to predict the mean μ_k and covariance $\mathbf{\Sigma}_k$, which uniquely define the Gaussian posterior $p(\mathbf{x}_k \mid \mathbf{y}_{k,1:t}, \mathbf{M}) = \mathcal{N}(\mu_k, \mathbf{\Sigma}_k)$. Furthermore, we restrict the covariance matrix $\mathbf{\Sigma}_k$ to be diagonal to simplify the training loss function, which reduces the computational complexity during training and improves the training convergence rate.

The two deep neural networks that learn the mappings from $(\mathbf{y}_{k,1:t}, \mathbf{M})$ to μ_k and $\boldsymbol{\Sigma}_k$, are denoted as $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$, respectively. The parameters of the two DNNs are denoted by θ.

The loss function is based on maximizing the log-likelihood of having observed the training dataset given a Gaussian distribution:

$$\mathcal{L}(\theta) = \frac{1}{2D}\sum_{i=1}^{D}(\Delta_k^{(i)})^T\Sigma_\theta(\mathbf{y}_{k,1:t}^{(i)}, \mathbf{M}^{(i)})^{-1}\Delta_k^{(i)} + \frac{1}{2D}\sum_{i=1}^{D}\log\left((\mathbf{z}_k^{(i)})^T\mathrm{diag}\left(\Sigma_\theta(\mathbf{y}_{k,1:t}^{(i)}, \mathbf{M}^{(i)})\right)\right) \tag{2.1}$$

where D is the number of training samples. In Eq. 2.1, we used a substitute variable $\Delta_k^{(i)} = \left(\mu_\theta(\mathbf{y}_{k,1:t}^{(i)}, \mathbf{M}) - \mathbf{x}_k^{(i)}\right) \odot \mathbf{z}_k^{(i)}$. The loss function considers only outdoor coordinates through the use of $\mathbf{z}_k$ in the expression. The variable $\mathbf{z}_k$ must be known for training but not for prediction during deployment. We minimize the loss function in Eq. 2.1 to optimize the parameters of the deep neural networks: $\theta = \arg\min_\theta \mathcal{L}(\theta)$.

2.5.1 Deep neural network design

We use a convolutional neural network architecture for deep neural networks $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$. Convolutional neural networks are suitable for task of spatial CG prediction because convolutional neural network layers consist of spatial filters that enforce a local connectivity pattern between neurons of adjacent layers. This architecture ensures that the learned filters produce the strongest response to spatially local CG measurements. The particular convolutional neural network architecture that we used was U-Net as illustrated in Fig. 2.2, which was first used for image segmentation problems [RFB15]. U-Net architecture is particularly suitable for CG prediction due to skip connections, shown in Fig. 2.2, which enable 3D map and CG measurements to be passed to the final layers without information loss due to encoding. To use convolutional neural networks for this problem, we convert the inputs $(\mathbf{y}_{k,1:t}, \mathbf{M})$ into matrices that can be processed by convolutional neural networks, as shown in

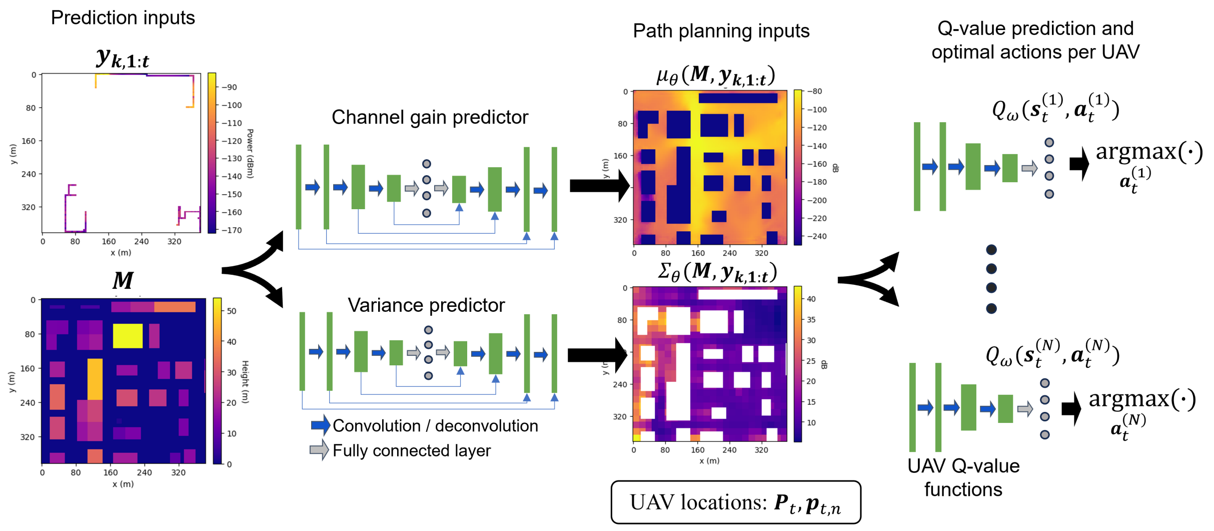

Prediction inputs
$y_{k,1:t}$
M
Channel gain predictor
Variance predictor
Convolution / deconvolution
Fully connected layer
Path planning inputs
$\mu_\theta(M, y_{k,1:t})$
$\Sigma_\theta(M, y_{k,1:t})$
UAV locations: $P_t, p_{t,n}$
Q-value prediction and optimal actions per UAV
$Q_\omega(s_t^{(1)}, a_t^{(1)})$
argmax(·)
$a_t^{(1)}$
$Q_\omega(s_t^{(N)}, a_t^{(N)})$
argmax(·)
$a_t^{(N)}$
UAV Q-value functions

Figure 2.2: The proposed active DL CG prediction approach consists of two main components: DL probabilistic predictor and a RL path planner. The predictor outputs mean and variance of the posterior distribution of the CG, $\mu_\theta(\mathbf{M}, \mathbf{y}_{k,1:t})$ and $\Sigma_\theta(\mathbf{M}, \mathbf{y}_{k,1:t})$, given the channel gain measurements $\mathbf{y}_{k,1:t}$ and a 3D map $\mathbf{M}$. UAV path planning is based on on reinforcement learning. Each UAV uses a DQN to predict the value of an action in a particular state and then select the optimal action accordingly. The DQN inputs are locations of the UAVs, $\mu_\theta(\mathbf{M}, \mathbf{y}_{k,1:t})$ and $\Sigma_\theta(\mathbf{M}, \mathbf{y}_{k,1:t})$.

Fig. 2.2. First, the 3D map $\mathbf{M}$ is converted into a tensor $\tilde{\mathbf{M}} \in \mathbb{R}^{\frac{l}{d} \times \frac{w}{d}}$, where each entry $\left[\tilde{\mathbf{M}}\right]_{i,j}$ is equal to the building or terrain height at coordinates (id, jd). Next, the measurements collected up to time t are converted into a matrix $\tilde{\mathbf{Y}}_{k,1:t} \in \mathbb{R}^{\frac{l}{d} \times \frac{w}{d}}$, where each entry $\left[\tilde{\mathbf{Y}}_{k,1:t}\right]_{i,j}$ is equal to the CG measurement at coordinates (id, jd, h_{UAV}), if $(id, jd, h_{\text{UAV}}) \in \mathcal{V}_{1:t}$, and is equal to c_L, otherwise. c_L is a padding value that is set to a value outside of the reasonable range of CG values. The outputs of $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ are also matrices of form $\mathbb{R}^{\frac{l}{d} \times \frac{w}{d}}$, which are transformed into a vector $\mathbb{R}^{|\mathcal{Q}_P|}$ and a diagonal matrix $\mathbb{R}^{|\mathcal{Q}_P| \times |\mathcal{Q}_P|}$, respectively.

2.6 Deep reinforcement learning for optimal measurement collection

In this section, we describe our RL approach for control of UAVs to optimally collect CG measurements, which is part of our active CG prediction framework, as illustrated in Fig. 2.2.

2.6.1 Path planning problem formulation

We formulate the trajectory design problem as a sequential decision making problem. At each time step t, the path planning controller will define an action $\mathbf{u}_t$. The action $\mathbf{u}_t$ specifies the next displacement for each of the N UAVs. We limit the number of possible displacements to $U = 4$, where each displacement is of length d and along one of the horizontal directions. Therefore, the motion actions are discrete and number of feasible motion actions across the entire UAV swarm is U^N. We define the rules according to which UAVs move using a transition function $\mathbf{P}_{t+1} = T(\mathbf{P}_t, \mathbf{u}_t)$, which defines the positions of UAVs at time $t+1$, $\mathbf{P}_{t+1}$, given the positions of UAVs at time t, $\mathbf{P}_t$, and motion action $\mathbf{u}_t$. The action at time step t is constrained to $\mathbf{u}_t \in G(\mathbf{P}_t)$ due to buildings and other obstacles. The function $G(\mathbf{P}_t)$ can be used to capture other constraints on motion of UAVs such as no-fly zones.

Given the above notation, we can define the trajectory design as an optimization problem:

$$\begin{aligned} \max_{\mathbf{u}_1,\dots,\mathbf{u}_T} \quad & \frac{1}{|\mathcal{Q}_\mathcal{P}|} ||\mu_\theta(\mathbf{y}_{k,1:T+1}, \mathbf{M}) - \mathbf{x}_k||_2^2 \\ \text{s.t.} \quad & \mathbf{P}_{t+1} = T(\mathbf{P}_t, \mathbf{u}_t), \mathbf{u}_t \in G(\mathbf{P}_t) \end{aligned} \tag{P1}$$

where the objective function is the mean square error (MSE) of the CG prediction after T time steps. This is a challenging problem to solve because we cannot predict how the motion actions $\mathbf{u}_1, \dots, \mathbf{u}_T$ will influence the MSE. Therefore, we turn to reinforcement learning to learn the relationship between $\mathbf{u}_1, \ldots, \mathbf{u}_T$ and the MSE $\frac{1}{|\mathcal{Q}_{\mathcal{P}}|}\,||\mu_\theta(\mathbf{y}_{k,1:T+1}, \mathbf{M}) - \mathbf{x}_k||_2^2$, to solve for optimal actions $\mathbf{u}_1, \ldots, \mathbf{u}_T$.

2.6.2 Reinforcement learning background

Reinforcement learning is a branch of machine learning that is concerned with making sequences of decisions. It can be applied to problems that can be casted as a Markov decision process (MDP). A Markov decision process (MDP) is defined by a tuple $\{\mathcal{S}, \mathcal{A}, P, r\}$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, P is the transition probability function from state $\mathbf{s}$ to $\mathbf{s}' \in \mathcal{S}$ after action $\mathbf{a} \in \mathcal{A}$ is performed, r is the reward obtained after $\mathbf{a}$ is executed in state $\mathbf{s}$. An action space $\mathcal{A}$ can be a discrete or a continuous set.

A policy $\pi : \mathcal{S} \to \mathcal{A}$ is a function that maps a state $\mathbf{s} \in \mathcal{S}$ into an action $\mathbf{a} \in \mathcal{A}$. With some abuse of notation, we also use variable t to denote the time-step in the MDP. Thus, at time t, the agent observes the state $\mathbf{s}_t$, then based on a specific policy π, it takes action $\mathbf{a}_t = \pi(\mathbf{s}_t)$. Consequently, a new state $\mathbf{s}_{t+1}$ will be reached with probability $P(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ and a reward r_t will be received. The observed information from the environment, the reward r_t and $\mathbf{s}_{t+1}$ are used to improve the policy. This process is repeated until the optimal policy is reached. We use $\rho_\pi(\mathbf{s}_t)$ and $\rho_\pi(\mathbf{s}_t, \mathbf{a}_t)$ to denote the state and state-action probability distributions induced by a policy π.

The objective function in reinforcement learning is normally the expected sum of rewards $\sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi}[r(\mathbf{s}_t, \mathbf{a}_t)]$, where γ is the discount factor. Reinforcement learning methods can broadly be classified into two categories: policy learning and Q-learning. In policy learning methods, the goal is to directly learn the optimal policy function π. In Q-learning methods, the goal is to learn the Q-value function $Q(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right]$, which defines the expected reward in a state $\mathbf{s}$, after taking the action $\mathbf{a}$. Based on the Q-value function, the optimal policy is then $\pi(\mathbf{s}) = \operatorname{argmax}_a Q(\mathbf{s}, \mathbf{a})$. Deep Q-learning is an extension to the Q-learning paradigm whereby a deep neural network is used to approximate

$Q(\mathbf{s}, \mathbf{a})$.

2.6.3 Path planning as a Markov decision process

Next, we convert the path planning problem described in Sec. 2.6.1 into an MDP. Based on the problem formulation (P1), if we were to define the MDP such that the entire UAV swarm is considered a single agent, the size of the action space would grow exponentially with the number of UAVs. While such MDP formulation would be suitable for solving the path planning problem (P1), for a large number of UAVs, the size of the actions space would prevent effective training of reinforcement learning policies. Therefore, we focus on decentralized control where each UAV will act as an independent agent, while treating the rest of the UAVs as part of the environment. Even though their control is decentralized in our approach, the UAVs within the swarm are still cooperative and share the same common goal stated in the objective function in (P1). Therefore, our problem can be formulated as a multi-agent reinforcement learning problem and the respective MDP formulation is explained next.

2.6.3.1 State space

Since the state of the environment is not fully observable, a set of observations replaces the role of the state in our MDP formulation.

In multi-agent reinforcement learning, the input observations often consist of observation related to the environment and messages emitted by other agents that assist the agents in collaboratively achieving the common goal. One of the main challenges related to multi-agent reinforcement learning is designing or learning communication protocols between the agents [Foe18]. In our approach, we do not aim to learn the messages to be passed between the agents but instead utilize the deep-learning CG predictor to enable cooperation between the UAVs. The DL predictor processes the information collected by the UAVs, namely the CG measurements collected by the UAVs and their locations, and outputs features $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$. The features $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ are part of the observation $\mathbf{s}_t^{(n)}$ observed by UAV n at time t. Additionally, each UAV n observes its own current location, location of other UAVs in the swarm, and a 3D map of the environment $\mathbf{M}$. In summary, the observation for UAV n at time t is $\mathbf{s}_t^{(n)} = (\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M}), \Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M}), \mathbf{M}, \mathbf{P}_t, \mathbf{p}_{t,n})$.

2.6.3.2 Action space

Since the control of the UAVs is distributed, the action $\mathbf{a}_t^{(n)}$ dictates the motion of the UAV n at time t. Each action maps to one of the U the possible displacements defined in Sec. 2.6.1.

2.6.3.3 Reward

The reward function is designed to maximize the objective function in (P1) and the agents receive the reward $r_t^{(n)}$ at time t. The agent receives a reward $r^{(n)}(t) = r_e e^{(n)}(t)$ when $1 \leq t < T$. $e^{(n)}(t)$ is equal to 1 if the UAV n visits a new location at time t and is 0 otherwise. The exploration reward is useful during training to ensure that UAVs don't visit the same location multiple times. At $t = T$, the reward is based on the prediction error, $r_t^{(n)} = -r_r \left|\left|\mu_\theta(\mathbf{y}_{k,1:T+1}, \mathbf{M}) - \mathbf{x}_k\right|\right|_2$. The constants r_e and r_r were empirically selected during training.

2.6.4 Deep Q-learning algorithm

The deep Q-learning algorithm that we developed to solve the MDP described in the previous section is based on the DQN algorithm [MKS13]. In DQN, the Q-value function is approximated by a neural network Q_ω, with parameters ω. An estimate of the true Q-value at time t, Q_t, can be obtained by using a single sample estimate of the Bellman backup operator

$$\widehat{\mathcal{T}Q}_t = r_t + \max_{\mathbf{a}_{t+1}} \gamma Q_\omega(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \tag{2.3}$$

This is called a single sample estimate because only the reward r_t at the current time instant t is used to approximate the infinite horizon Q-value function.

In order to train Q_ω to approximate Q, the following minimization is done over sample data,

$$\underset{\omega}{\text{minimize}} \sum_t \left|\left|\widehat{\mathcal{T}Q}_t - Q_\omega(\mathbf{s}_t, \mathbf{a}_t)\right|\right|^2 \tag{2.4}$$

In the DQN algorithm, the training and the interaction of the agent with the environment happen in parallel. As the agent gathers experience, samples of that experience are stored and the minimization in the Eq. 2.4 is done periodically, every τ_L steps, by randomly sampling a batch of B_L recorded samples and applying gradient descent. This is referred to as experience replay. Each sample is a tuple $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ and these are stored in the replay buffer.

The agent interacts with the environment following the ϵ-greedy policy, where at any time t the agent either takes a random action at probability ϵ or the Q-value optimal action $\text{argmax}_{\mathbf{a}_t} Q_\omega(\mathbf{s}_t, \mathbf{a}_t)$ at probability $(1 - \epsilon)$. In the implementation of DQN, there is an additional Q_ω, called the target Q-network. The target Q-network is used in the Bellman backup operator but it is not directly optimized over. Instead, its parameters are copied from the main Q-network at period τ_{DQN}. The target Q-network is included to improve the stability during training.

Since its original inception, several modifications of the original DQN algorithm have been shown to improve performance over a variety of tasks. In this chapter, we apply two of such modifications we found to be useful on our problem: multi-step learning and distributional RL [BDM17]. In multi-step learning, Bellman backup operator in Eq. 2.3 is extended to include reward samples from M consecutive steps:

$$\widehat{\mathcal{T}Q_t} = \sum_{m=0}^{M-1} \gamma^m r_{t+m} + \max_{\mathbf{a}_{t+M}} \gamma^M Q_\omega(\mathbf{s}_{t+M}, \mathbf{a}_{t+M}) \tag{2.5}$$

In distributional RL, the DQN is trained to predict a discrete distribution of Q-values on a discrete support $\mathbf{v}$, where $\mathbf{v}$ is vector with N_a atoms. To accomplish this, the DQN is modified to have $N_a \times U$ outputs. Furthermore, the loss functions in Eq. 2.4 is replaced by a loss function that ensures that the predicted distribution closely matches the actual distribution of returns, the details of which are omitted for brevity.

2.6.5 Multi-agent deep Q-learning for measurement collection using multiple UAVs

We extended the DQN algorithm to multiple agents in the following way. First, all of the agents or UAVs share the same DQN network Q_ω with identical parameters ω, but each agent acts differently due to different input observations. The policy is trained in a centralized way, where the training samples from all agents are collected and fed to a centralized replay buffer, which is used to train the common policy Q_ω. Similarly to the single-agent DQN, each agent retains a copy of the main DQN Q_ω to collect training samples and these copies are periodically synced with main DQN Q_ω. Overall, the changes we had made to the single-agent DQN algorithm are minimal because we use the deep-learning CG predictor to process information collected by the UAVs and extract features which are relevant for maximizing the reward obtained. These shared input features facilitate collaboration. Furthermore, since all of the agent Q-networks share the same parameters there is an indirect knowledge transfer in the parameter space between the agents, which also enables collaboration.

2.6.6 Deep Q-network design

We used a combination of convolutional and fully-connected layers for the design of our DQN. Convolutional neural networks are suitable for this task because the outputs $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ provided by deep learning predictors are matrices as shown in Fig. 2.2. Similarly, the location information $\mathbf{P}_t$ and $\mathbf{p}_{t,n}$ can be converted into binary matrices, with non-zero entries corresponding to locations of the UAVs. Using convolutional layers, we can efficiently extract lower dimensional features from the inputs, which are utilized by fully-connected layers for Q-value prediction. The last layer has $N_a \times U$ outputs, which correspond to distributional prediction of Q-value.

2.7 Active channel prediction based on Kriging interpolation

Kriging interpolation has been extensively used for CG prediction. However, no methods for active prediction approaches based on Kriging interpolation using multiple UAVs have been proposed in prior literature. While our DL active prediction approach can accomplish CG prediction without transmitter location, traditional interpolation methods such as Kriging remain useful when transmitter location is available. Moreover, the Kriging method has the advantage of not requiring extensive training compared to DL approaches and is therefore easier to deploy. Hence, in this section, we develop an active CG prediction algorithm using multiple UAVs based on Kriging spatial interpolation.

2.7.1 Kriging interpolation for channel gain prediction

Kriging interpolation is an equivalent method to Gaussian process regression (GPR), which is a widely used method for interpolation, classification, supervised learning, and active learning [Ras03]. GPR constructs a probabilistic prediction of a partially observed function (of time and/or space) assuming this function is a realization of a Gaussian process (GP).

In statistical models of the CG, time-averaged CG $\psi_k(\mathbf{q}_j)$ at location $\mathbf{q}_j$ is split into two components, $\psi_k(\mathbf{q}_j) = \psi_{k,PL}(\mathbf{q}_j) + \psi_{k,SH}(\mathbf{q}_j)$, where $\psi_{k,PL}$ is the path loss due to free space attenuation and $\psi_{k,SH}(\mathbf{q}_j)$ is the loss due to shadowing. $\psi_{k,PL}(\mathbf{q}_j)$ can be predicted knowing the antenna radiation pattern and separation of the receiver and the transmitter. On the other hand, $\psi_{k,SH}(\mathbf{q}_j)$ is often modeled as a Gaussian random variable with exponentially decaying spatial correlation according to the Gudmundson model [Gud91]. Accordingly, Kriging interpolation or GPR can be used to predict $\psi_{k,SH}(\mathbf{q}_j)$, while the $\psi_{k,PL}(\mathbf{q}_j)$ component can be obtained knowing the distance of location $\mathbf{q}_j$ to the transmitter.

Using Kriging interpolation, we aim to predict the shadowing gain $\tilde{\mathbf{x}}_{k,1:t}$ at unvisited locations $\mathcal{Q}_P \backslash \mathcal{V}_{1:t}$. Then, we can obtain the CG at locations $\mathcal{Q}_P \backslash \mathcal{V}_{1:t}$ by adding $\tilde{\mathbf{x}}_{k,1:t}$ to the estimated free-space path loss gain. In simple Kriging, the data is modeled as a GP with a zero mean and a prescribed form of the stationary covariance function (also known as kernel). This modelling is compatible with the Gudmundson shadowing model, therefore we will utilize simple Kriging as the foundation of our active prediction approach. The path-loss $\psi_{k,PL}(\mathbf{q}_j)$ is estimated using the model:

$$\psi_{k,PL}(\mathbf{q}_j) = \alpha - \beta \log \left(||\mathbf{q}_j - \mathbf{w}_{TX}||_2 \right) \tag{2.6}$$

where the constants α and β are estimated from CG data by minimizing the mean square error loss.

The kernel defines the shadowing gain cross-covariance between two locations $\mathbf{q}_i$ and $\mathbf{q}_j$: $k(\mathbf{q}_i, \mathbf{q}_j) = \operatorname{Cov}\left(\psi_{k,SH}(\mathbf{q}_i), \psi_{k,SH}(\mathbf{q}_j)\right)$. The kernel we used is based on the Gudmundson model:

$$k(\mathbf{q}_i, \mathbf{q}_j) = \phi \exp\left(\frac{-\,||\mathbf{q}_i - \mathbf{q}_j||_2}{\delta} \right), \tag{2.7}$$

where ϕ and δ are positive constants that are estimated from CG data via negative log-likelihood minimization. The kernel $k(\mathbf{q}_i, \mathbf{q}_j)$ is isotropic, i.e. cross-covariance only depends on distance between $\mathbf{q}_i$ and $\mathbf{q}_j$, but not on the specific values of $\mathbf{q}_i$ and $\mathbf{q}_j$. Given that

UAVs have visited a set of locations $\mathcal{V}_{1:t}$ up to time t, a vector $\tilde{\mathbf{y}}_{k,1:t}$ of shadowing gain measurements will be obtained. The shadowing gain measurements $\tilde{\mathbf{y}}_{k,1:t}$ are obtained by subtracting the estimated free-space path-loss gain obtained using Eq. 2.6 from $\mathbf{y}_{k,1:t}$.

The covariance matrix of the observed shadowing gains $\tilde{\mathbf{y}}_{k,1:t}$ is denoted by $\mathbf{\Sigma}_{v,v} = \mathrm{Cov}\left(\tilde{\mathbf{y}}_{k,1:t}, \tilde{\mathbf{y}}_{k,1:t}\right)$. The matrix $\tilde{\mathbf{y}}_{k,1:t}$ can be obtained using the kernel as follows:

$$\mathbf{\Sigma}_{v,v} = \begin{bmatrix} k(\mathbf{v}_1, \mathbf{v}_1) & \cdots & k(\mathbf{v}_1, \mathbf{v}_{|\mathcal{V}_{1:t}|}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{v}_{|\mathcal{V}_{1:t}|}, \mathbf{v}_1) & \cdots & k(\mathbf{v}_{|\mathcal{V}_{1:t}|}, \mathbf{v}_{|\mathcal{V}_{1:t}|}) \end{bmatrix}. \tag{2.8}$$

Furthermore, we introduce a matrix $\mathbf{\Sigma}_{v,p}$, which denotes the cross-covariance of the shadowing gains between measured locations and prediction locations: $\mathbf{\Sigma}_{v,p} = \mathrm{Cov}\left(\tilde{\mathbf{y}}_{k,1:t}, \tilde{\mathbf{x}}_{k,1:t}\right)$. The cross-covariance matrix $\mathbf{\Sigma}_{v,p}$ can be obtained using the shadowing kernel in Eq. 2.7. Finally, we define the covariance matrix $\mathbf{\Sigma}_{p,p}$ of shadowing gain at predicted locations: $\mathbf{\Sigma}_{p,p} = \mathrm{Cov}\left(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_k\right)$.

Using the matrices $\mathbf{\Sigma}_{v,v}$ and $\mathbf{\Sigma}_{v,p}$, and assuming that the shadowing gain is a zero-mean GP, we can predict $\tilde{\mathbf{x}}_{k,1:t}$ given $\tilde{\mathbf{y}}_{k,1:t}$ as follows:

$$\tilde{\mu}_{k,1:t} = \mathbf{\Sigma}_{v,p}^{T} \mathbf{\Sigma}_{v,v}^{-1} \tilde{\mathbf{y}}_{k,1:t} \tag{2.9}$$

Similarly, we can calculate the conditional covariance of the predicted shadowing gains given $\tilde{\mathbf{y}}_{k,1:t}$ as:

$$\tilde{\mathbf{\Sigma}}_{k,1:t} = \mathbf{\Sigma}_{p,p} - \mathbf{\Sigma}_{v,p}^{T} \mathbf{\Sigma}_{v,v}^{-1} \mathbf{\Sigma}_{v,p} \tag{2.10}$$

The predicted covariance $\tilde{\mathbf{\Sigma}}_{k,1:t}$ depends on $\mathbf{\Sigma}_{v,v}$, $\mathbf{\Sigma}_{v,p}$ and $\mathbf{\Sigma}_{p,p}$, which are only dependent on the visited locations $\mathcal{V}_{1:t}$ and unvisited locations $\mathcal{Q}_P \backslash \mathcal{V}_{1:t}$ (see for example the definition of $\mathbf{\Sigma}_{v,v}$ in Eq. 2.8). Therefore, $\tilde{\mathbf{\Sigma}}_{k,1:t}$ only depends on the selection of the explored locations and not the observed measurements $\tilde{\mathbf{y}}_{k,1:t}$.

2.7.2 Optimal path planning for measurement collection

Next, we develop optimal path planning methods for CG measurement collection. The goal of optimal path planning remains to minimize the MSE $\frac{1}{|\mathcal{Q}_{\mathcal{P}}|}||\tilde{\mu}_{k,1:T+1} - \tilde{\mathbf{x}}_{k,1:T+1}||_2^2$. This problem could also be solved using RL. However, since Kriging interpolation unlike DL prediction is not a data-driven method, we aim to develop optimal path planning methods which also do not rely on large data for this approach. The MSE is not useful for UAV path design since it is unknown to the UAVs. Instead, other criteria that are found to strongly correlate to minimizing the mean square error are utilized for sensing of GPs, such as the entropy of $\tilde{\mathbf{y}}_{k,1:T+1}$ [KSG08]. Since $\tilde{\mathbf{y}}_{k,1:T+1}$ has a Gaussian distribution, this entropy can be calculated as:

$$H(\tilde{\mathbf{y}}_{k,1:T+1}) = \frac{|\mathcal{V}_{1:T+1}|}{2}\log(2\pi e) + \frac{1}{2}\log|\mathbf{\Sigma}_{v,v}| \tag{2.11}$$

The purpose of the metric in Eq. 2.11 can be explained as follows. Using the chain rule for entropy, $H(\tilde{\mathbf{x}}_{k,1:T+1}, \tilde{\mathbf{y}}_{k,1:T+1}) = H(\tilde{\mathbf{x}}_{k,1:T+1} \mid \tilde{\mathbf{y}}_{k,1:T+1}) + H(\tilde{\mathbf{y}}_{k,1:T+1})$. Since $H(\tilde{\mathbf{x}}_{k,1:T+1}, \tilde{\mathbf{y}}_{k,1:T+1})$ is a constant as a function of measured locations, by maximizing $H(\tilde{\mathbf{y}}_{k,1:T+1})$, the conditional entropy $H(\tilde{\mathbf{x}}_{k,1:T+1} \mid \tilde{\mathbf{y}}_{k,1:T+1})$ is minimized and so is $|\tilde{\mathbf{\Sigma}}_{k,1:t}|$.

Based on the entropy metric in Eq. 2.11, we can formulate the path planning problem as:

$$\max_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \quad H(\tilde{\mathbf{y}}_{k,1:T+1}) \text{ s.t. } \mathbf{P}_{t+1} = T(\mathbf{P}_t, \mathbf{u}_t), \mathbf{u}_t \in G(\mathbf{P}_t) \tag{P2}$$

The problem (P2) is known to be NP-hard and can only be optimally solved using an exhaustive algorithm. However, the number of possible paths exponentially increases with T and N, so an exhaustive approach becomes intractable for real-time applications. Therefore, it is necessary to develop a suboptimal tracktable heuristic.

We utilize the derivation in [LDK11] to recast the problem (P2) into a deterministic MDP. Let us denote the measurements collected by the UAVs at time i as $\tilde{\mathbf{y}}_{k,i}$. Then, using the chain rule for entropy, we can rewrite the entropy of the shadowing gains at measured locations as: $H(\tilde{\mathbf{y}}_{k,1:T+1}) = H(\tilde{\mathbf{y}}_{k,1}) + \sum_{t=1}^{T} H(\tilde{\mathbf{y}}_{k,t+1} \mid \tilde{\mathbf{y}}_{k,1:t})$. By approximating $H(\tilde{\mathbf{y}}_{k,t+1} \mid \tilde{\mathbf{y}}_{k,1:t})$ by an upper bound $H(\tilde{\mathbf{y}}_{k,t+1} \mid \tilde{\mathbf{y}}_{k,t})$, we can simplify our objective function to be

$$H(\tilde{\mathbf{y}}_{k,1:T+1}) \approx H(\tilde{\mathbf{y}}_{k,1}) + \sum_{t=1}^{T} H(\tilde{\mathbf{y}}_{k,t+1} \mid \tilde{\mathbf{y}}_{k,t}) \tag{2.13}$$

Then, the new optimization problem based on the approximation in Eq. 2.13 is:

$$\begin{aligned} \max_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \quad & \sum_{t=1}^{T} H(\tilde{\mathbf{y}}_{k,t+1} \mid \tilde{\mathbf{y}}_{k,t}) \\ \text{s.t.} \quad & \mathbf{P}_{t+1} = T(\mathbf{P}_t, \mathbf{u}_t), \mathbf{u}_t \in G(\mathbf{P}_t) \end{aligned} \tag{P3}$$

where we have omitted the term $H(\tilde{\mathbf{y}}_{k,1})$ from the objective function since it does not depend on $\mathbf{u}_1, \ldots, \mathbf{u}_T$. This objective function leads to paths with actions such that entropy of locations explored at time $t+1$ given the shadowing gain measurements at time t is maximized.

The problem (P3) can be converted into a deterministic MDP, where the state at time t is simply $\mathbf{s}_t = \mathbf{P}_t$ and the action is $\mathbf{a}_t = \mathbf{u}_t$. The reward function at time t is defined as:

$$r(\mathbf{P}_t, \mathbf{u}_t) = \begin{cases} H(\tilde{\mathbf{y}}_{k,t+1} \mid \tilde{\mathbf{y}}_{k,t}) & \mathbf{u}_t \in G(\mathbf{P}_t) \\ -\infty & \text{o.w.} \end{cases} \tag{2.15}$$

where the negative infinity reward is assigned if an illegal action is taken at time t. To solve this MDP, we can apply the value iteration algorithm. Let $V_\pi(\mathbf{P}_t)$ be the value function that defines the sum future reward when acting according to a certain policy π starting from some state $\mathbf{P}_t$. Let $V^*(\mathbf{P}_t)$ be the value function obtained using an optimal policy π^* that yields the maximum $V_\pi(\mathbf{P}_t)$:

$$V^*(\mathbf{P}_t) = \max_{\mathbf{u}_t,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} r(\mathbf{P}_t, \mathbf{u}_t) \tag{2.16}$$

We can express the optimal value using a recurrent relation as:

$$V^*(\mathbf{P}_t) = \max_{\mathbf{u}_t} \left(r(\mathbf{P}_t, \mathbf{u}_t) + V^*(T(\mathbf{P}_t, \mathbf{u}_t)) \right) \tag{2.17}$$

Since $V^*(\mathbf{P}_t)$ can be expressed using a recurrent relation, we can then use forward value iteration to solve for $V^*(\mathbf{P}_1)$ and optimal $\mathbf{u}_1, \ldots, \mathbf{u}_T$ [LaV06, p. 48].

2.7.3 Computational complexity of path planning

The main source of computational load in the proposed active sensing approach is the forward value iteration, which is used to solve for $V^*(\mathbf{P}_1)$. Forward value iteration will have a complexity $O\left(TU^N \left(\frac{lw}{d^2}\right)^N\right)$, where U^N is the size of the action space and $\left(\frac{lw}{d^2}\right)^N$ is the size of the state space.

2.8 Results

2.8.1 Simulation environment

In this section, we describe the details of wireless channel simulation and the set of environments generated to train and test the proposed algorithms.

The main tool used for wireless channel simulation was ray tracing. Ray tracing is a channel propagation modeling tool that provides estimates of channel gain, angle of arrival/departure, and time delays by numerically solving Maxwell's equations in far-field propagation conditions [YI15]. A ray-tracing software takes in the 3D map of the environment, along with other parameters, such as transmission frequency, transmitter location,

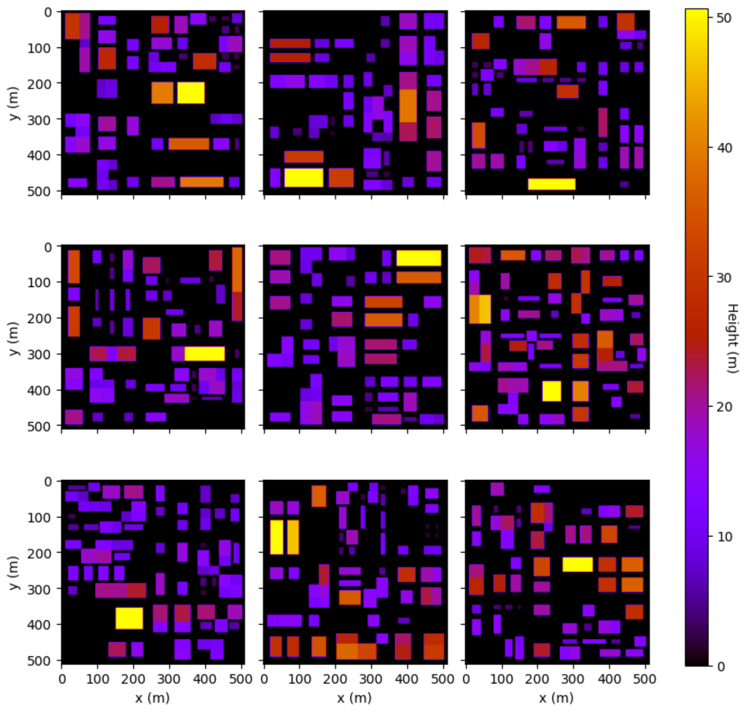


Figure 2.3: Heat maps of randomly generated urban environments. The colors corresponds to building heights.

and material properties of environment objects to trace the radio propagation paths and calculate the channel state at the desired points. The particular ray tracing software we used was Wireless Insite. To limit the ray-tracing computation time, we constrain the maximum number of reflections per a propagation ray to 3 and the maximum number of diffractions per ray to 1.

In order to create an expansive set of environments, we used a handcrafted script to randomly generate Manhattan-grid-like urban environments. We simulate a square shaped area of dimensions 486m $\times$ 486m. The generation procedure starts by dividing the area into city blocks with random widths and length. The number of blocks per each dimension is 5, with a total of 25 blocks in the environment. Then, open spaces and rectangular-base buildings with random dimensions are added within those blocks. Some examples of randomly generated environments are shown in Fig. 2.3. In total, we generated 300 urban environments. In each environment, we placed transmitters uniformly spaced at 97.2m apart, which equates to 25 transmitter positions per environment. However, if a randomly generated transmitter location was indoor, it was removed from simulated environments. For each transmitter location, Wireless InSite was used to calculate the channel gain values over a 3D grid of points spaced at 4 m apart and at altitudes ranging from 10 m to 30 m, over the entire width and length of the environment. The calculations were ran for a carrier frequency of 5 GHz. The dataset will be provided upon request to the authors. The outputs from Wireless InSite were then processed in Python and used for simulations. During training and testing, we crop the size of the simulated environment to a space with a footprint of size 384m $\times$ 384m with a random center within the original 486m $\times$ 486m area. This data augmentation was performed to add more diversity into the original data set and to add randomness to the transmitter locations.

Table 2.2: Architecture of the deep neural networks used for channel gain prediction.

CG prediction U-Net					
Layer	**In**	**1**	**2**	**3**	**4**
Out. size	96×96	96×96	48×48	48×48	24×24
Channels	in	16	16	32	32
Type	Conv.	Conv.	Conv.	Conv.	Conv.
Layer	**5**	**6**	**7**	**8**	**9**
Out. size	24×24	12×12	12×12	6×6	4608
Channels	64	64	128	128	
Type	Conv.	Conv.	Conv.	Conv.	Dense
Layer	**10**	**11**	**12**	**13**	**14**
Out. size	12×12	24×24	48×48	96×96	96×96
Channels	128	64	32	16	1
Skip connect.	8	7	5	3	1
Type	Deconv.	Deconv.	Deconv.	Deconv.	Conv.

Table 2.3: Architecture of the deep neural networks used as DQNs.

DQN				
Layer	**In**	**1**	**2**	**3**
Out. size	96×96	48×48	24×24	16×16
Channel	in	64	128	256
Filter size		4	4	2
Type	Conv.	Conv.	Conv.	Conv.
Layer	**4**	**5**	**6**	**7**
Out. size	512	256	40	160
Type	Dense	Dense	Dense	Dense

Table 2.4: Training parameters for CG predictor and DQN

CG predictor training parameters		**DQN training parameters**	
Description	**Parameter**	**Description**	**Parameter**
Learning rate	10^{-3}	Learning rate	10^{-5}
Adam param.	$\beta_1 = 0.9$	Exploration	$\varepsilon = 0.03$
Adam param.	$\beta_2 = 0.999$	Replay buffer	6×10^6
Adam param.	$\hat{\epsilon} = 10^{-6}$	Batch size	256
Random walk param.	$p = 0.8$	Target update	$\tau_{DQN} = 1000$
Batch size	160	Learning steps	$M = 5$

2.8.2 Training of DL channel gain predictor

Next, we describe the training details of the channel gain predictor proposed in Sec. 2.5. The data generated in 75 out of 300 city environments was used to train the predictor, while the data from 25 environments was used to validate the dataset. We refer to the former portion of the dataset as T1 and to the latter as T2. We used the Adam optimizer to minimize the loss function in Eq. 2.1 [KB14]. We found that training performance was highly dependent on the selection of the Adam parameters, which are shown in Table 2.4, along with other relevant training parameters. We used the same notation for Adam parameters as in the original chapter [KB14]. In order to train the predictor, we randomly generated measurement inputs $\mathbf{y}_{k,1:t}$. During training, we assume measurements are obtained using a random waypoint motion model. We use a random trajectory to emulate measurement collection by UAVs on some planned paths. A random waypoint trajectory for a UAV is obtained as follows. At time t, each UAV takes independent random motion actions at probability $1-p$, and at probability p, the previous motion action is repeated by the UAV. The path length per UAV is random and uniformly distributed between 50 and 300 steps. Furthermore, we train separate DL models depending on the number of UAVs N collecting the measurements.

The architecture of the U-Nets used for $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ is shown in Table 2.2. The architecture consists of a series of convolutional layers or convolutional plus max-pooling layers to encode the inputs. The size of the output of each layer is shown in the table. Each layer outputs a number of channels which is equal to the number of convolutional filters in the layer. The convolutional filter size was 4×4, with stride size 1. A dense layer follows after the encoding layers. After, there is a sequence of layers that perform upsampling and convolution, which we refer to as deconvolution layers. The inputs of each deconvolution layer are concatenated with outputs of one of the encoding layers using skip connections. The skip connections are denoted in the table. The final layer is a convolutional layer which also uses skip connections. We used the ReLU activation function for $\mu_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ and tanh activation for $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$. We found that using tanh activation for $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ leads to better performance than when using ReLU.

2.8.3 Training of DQN policies

In this subsection, we describe the training details of the DQN policies for UAV control proposed in Sec. 2.6. The data generated in 170 out of 300 city environments was used to train the algorithm while the data from 30 out of 300 environments was used to test the RL policy. We refer to the former portion of the dataset as T3 and to the latter as T4.

We use the ϵ-greedy policy for exploration, however the agent's random actions are steered. Namely, the agent never takes a random action that would lead to it leaving the map or colliding with a building. The value of ϵ is shown in Table 2.4. We also ensure that the agent never leaves the map or collides with a building when taking actions according to the DQN or when moving randomly.

The neural network architecture for DQNs is shown in Table 2.3. The DQNs consist of a series of convolutional layers with strides of size 4 or 2. The output size, the number of channels and filter size are shown in the Table 2.3. The final layers of DQN are fully connected. The activation function used was ReLU.

2.8.4 Benchmarks

Next, we explain the benchmark algorithms that we will compare our proposed approaches to.

2.8.4.1 Greedy active DL prediction

The first benchmark is based on the CG predictor that we introduced in Sec. 2.5. The paths are designed to move the UAVs through the locations of maximum variance as predicted by $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$. The intuition behind this approach is to collect new measurements in the locations where the predicted error is the largest. This also limits this approach to scenarios where $h_P = h_{\text{UAV}}$. Since the variance prediction $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ is continuously updated by measurements obtained by the UAVs, the planned paths also need to be updated periodically over the course of time $1 \leq t < T$. In the first $T_{start} = 20$ steps, the UAVs move randomly since $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$ is unreliable for the purposes of path planning. Afterwards, UAV trajectories are updated every $T_{plan} = 40$ steps. Let us denote the predicted covariance matrix for the set of locations $\mathcal{X}$ by $[\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})]_{\mathcal{X}}$. Then, the paths for the UAVs at time t_1 can calculated by solving the optimization problem:

$$\begin{aligned} \max_{\mathbf{u}_{t_1},\ldots,\mathbf{u}_T} \quad & \operatorname{Tr}\left([\Sigma_\theta(\mathbf{y}_{k,1:T+1}, \mathbf{M})]_{\mathcal{V}_{t_1:T+1}} \right) \\ \text{s.t.} \quad & \mathbf{P}_{t+1} = T(\mathbf{P}_t, \mathbf{u}_t), \mathbf{u}_t \in G(\mathbf{P}_t) \end{aligned} \tag{P4}$$

As with problem (P3), we convert this problem into an MDP. In order to maximize the objective function in (P4), it is necessary to keep track of the locations visited by the UAVs to avoid repeated visits. This can be achieved by defining the state $\mathbf{s}_t$ to include all locations visited up to time t. However, in this case, the size of the state space would be too large for efficient computation of optimal paths. Instead, we ensure that UAVs do not perform repeated visits by appropriately designing the reward function. The reward function at time t is defined as:

$$r(\mathbf{P}_t, \mathbf{u}_t) = \begin{cases} \operatorname{Tr}\left([\Sigma_\theta(\mathbf{y}_{k,1:t+1}, \mathbf{M})]_{\mathcal{V}_{t:t+1}} \right) & \mathbf{u}_t \in G^*(\mathbf{P}_t, \mathbf{P}_{t_1}) \\ -\infty & \text{o.w.} \end{cases} \tag{2.19}$$

where the function $G^*(\mathbf{P}_t, \mathbf{P}_{t_1})$ ensures that no illegal actions are taken and also that the UAVs are moving away from their respective starting locations. The latter is necessary to ensure that UAVs are not visiting the same location multiple times. The state at time t is defined as $\mathbf{s}_t = \mathbf{P}_t$ and the action is $\mathbf{a}_t = \mathbf{u}_t$. Given this MDP definition, we can calculate the UAV paths using value iteration (Eq. 2.17). Furthermore, due to the nature of the reward function, forward value iteration can be applied independently per UAV.

We will use this benchmark to compare against our proposed active DL CG prediction approach, since it is also transmitter location free and uses 3D maps. The disadvantage of this benchmark compared to the proposed active DL approach is that the greedy objective function in (P4) can lead to multiple UAVs exploring locations in close proximity of one another if these locations have high variance as predicted by $\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})$. Furthermore, as discussed in Sec. 2.6, since DNNs are black-box models, collecting the measurements in regions of highest predicted variance may not minimize the final prediction error.

The main source of computational load in this benchmark is the forward value iteration, which has a computational complexity $O\left(NTU\left(\frac{lw}{d^2}\right)\right)$.

2.8.4.2 Greedy active Kriging prediction

The second benchmark is based on the Kriging predictor explained in Sec. 2.7.1. Similar to our previous benchmark, the paths are designed to move the UAVs through the locations of maximum variance as predicted by $\tilde{\mathbf{\Sigma}}_{k,1:t}$ in Eq. 2.10.

$$\begin{aligned} \max_{\mathbf{u}_{t_1},\ldots,\mathbf{u}_T} \quad & \mathrm{Tr}\left(\left[\tilde{\mathbf{\Sigma}}_{k,1:T+1}\right]_{\mathcal{V}_{t_1:T+1}}\right) \\ \text{s.t.} \quad & \mathbf{P}_{t+1} = T(\mathbf{P}_t, \mathbf{u}_t), \mathbf{u}_t \in G(\mathbf{P}_t) \end{aligned} \tag{P5}$$

The path calculation is performed using forward value iteration in the same way as in the previous benchmark. We will use this benchmark to compare against our proposed active Kriging CG prediction approach, since it also requires transmitter location to be known. This benchmark has the disadvantage that it can lead to multiple UAVs exploring similar regions due to the greedy objective function in (P5).

2.8.4.3 Random waypoints with DL prediction approach

In this benchmark approach, UAVs move according to the random waypoints strategy and prediction is done using our proposed DL predictor, as explained in Sec. 2.5. The purpose of this approach is to evaluate the importance of optimal path planning for CG prediction. We will use this benchmark to quantitatively compare against our proposed active DL CG prediction approach. Furthermore, we will use it to evaluate the accuracy of the CG predictor for various scenarios in the absence of optimized path planning.

2.8.4.4 Random waypoints with Kriging prediction approach

In this benchmark approach, UAVs move according to the random waypoints strategy and prediction is done using Kriging prediction, explained in Sec. 2.7.1. We will use this approach as a benchmark to compare against our proposed active Kriging prediction approach. Furthermore, we will use it to evaluate the accuracy of Kriging prediction for various scenarios in the absence of optimized path planning.

2.8.5 Evaluation of the deep learning channel gain predictor

First, we evaluate the performance of the probabilistic DL CG predictor without optimized path planning and use random-waypoints UAV motion with $p = 0.8$ for measurement collection. There are $N = 3$ UAVs collecting the measurements. The starting location of the UAVs is randomized within a randomly placed 40m $\times$40m rectangle. This simulates a UAV swarm being deployed from a common starting area. We use the RMSE as the metric to evaluate the accuracy of CG prediction. We only evaluate the accuracy at unvisited locations $\mathcal{Q}_P \backslash \mathcal{V}_{1:T}$, since the accuracy at visited locations is perfect due to the assumption of noiseless measurements. We define a utility binary vector variable $\tilde{\mathbf{z}} \in \mathbb{Z}_2^{|\mathcal{Q}_P|}$, where $[\tilde{\mathbf{z}}]_j = 0$ if the location $\tilde{\mathbf{q}}_j \in \mathcal{Q}_P$ is obstructed by a building or if it is in $\mathcal{V}_{1:T}$, and $[\tilde{\mathbf{z}}]_j = 1$ otherwise. Then, the RMSE is defined as: $\sqrt{\frac{1}{||\tilde{\mathbf{z}}||_1} \Delta_k^T \text{diag}\left(\tilde{\mathbf{z}}\right) \Delta_k}$. We show the RMSE as a function of number of steps T per UAV in Fig. 2.4 on T4 dataset. We compare our DL CG predictor against Kriging interpolation and a 3D-map-blind predictor. The 3D-map-blind approach is identical to our proposed DL approach except it does not use 3D maps as an input. We evaluate the prediction methods for different prediction altitudes h_P. First, we can observe that the 3D-map-blind approach performs significantly worse compared to the proposed approach for $h_P = 10$ m, which is why we do not evaluate it for other altitudes. This implies that our proposed approach successfully uses 3D maps for prediction and also that 3D maps are particularly useful when transmitter location is unknown. Furthermore, the proposed DL CG approach performs significantly better than Kriging interpolation, even though Kriging interpolation relies on transmitter location. This is achieved through the use of 3D maps and deep learning for CG prediction. The gap between Kriging interpolation and the proposed DL CG predictor decreases with increasing h_P. This is likely due to the fact that at higher altitudes, the channel gain is easier to predict due to line-of-sight channel being more common between receiver and transmitter.

Next, we evaluate the accuracy of the DL CG predictor as a function of CG after $T = 200$ steps per UAV. In Fig. 2.5, red bars correspond to the RMSE for different CG value bins. There are 16 CG bins in the figure between -240 dB and -80 dB. The figure is obtained by grouping all of the locations in T4 environments based on their corresponding CG bin and then taking the RMSE in each group. From the figure, we observe that the prediction RMSE is lower for higher CG values. This likely occurs because there are more training points for higher CG values in the training data, which skews the accuracy of the predictor towards higher CG values. We also evaluate the accuracy of variance prediction $\Sigma_\theta(\mathbf{y}_{k,1:T}, \mathbf{M})$. We introduce a goodness of fit metric which is equal to the log of average ratio of the square prediction error over the predicted variance at any location j: $\log \mathbf{E}_j \left[\left([\Delta_k]_j^2 \right) / \left([\Sigma_\theta(\mathbf{y}_{k,1:t}, \mathbf{M})]_{j,j} \right) \right]$. The goodness of fit value should be ideally close to 0, which would happen if variance prediction is equal to the observed error. Low absolute value of goodness of fit is necessary for the output $\Sigma_\theta(\mathbf{y}_{k,1:T}, \mathbf{M})$ to be useful for the proposed path planning algorithms. The blue

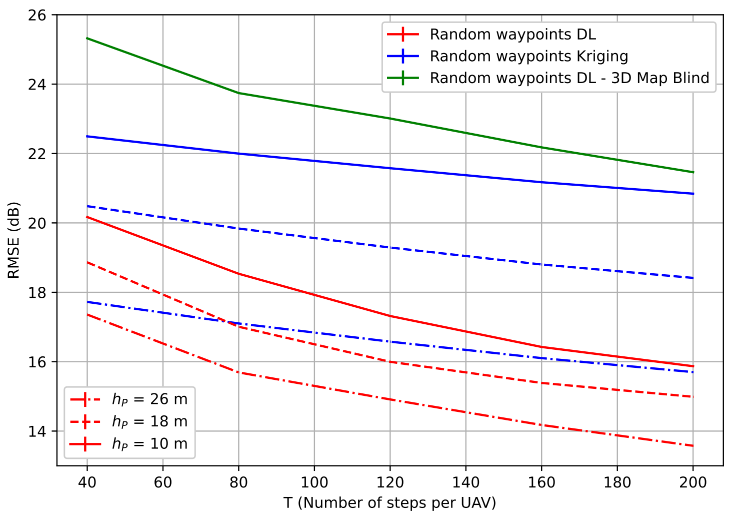


Figure 2.4: RMSE of CG prediction for different prediction altitudes h_P and different number of moved steps per UAV T. The measurements are collected on random UAV paths using 3 UAVs.

bars in Fig. 2.5 correspond to the goodness of fit values for different CG bins. The absolute values of goodness of fit are close to 0 across all CG bins and are generally positive, which indicates that the predicted variance is on average lower than the actual error. Overall, the absolute value of goodness of fit is lower for lower CG values, where the RMSE is also high.

2.8.6 Computational delay of proposed active channel gain prediction approaches

We evaluate the proposed active prediction approaches in terms of path planning compute delay measured in seconds for $N = 2$ and $N = 3$ (Fig. 2.6). The results are obtained on a workstation with a AMD Ryzen Threadripper PRO 5975WX 32-Core CPU and an NVIDIA GeForce RTX 3090 GPU. The GPUs were used for execution of neural networks whenever applicable. The active prediction approaches were implemented in Python. Path planning was accelerated by transforming various path-planning operations such as forward value iteration into array or matrix operations using NumPy library. Tensorflow library was used for implementation of DL components. The greedy Kriging and DL approach have identical computational delay since the path planning algorithms have identical computational

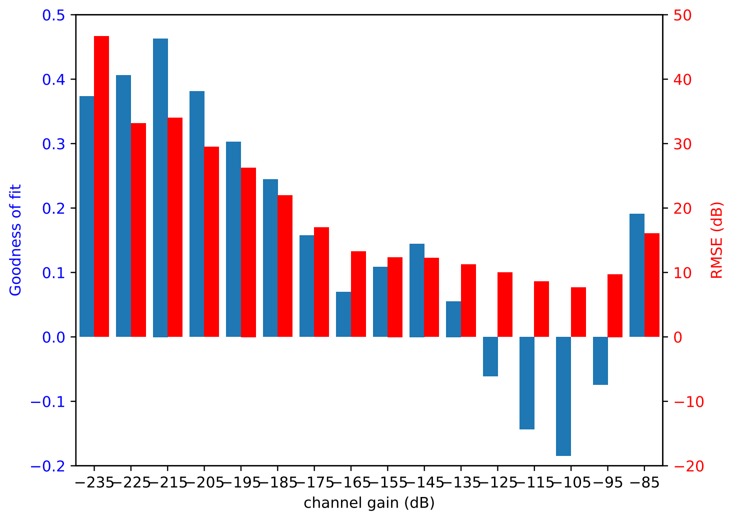


Figure 2.5: RMSE and goodness of fit of DL CG predictor for different CG values.

complexity. The path planning delay for the proposed DL approach is due to the delay of the RL policy DNN, so it is dependent on the size of the DNN and the GPU used for execution. On our workstation, the proposed DL approach is significantly faster than the greedy approaches. The highest delay approach is the proposed Kriging approach, whose complexity scales exponentially with N. Given our Python implementation and capabilities of our workstation, running the proposed Kriging approach for $N > 3$ is not feasible. The complexity of path-planning of this approach could be reduced by down-sampling the AoI $\mathcal{Q}$ to reduce the state space size or by dividing the UAV swarm into clusters of UAVs whose path planning is performed independently. However, this is beyond the scope of this chapter and so we limit the evaluation of the proposed Kriging approach to $N \leq 3$.

2.8.7 Evaluation of proposed active channel gain prediction approaches

Next, we evaluate the performance of the proposed active prediction approaches in terms of the prediction RMSE and compare them to the benchmarks described in Sec. 2.8.4. In Fig. 2.7, we display the results for three coordinated UAVs for $h_P = 10$m. The starting location of the UAVs is randomized within a randomly placed 40m ×40m rectangle. The

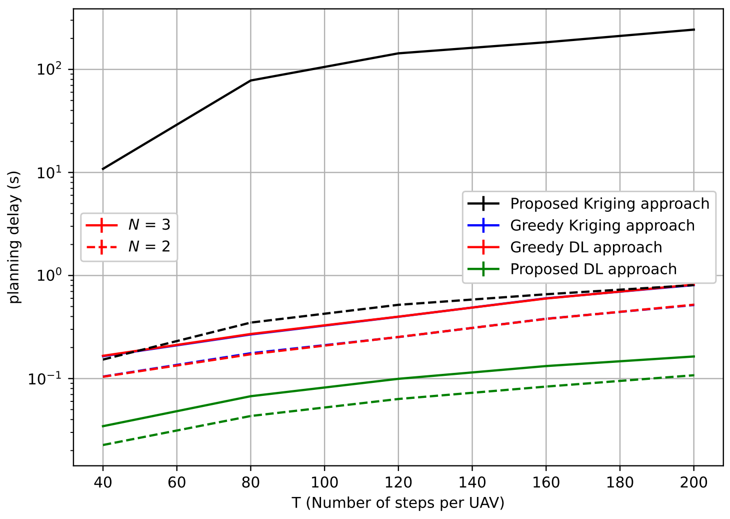


Figure 2.6: Compute delay of path planning for proposed active prediction approaches for different numbers of UAVs N.

proposed active DL CG prediction approach and the proposed active Kriging approach outperform their greedy and random waypoints benchmarks. We can observe a significant gap in RMSE between proposed approaches and their random waypoints benchmarks, which demonstrates the importance of optimal path planning for measurement collection. The proposed active Kriging approach also outperforms the greedy Kriging benchmark. The gap exists because the proposed active Kriging approach design paths that maximize the joint entropy of measured CGs instead of independently moving the UAVs towards locations with highest predicted variance. For similar reasons, the proposed active DL prediction approach that relies on RL for path planning outperforms the greedy DL benchmark. Furthermore, greedy measurement collection may not be optimal for DL-based predictors since we do not know how a deep neural network predicts channel gain, therefore RL-based measurement collection can have an advantage over greedy measurement collection. Overall, the proposed active DL prediction method performs better than the proposed active Kriging prediction method in terms of RMSE. However, both proposed methods have practical advantages. Kriging-based active prediction has the advantage of not requiring extensive training data and 3D map knowledge, while the proposed DL approach does not require the knowledge of transmitter location and provides higher accuracy.

We also evaluate the proposed algorithms and the benchmarks for scenarios when the starting locations of the UAVs are randomized across the entire AoI. This can for example simulate the case when UAVs have been previously deployed to complete different tasks and have moved far apart before commencing collection of CG measurements. These results are shown in Fig. 2.8. We can see that the RMSE across all approaches decreases, which occurs because the UAVs are more spread out across the AoI. Moreover, the gap between the proposed approaches and greedy benchmarks decreases due to UAVs being more likely to move in non-overlapping areas since their starting locations are far apart. In Fig. 2.9, we show the results for 5 UAVs with starting locations randomized within a randomly placed 40m $\times$40m rectangle. The RMSE across all approaches decreases compared to the results with 3 UAVs. Moreover, the gap between the proposed active DL prediction approach and its greedy benchmark decreases compared to the scenario with 3 UAVs. This indicates that for DL prediction, coordination is less important for a larger number of UAVs.

2.9 Summary

In this chapter, we developed methods for prediction of CG that use environment-specific features such as building maps and CG measurements to achieve a high level of prediction accuracy. We assume that measurements are collected using a swarm of coordinated UAVs. We developed two active prediction approaches based on DL and Kriging interpolation. We trained and evaluated the two proposed approaches in a ray-tracing-based channel gain simulator. Using channel simulations based on the ray-tracing approach, we demonstrated the importance of active prediction compared to prediction based on randomly collected measurements of channel gain. Furthermore, we showed that using DL and 3D maps, we can achieve high prediction accuracy even without knowing the transmitter location. We also demonstrated the importance of coordinated path planning for active prediction when

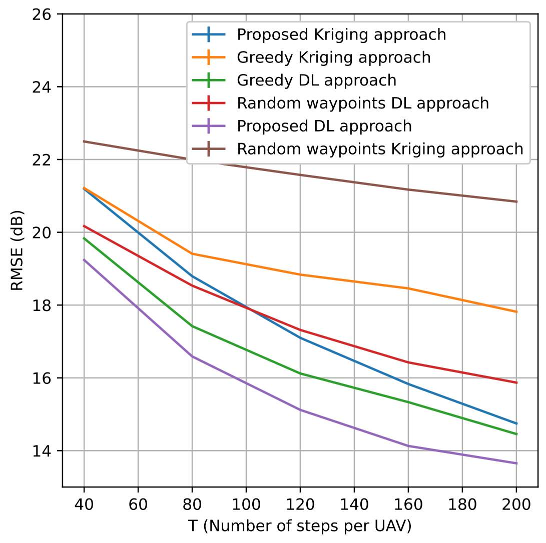


Figure 2.7: Prediction RMSE for three UAVs for the proposed approaches and the benchmarks. The starting location of the UAVs is randomized within a randomly placed 40m ×40m rectangle.

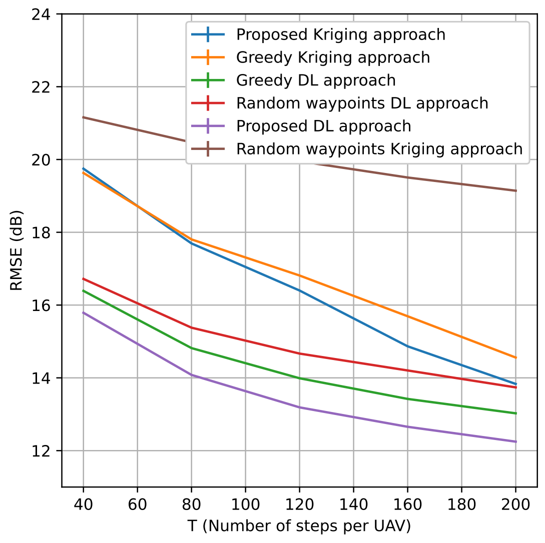


Figure 2.8: Prediction RMSE for three UAVs for the proposed approaches and the benchmarks. The starting location of the UAVs is randomized within the entire AoI.

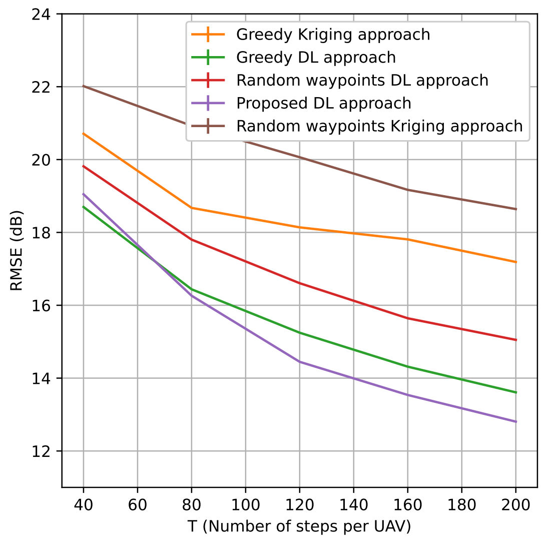


Figure 2.9: Prediction RMSE for five UAVs for the proposed approaches and the benchmarks. The starting location of the UAVs is randomized within a randomly placed 40m ×40m rectangle.

using multiples UAVs compared to UAVs collecting measurements independently in a greedy manner.

CHAPTER 3

UAV Access Point Placement for Connectivity to a User with Unknown Location Using Deep RL

In recent years, unmanned aerial vehicles (UAVs) have been considered for telecommunications purposes as relays, caches, or IoT data collectors. In addition to being easy to deploy, their maneuverability allows them to adjust their location to optimize the capacity of the link to the user equipment on the ground or of the link to the basestation. The majority of the previous work that analyzes the optimal placement of such a UAV makes at least one of two assumptions: the channel can be predicted using a simple model or the locations of the users on the ground are known. In this chapter, we use deep reinforcement learning (deep RL) to optimally place a UAV serving a ground user in an urban environment, without the previous knowledge of the channel or user location. Our algorithm relies on signal-to-interference-plus-noise ratio (SINR) measurements and a 3D map of the topology to account for blockage and scatterers. Furthermore, it is designed to operate in any urban environment. Results in conditions simulated by a ray tracing software show that with the constraint on the maximum number of iterations our algorithm has a 90% success rate in converging to a target SINR.

3.1 Introduction

Due to their high mobility and low cost, unmanned aerial vehicles (UAVs) have found their way to many applications in recent years, including package delivery, law enforcement, search and rescue, etc. Following this trend, UAVs are getting an increased attention in the telecommunications sector. Deploying UAVs as aerial basestations has recently emerged as an idea to respond to high localized traffic demands in the next-generation cellular networks [LC17, WXZ18, MSB16]. Using UAVs in such way provides the opportunity to exploit their agility of motion to improve the air-to-ground link capacity by optimal air placement. UAVs can also be utilized for data harvesting in IoT or as data caches and in these applications it is also important to maximize the air-to-ground capacity by optimal placement.

In this chapter, we are interested in optimizing the capacity of the channel between the UAV and a ground user in an urban environment. This is a challenging problem considering that the environment between the UAV and the ground equipment can be abundant in scatterers and therefore hard to account for analytically and numerically. Nevertheless, the said problem has been addressed in literature before, under different assumptions. In most cases, however, the solutions are based on the assumptions that the ground user locations are known and that the wireless channel can be predicted with a simple model.

The problem of a UAV relay placement has been considered mostly for line-of-sight (LOS) channels. In [WRC18, JZC12, ZZL16] transmit power and placement of a UAV relay are jointly optimized. However, the authors' solutions apply only to a LOS propagation channel, which makes this approach less applicable in the environments with scattering and obstacles, such as cities. Furthermore, all of [WRC18, JZC12, ZZL16] assume that the locations of the users and channel propagation characteristics are known. In [MM17], a method for optimizing the location of a ground unmanned vehicle is proposed. The approach relies on user location and assumes a fading channel. The algorithm predicts the channel quality across the entire map from a small number of measurements and then using stochastic dynamic programming an unmanned vehicle is optimally routed. While this paper considers ground vehicles, it is relevant to our work since their approach can apply to aerial vehicles flying at a fixed altitude.

In addition to statistical models, some of the previous approaches have also utilized topology maps. The work in [LOC16] considers a UAV swarm that relays communication between users on the ground in an urban environment. The approach relies on known user locations and topology maps to perform swarm particle optimization of placement. Works [CG17, EGG18a, EGG18b] utilize 3D topology maps to help UAV placement optimization. Additionally, [CG17] uses a statistical channel model and the user location to perform optimization. While [EGG18a] does rely on user locations, it does not need to know the channel model parameters as these are learned. The algorithm in [EGG18b] simultaneously learns user locations and parameters, but as a result, the approach has an extensive learning phase.

Seeking to develop an exploration algorithm that performs learning and placement optimization simultaneously we turned to reinforcement learning. Reinforcement learning has been previously applied to similar problems. [LLC18] uses table-based Q-learning for optimal placement of aerial basestations with the knowledge of user locations. However, since the inputs to this algorithm are only user and UAV locations, it cannot perform outside of the environment it has been trained on. Similarly, [CEG19] uses received signal strength at the UAV and the UAV location to track indoor users with a shallow Q-learning algorithm. However, the paper only considers two indoor scenarios and the training and testing dataset are the same. Therefore, it is not clear whether the algorithm could perform in a new environment.

We address the problem of optimal UAV placement assuming that the user location is not known. Algorithms that rely on statistical models of the channel may fail to generalize to all environments since the local topology can significantly differ from statistical predictions. To that end, we use deep reinforcement learning to obtain a model-free algorithm for UAV positioning. The proposed algorithm relies on the knowledge of local topology and does not require the knowledge of the user location. Deep reinforcement learning allows us to take a high-dimensional input that is the topology map and use it alongside SINR measurements collected on the trajectory of the UAV to predict the optimal direction of motion. We test our performance in a realistic environment that emulates the wireless channel using a ray-tracing software. Furthermore, the training and the testing dataset are different.

The rest of this chapter is organized as follows. In Section 3.2 we introduce the relevant reinforcement learning background, define our problem and describe how reinforcement learning can be used to tackle it. In Section 3.3, we describe the simulation environment. Sections 3.4 and 3.5 are dedicated to results and conclusions, respectively.

3.2 UAV Placement using Q-learning

In this section, we first introduce the necessary background in reinforcement learning in part 3.2.1. Next, in subsection 3.2.2, we formulate our problem as a partially observable Markov decision process (PO-MDP) and in subsection 3.2.3 we discuss how we apply Q-learning to this PO-MDP.

3.2.1 Reinforcement learning background

Reinforcement learning (RL) is the branch of machine learning that is concerned with making sequences of decisions. It is mainly concerned with problems that can be casted as a Markov decision process (MDP). In an MDP, an agent $\mathcal{A}$ is situated in an environment $\mathcal{E}$. At each timestep t, the agent is in a state s_t and takes an action a_t, it receives a reward r_t while moving into the state s_{t+1}. In a partially observable MDP (PO-MDP), the full knowledge of the state of the environment is not known to the agent and in that case it will only have access to an observation of the state. This observation then replaces the function of the state in the reinforcement learning algorithms.

The objective function in reinforcement learning is often the expectation of the discounted reward, $\mathbb{E}\sum_{t=0}^{\infty}\gamma^t r_t$, where γ is the discount factor. Reinforcement learning methods can broadly be classified into two categories: policy learning and Q-learning. In policy learning methods, the goal is to learn the optimal policy function that defines $\pi(a|s)$, which is the probability of taking the action a in the state s. The policy is often deterministic and in that case $\pi(a|s)$ defines a single action in each state. In Q-learning methods, the goal is to learn the Q-value function

$$Q(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a],$$

that defines the expected reward in a state s, after taking the action a. If the Q-value function is known, the optimal action in a state s is then $\text{argmax}_a \; Q(s,a)$. Deep Q-learning is an extension to the Q-learning paradigm that uses deep learning models, such as convolutional neural networks and recurrent neural networks to approximate $Q(s,a)$. Furthermore, Q-learning is a model-free reinforcement learning technique, meaning that it does not rely on known dynamics of the system.

One of the first deep Q-learning algorithms was proposed by Deepmind and was successfully demonstrated on Atari video games [MKS13]. The algorithm was named the deep Q-network (DQN) algorithm. In it, the Q-value function is parametrized by a neural network Q_θ, with parameters θ. An estimate of the true Q-value at time t, Q_t, can be obtained by using a single sample estimate of the Bellman backup operator

$$\widehat{\mathcal{T}Q_t} = r_t + \max_{a_{t+1}} \gamma Q_\theta(s_{t+1}, a_{t+1}) \tag{3.1}$$

This is called a single sample estimate because only the reward r_t at the current time instant t is used to approximate the infinite horizon Q-value function.

In order to approximate Q by Q_θ the following minimization is done over sample data,

$$\underset{\theta}{\text{minimize}} \sum_t \left|\left| \widehat{\mathcal{T}Q_t} - Q_\theta(s_t, a_t) \right|\right|^2 \tag{3.2}$$

In the DQN algorithm, the training and the interaction of the agent with the environment happen in parallel. As the agent gathers experience, samples of that experience are stored and the minimization in the Equation 3.2 is done periodically, every τ_L steps, by randomly sampling a batch of B_L recorded samples and applying gradient descent. This is referred to as experience replay. Samples are arrays of data (s_t, a_t, r_t, s_{t+1}) and these are stored in the replay buffer.

The agent interacts with the environment following the ϵ-greedy policy, where at any time t the agent either takes a random action at probability ϵ or the Q-value optimal action $\text{argmax}_a\ Q_\theta(s, a)$ at probability $(1 - \epsilon)$. Over the course of the training, the value of epsilon decreases from 1 to 0. In the implementation of DQN, there is an additional Q_θ, called the target Q-network. The target Q-network is used in the Bellman backup operator but it is not optimized over. Instead, it is periodically copied from the main Q-network. The target Q-network is included to improve the stability during training.

The original vanilla DQN algorithm has been improved upon over the years. The two expansions that we will use are double Q-learning [VGS16] and dueling networks [WSH15]. For the sake of brevity we omit the details of these algorithms and the reader is referred to the cited works for more information.

3.2.2 UAV placement problem as a PO-MDP

We now formulate the UAV placement problem as PO-MDP. We consider the scenario of a UAV located in an urban environment communicating to a radio device on the ground. The area topology is such that LOS connection to the ground user is not always possible, and the communication will often occur over non-line-of-sight paths. At time zero, the UAV and the ground device are located at random positions and the goal of the UAV is to adjust its position so as to increase the SINR at the UAV. We assume that the UAV receives some signal power from the user on the ground to begin with. The UAV moves until an SINR threshold is reached or until maximum time for optimization expires.

In the following, we describe the mechanics according to which the UAV moves around.

Since exploring the entire 3D space is a complex task, we restrict the motion of the UAV to the horizontal plane and assume that its altitude is kept constant. We do this as a relaxation but it is worth pointing out that the optimal position for a UAV will often be at the lowest allowed altitude since this brings it closest to the ground user. The UAV can only move around buildings or fly above buildings that are below its altitude and it makes adjustments in its position at discrete time steps. We restrict the directions of the motion of the UAV to the four orthogonal horizontal directions and the motion step size d_S is fixed. We impose this constraint because Q-learning lends itself better to tasks with a discrete set of actions. With these restrictions on the motion, the UAV effectively moves in a uniform plane grid space that spans the environment.

We assume that the UAV location is known. Furthermore, the UAV has access to a 3D map of the environment that maps all the buildings, which can be drawn from a database. 3D maps of major urban areas are generally available and easy to acquire. We also make the assumptions that the channel is slow fading and that the user location does not change significantly over the course of optimization. Furthermore, we assume that there is a sufficient backhaul capacity between the UAV and the core basestation, so we only focus on optimizing the capacity between the UAV and the ground device.

3.2.2.1 Observation space

Since the full state of the system in which the UAV operates is not available, the agent in our algorithm relies on two types of observations of the environment to drive its decision making. The first type of observation is the 3D map of the local area. The local area in our case is a square area of side l_O centered at the current location of the UAV. The 3D map information is compressed into a 2D array representation, where each entry represents a grid point in the local area and the value of each entry corresponds to the height of the terrain at that point relative to the UAV altitude. We use heights relative to the UAV altitude to make the algorithm adjustable to different starting UAV flying heights.

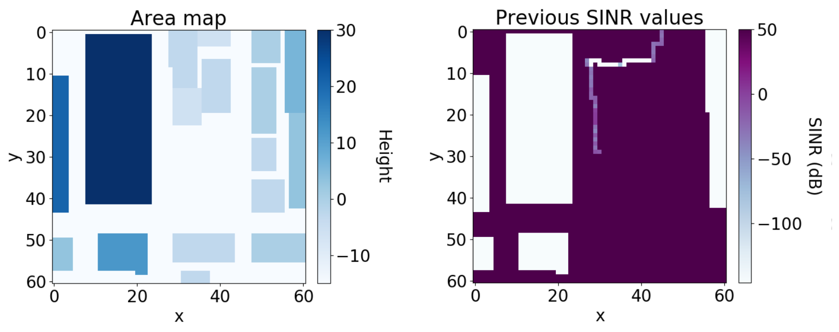


Figure 3.1: Example observation for an agent moving according to a random exploration policy. Note that the SINR values of areas that are obstructed and therefore unreachable are set to a very low value (-150 dB), while the areas that have not yet been visited are set to a very high value outside of the possible range of SINR values (50 dB).

The second type of observation that the agent uses are the SINR measurements at previously visited locations in the local area. These are also stored in a 2D array with grid point locations matching the locations of the grid points in the topology observation. The value of each entry is the measured SINR value. To complete the array, we populate the entries with unknown SINR values with a high value P_H outside of the regular range. Furthermore, the points that are blocked by buildings and cannot be visited are populated with low values P_L outside of the span of possible SINR values. With successful training, the algorithm will learn the significance of P_H and P_L. Example observations are shown in Figure 3.1.

3.2.2.2 Action space

Since the action is the motion of the UAV at each time step, it can take on the values (0, d_S), (0, -d_S), (d_S, 0), (-d_S,0), where each vector represents the displacement in the x- and y-coordinates.

3.2.2.3 Reward

The UAV receives a reward equal to the difference between the SINR in the next state and the SINR in the current state. This incentivizes the agent to move towards higher SINR.

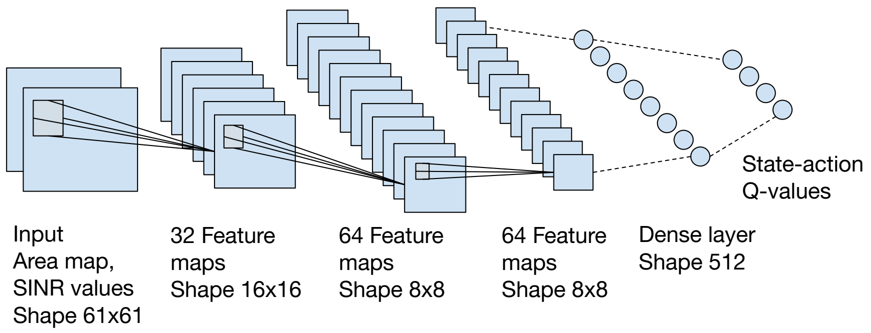


Figure 3.2: Neural network model used as the Q-network.

Furthermore, we assign a constant exploration reward c_E which the agent receives for visiting a new location. We empirically established that an exploration reward incentivizes the agent to explore further away from its starting location, which results in better performance. The reward is mathematically expressed as

$$r_t = \mathrm{SINR}_{t+1} - \mathrm{SINR}_t + c_E \delta_t^E,$$

where δ_t^E is an indicator function activated when the UAV visits a new location.

3.2.3 Deep Q-learning implementation

With our problem casted as a PO-MDP, we can apply the deep Q-learning algorithm. For our application, we utilize double Q-learning [VGS16] and dueling networks [WSH15] extensions to the base DQN algorithm. The choice of the neural network model used as the Q-network depends on the application and therefore it needs to be carefully selected for optimal performance. The neural network model we used is shown in Figure 3.2. At the input, there are two 2D arrays, corresponding to the SINR and topology observations described in the previous subsection. As displayed, we use 3 convolutional layers with varying number of filters and with each layer having a different filter size. There are two fully connected layers, with the final layer output corresponding to the Q-value for each of the possible actions. We use ReLU as the activation function after each layer prior to the last layer. The layer enabling the dueling networks extension is located before the final layer, however we do not show it in the figure for clearer presentation. Additionally, we used batch normalization and dropout with probability p_D to accelerate the training of the neural network and for regularization purposes.

3.3 Simulation environment

We use two separate environments for the training and the testing of our algorithm, shown in Figure 3.3 and Figure 3.4, respectively. Both spaces are meant to resemble a typical medium-elevation urban area.

In order to create realistic conditions to train our Deep RL model, we used a ray-tracing software called Wireless InSite [rem] to emulate the wireless channel. For a given user on the ground we measure the SINR across a uniform grid of points at a fixed height that corresponds to the UAV flying altitude. The grid points are separated by 4 meters and they span the entire environment. The UAV altitude is set to 10 meters. To generate the training data, the user radio was placed at 27 locations uniformly spanning the training environment, while for the testing data we placed the user at 25 different locations in the testing environment. The numbers were decided such that user locations uniformly cover the entire space, while still taking a feasible amount of time to make calculations for in the ray tracing software. The users transmit a narrow-band signal of 20 dBm power using a frequency of 800 MHz. We introduce a Gaussian noise and a Gaussian background interference across the space with the average combined power of -104 dBm. Half-wave dipole antennas with vertical orientation are used at the user and the UAV.

The SINR measurements were then exported and used to build a training and a testing environment in software that the DQN algorithm can interact with. At each realization or episode of the environment we use the SINR measurements for a random user location and the UAV is placed at a random location on the grid. The locations that the UAV can visit

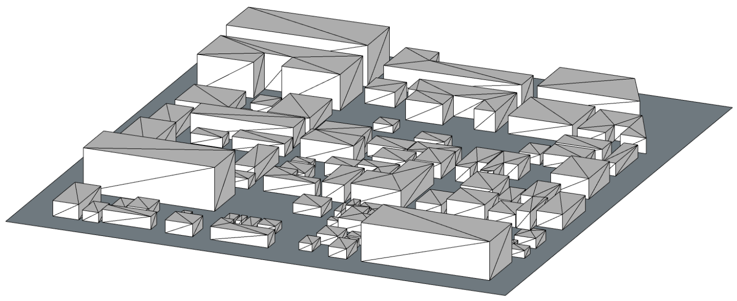


Figure 3.3: The training urban environment. Approximate size: 550x500 m.

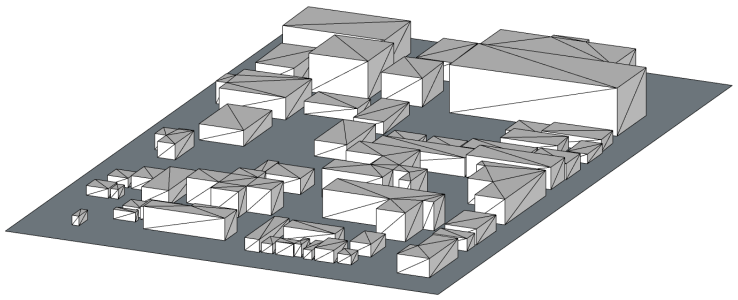


Figure 3.4: The testing urban environment. Approximate size: 400x500 m.

correspond to the ones where SINR measurements were recorded. The episode finishes if the UAV reaches the required SINR or the maximum number of steps that the UAV can take is exceeded.

3.4 Results

In the first part of this section we describe the details of the training of our algorithm and demonstrate that it learns how to move the UAV in order to increase SINR. In the second part, we validate the performance of the algorithm in the testing environment and compare it to a genie algorithm in terms of steps made until convergence to the required SINR.

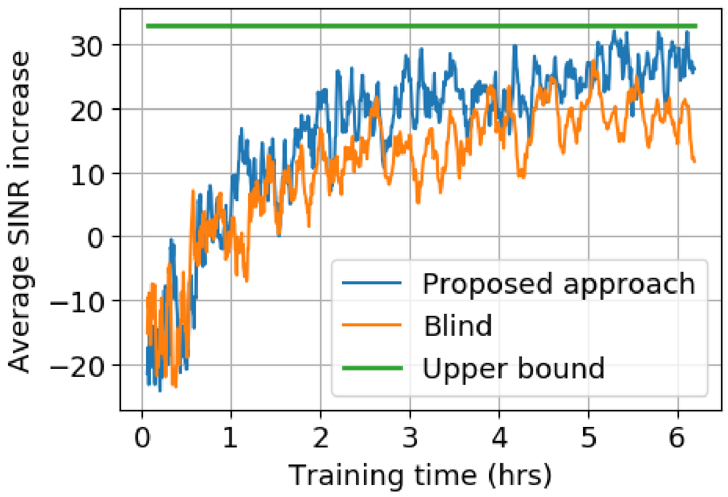


Figure 3.5: The training results for the proposed model and blind model that does not rely on topology information.

3.4.1 Training

We train the DQN algorithm in the training environment described in the previous section. The maximum number of steps during an episode t_{MAX} was set to 800 and the target SINR P_T, was set to 5 dB. When deploying the UAV on the map we ensure that it is not placed in a dead zone with no signal reception, as this would be outside of our problem statement. In the ray-tracing simulator, these regions occur when there are no direct or reflected paths that can reach the UAV. For regularization purposes, we rotate the coordinate system of the map by a random multiple of $90°$ every training episode. This ensures that the algorithm does not become biased towards moving in any particular direction over the course of the training.

We use the ϵ-greedy policy for exploration, however the agent's random actions are steered. Namely, the agent never takes a random action that would lead to it leaving the map or colliding with a building. Furthermore, the agent repeats the action it has taken in the previous step at probability 0.4ϵ and takes any random allowed action at probability 0.6ϵ. The repeated movements lead to the agent exploring a larger area through random walk in the early training stage, instead of staying confined to the local space around the starting location. The optimal action $\text{argmax}_a\ Q_\theta(s, a)$ is taken at probability $1 - \epsilon$. We also ensure that the agent never leaves the map or collides with a building when taking actions according to the DQN. This is done by choosing an action that gives the highest Q-value while still being a legal movement in the environment. As the UAV moves around, its experience samples are stored in a replay buffer that can store up to 5×10^5 samples and when this limit is reached the oldest samples are thrown out to make space for the new ones.

The parameter values used for the training of the DQN algorithm are shown in Table 3.1. They were selected after tuning for best performance. During minimization we employ gradient clipping and also anneal the learning rate. The width and length of the observation is 61 grid points or 244 m. The movement step size d_S was 4 m.

The training results are shown in Figure 3.5. To keep track of the progress of training, we measure the average SINR increase from the SINR at the start of the episode to the SINR at the end of the episode, over the most recent 100 episodes.

To demonstrate the benefits of using 3D maps we evaluate a DQN algorithm that doesn't rely on the 3D map but is otherwise identical to our proposed algorithm. We refer to this algorithm as the 'blind' algorithm. The blind algorithm only has a 2D SINR array as an input and the regions blocked by buildings are not populated by P_L but instead left as P_H.

Table 3.1: DQN parameter values used in training

Description	Parameter
Exploration reward	$c_E = 1.2$
Discount factor	$\gamma = 0.99$
Training batch size	$B_L = 20$
Training interval	$\tau_L = 3$
Dropout probability	$p_D = 0.4$

In training, we use the same parameter settings for the blind and the proposed algorithm.

Furthermore, we include an upper bound on the mean SINR increase in the training stage. It is calculated by taking the average of the SINR differences between the SINR at all possible starting UAV locations and P_T, for all user locations. This is an upper bound on the mean performance across a large number of episodes.

The results in Fig. 3.5 show that the learning capacity of our proposed algorithm is larger than that of the blind algorithm. The intuitive explanation for this is that the algorithm with the knowledge of the topology is more efficient in exploring the space because it can eliminate obstructed areas and because the building knowledge combined with SINR measurements can give it indication where to move to find better SINR. The performance curves are noisy due to the nature of training through experience replay and due to the fact that over a 100 episodes the algorithm only experiences a subset of the training environment, which makes the difficulty vary as some user locations are harder to find optimal paths for than others.

3.4.2 Testing

In the testing stage, the algorithm is placed in an entirely new environment and relies only on the trained neural network Q_θ to guide the movement of the UAV. We use the copy of Q_θ that had the highest performance during training. We also introduce a small amount of randomness in decision making, which we found to lead to a better performance. The agent takes a random action at probability 0.096. The value of P_T for the testing case was again set to 5 dB and the maximum number of steps t_{MAX} was reduced to 500, since the testing environment is smaller than the training environment.

In Fig. 3.6, we show a number of trajectories of the UAV moving according to our algorithm, where the target SINR was reached in less than t_{MAX} steps. The upper figure shows the building height relative to the UAV altitude and the lower figure shows the heatmap of the SINR across the map. The user location is the same in each episode, and we place the

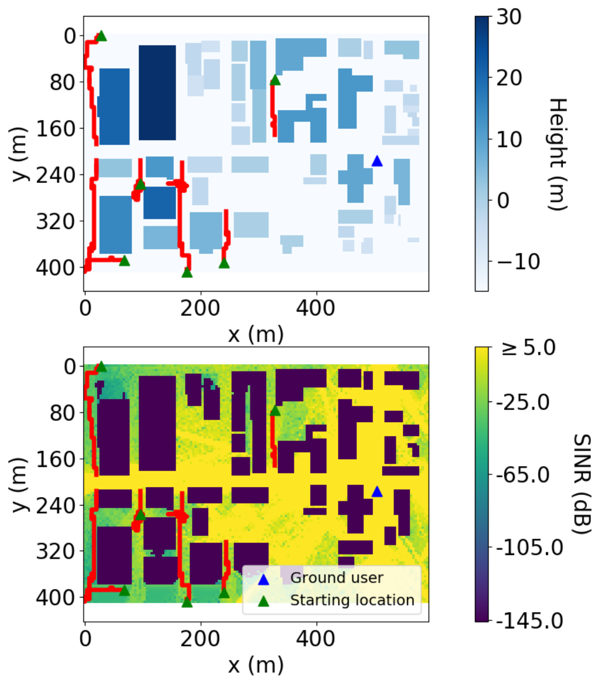


Figure 3.6: Successful trajectories of the UAV moving according to our algorithm. The upper figure shows the building height relative to the UAV altitude and the lower figure shows the heatmap of the SINR across the map. The green triangle markers represent the starting positions of the UAV. The blue marker represents the location of the user on the ground.

UAV at random locations. We can see that the algorithm is capable of following the direction that leads it to improving the SINR and it is also capable of correcting itself when realizing that the current direction of motion is not leading it to better SINR. In the instances where the UAV loops around its location, we can infer that the algorithm explores several path options before settling on the one it deems optimal. Buildings are avoided by the algorithm and it can be inferred that the algorithm eliminates path options because of them. An evidence for the latter is that the UAV tends to move parallel to the building edges, for example.

Finally, we analyze the performance of our algorithm over many realizations. To get a reference on how fast our algorithm converges to an optimal point we used a genie algorithm for optimal placement. The genie algorithm has a complete knowledge of the SINR distribution and building topology, and uses dynamic programming to find the shortest path

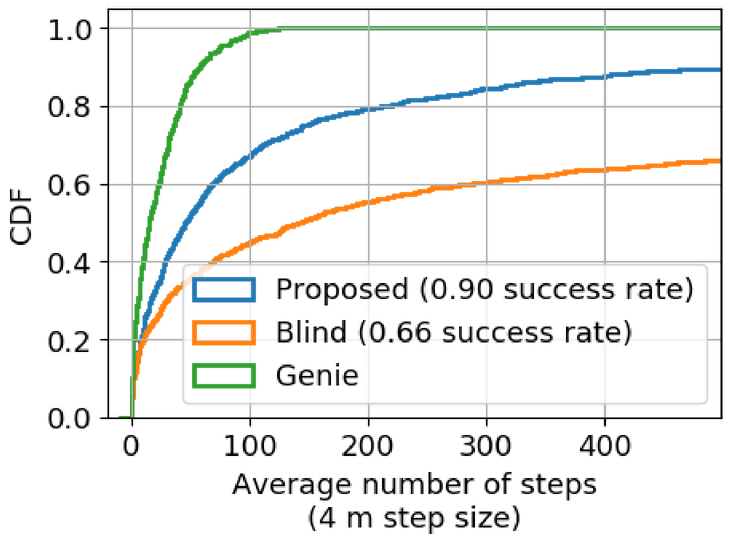


Figure 3.7: The CDF of the number of steps until convergence to the sufficient SINR for the proposed approach, the blind approach and the genie algorithm.

to a location with sufficient SINR. We run the proposed and the blind algorithm for 1000 realizations across the entire testing data set over different user positions. The results for the number of steps made until convergence to the sufficient SINR for all three algorithms are shown in Fig. 3.7. The median number of steps until convergence for the proposed algorithm is 44 and 69 for the blind algorithm. Furthermore, the proposed algorithm is 90% successful in under t_{MAX} steps compared to 66% of the blind algorithm. Therefore, we show that the knowledge of topology map assists our algorithm even in a novel environment. The median number of steps required for the genie algorithm is 14. The difference in the number of steps required relative to the proposed algorithm is due to the fact that our algorithm has to explore the space to find good SINR since it does not where the points with sufficient SINR are a priori.

3.5 Summary

In this chapter, we used deep reinforcement learning methods to optimize the placement of a UAV communicating to the user on the ground. We consider the case where the ground user location is not known and use topology data to replace statistical models of the channel. We were able to achieve 90% success rate in moving the UAV to a location that has a sufficient SINR within a limited number of steps. Moreover, our reinforcement learning approach stands out in that it can be applied in any urban environment. Our future work will focus on the scenario where the ground user is moving over the course of optimization. Furthermore, we will explore how the deep reinforcement learning techniques can be used to simultaneously optimize the locations of multiple UAVs serving multiple users on the ground.

CHAPTER 4

Cooperative Computing in Vehicular Micro Clouds for Emerging Vehicular Applications

We consider the use of vehicular cooperative computing to support sensing-based computationally-intensive and delay-constrained vehicular applications such as cooperative perception, augmented reality and platooning, which have been recently proposed to improve the driving safety and efficiency by fusing sensing data from multiple vehicles. To enable such computationally demanding applications, intelligent vehicles have traditionally relied on independent processing using on-board computers, which requires expensive computing hardware. Cloud computing or edge computing can be used to offload the computing load, but these solutions suffer from low-coverage and high latency. A group of intelligent vehicles can collaborate using 802.11 vehicle-to-vehicle (V2V) communication to form a VMC and operate as virtual edge servers, which can be used to complete computationally intensive tasks without requiring powerful onboard hardware and without relying on cloud/edge computing. We develop resource assignment and scheduling (RAS) methods to optimally allocate vehicular computing resources and schedule computing tasks for cooperative computing of the considered applications. Our proposed RAS methods are resilient to rapid changes in achievable data rates between vehicles, which occur due to vehicular mobility. Furthermore, when performing scheduling, the proposed methods ensure that the capacity limits are not exceeded, which can cause congestion and tasks not being delivered. Finally, the proposed methods are computationally efficient and can be used in real-time. In a realistic simulated environment, we evaluate the benefits of cooperative computing compared to independent computing using our RAS methods under factors such as task features, traffic characteristics and incumbent interference. The simulation results also demonstrate the importance of accurate congestion control and data rate prediction for reliable cooperative computing.

4.1 Introduction

In this work, we focus on the use case of cooperative computing to support sensing-based computationally-intensive and delay-constrained vehicular applications such as cooperative perception, augmented reality, driving assistance and navigation. The common characteristics that the considered vehicular applications share are that their computing tasks are repeatedly processed at a certain rate and the input data for the tasks is based on the sensor data collected by the vehicles. For example, cooperative perception is a delay-sensitive application where sensor data from multiple vehicles is regularly pooled and processed to expand the field of view of the vehicles. The processing to combine the pooled data has a significant computational load.

To enable computationally demanding applications, intelligent vehicles have traditionally relied on independent processing using on-board computers. However, independent processing is cost inefficient since powerful hardware is required to meet the joint computing demand of all computationally demanding applications that may be active at the same time at a single car. Furthermore, independent processing prohibits deployment of novel applications on older generations of vehicles since they may not have sufficient computing capabilities to support them. Cloud computing [AFG10] or edge computing [SCZ16] are possible solutions, in which networked servers are accessed using wireless communications to satisfy the computational requirements. Generally, the main difference between cloud and edge computing is the proximity of the servers to the end users, with cloud servers being located on the Internet, whereas edge servers are located closer to the end users and perform various tasks in conjunction with cloud data centers. Communications with cloud or edge servers heavily rely on wireless and underlying backbone networks, which can suffer from potentially large end-to-end latency, limited capacity and low coverage. This makes cloud and edge computing unsuitable for delay-sensitive vehicular applications.

Cooperative computing using VMCs addresses the described limitations of independent processing and cloud/edge computing for sensing-based computationally-demanding and delay-sensitive applications. A group of intelligent vehicles with advanced computation and communication abilities can collaborate using V2V communication to form a VMC, which offers data processing, data storage and communication services as virtual edge servers [HJD17]. The vehicular computing resources of the MC can be used to jointly process computationally intensive tasks.

In this work, we develop resource assignment and scheduling methods to determine how to assign vehicular computing resources and decide the quality of service (QoS) for tasks that will be scheduled for cooperative computing. The QoS determines the task processing rate and the computational complexity of each task, which are not predetermined for the considered applications, but instead can be increased with more available computing resources. For example, consider cooperative perception based on LIDAR frames. Increasing the QoS could mean increasing the resolution of LIDAR frames, which would be more desirable for accurate cooperative perception, but this will increase the computational complexity per each processed frame. Similarly, in cooperative perception, higher QoS can imply a higher frame rate which is more desirable but it also increases task processing rate. The objective function that we aim to maximize is the sum QoS across all tasks.

There are multiple challenges in achieving optimal resource assignment and scheduling for cooperative computing in VMCs. Delivery of tasks for cooperative computing and their input data occurs wirelessly between vehicles. Due to high mobility, achievable data rates between vehicles change rapidly, which impacts the task transmission delay and task delivery success. When scheduling tasks it is necessary to account for these rapid changes to meet the delay constraints of the considered applications and prevent task loss. This makes resource assignment and scheduling for vehicular cooperative computing more challenging compared to cooperative computing in stationary networks. We focus on V2V networking with hardware from one of the IEEE 802.11 standards. In IEEE 802.11 standards, channel access is coordinated between vehicles using 802.11 Distributed Coordination Function (DCF). When performing scheduling it is essential to not exceed the capacity limits imposed by the DCF and cause congestion, which can severely delay the delivery of tasks and their input data. Finally, since resource assignment and scheduling will occur repeatedly in the VMC, these methods must have a computational complexity low enough to run in real time.

We make the following contributions in this paper:

- We develop two resource assignment and scheduling approaches based on mixed-integer linear programming (MILP). The first introduced approach has a lower computational complexity but is less accurate in preventing channel congestion. The second approach has a higher computational complexity but is more reliable in preventing channel congestion. Both proposed resource algorithms adapt to the changes in achievable data rates between the vehicles.

- The complexity of both MILP approaches scales exponentially with factors such as the number vehicles participating in cooperative computing, so they may not be feasible to run in real time in all circumstances. For these scenarios, we develop approximate resource assignment methods that run in polynomial time.

- To account for data rate fluctuations, in our resource assignment and scheduling framework we utilize approaches for prediction of data rate between vehicles. Furthermore, using the data rate prediction methods, we extend our resource assignment and scheduling approaches to dynamically adjust the assignment and scheduling update frequency based on the mobility of vehicles.

- We quantify feasible gains of cooperative computing compared to non-cooperative computing in terms of QoS and as a function of different factors including computing task features, traffic characteristics, vehicular computing resources and incumbent wireless transmitter activity. The presented results are for cooperative computing based on vehicles utilizing IEEE 802.11p standard with a MAC layer that was adapted to closely resemble 802.11 WiFi standards. This makes the conclusions that we make using our simulation results applicable to 802.11 WiFi standards. Vehicular communication is simulated using VEINS simulator [SGD11] for vehicular networks, which provides accurate modelling of IEEE 802.11p MAC and PHY layer. Traffic simulations are conducted using the SUMO simulator for vehicular urban mobility [LBB18].

The rest of this paper is organized as follows. In Sec. 4.2 we review the related work and compare our work against existing approaches. In Sec. 4.3, we explain the models of VMCs, cooperative computing and communications. In Sec. 4.4 we introduce a generic resource assignment problem, which will be used to develop two different approaches for resource assignment with congestion control in sections 4.5 and 4.6. In Sec. 4.7 we discuss methods for achievable data rate prediction and in Sec. 4.8 we develop methods for control of resource assignment duration. In Sec. 4.9 we present the main results of our approaches and in Sec. 4.10 we summarize the conclusions.

4.2 Related work

Generally, we can distinguish the current work on computational offloading for vehicular applications based on where the computing takes place. Most distributed computing approaches assume that tasks are offloaded to edge or cloud servers [CJL15, SLT21, FLW17, CCM23, NZW20, ZLG19]. Some works have considered joint offloading to both nearby vehicles and edge/cloud servers [ZCM19, ZPX18, ZTW19, CWY22, QHC23, KMH20, SGS19, BBG21, WLZ22]. In our work, we focus on cooperative computing and, therefore, task offloading occurs exclusively within a VMCs. Moreover, all vehicles jointly coordinate to utilize the computing and communication resources within a VMC.

Offloading to edge/cloud servers is different from cooperative computing within a VMC for several reasons. First, edge/cloud servers have much higher computational capacities than vehicles but are less abundant. Second, they are usually connected to vehicles via cellular networks. In cellular networks, vehicles can offload their tasks to servers in orthogonal resource blocks allotted by the cellular base station. Therefore, task offloading is generally not limited by network capacity constraints and transmissions from different vehicles can be treated as independent. In a VMC based on a 802.11 standard, tasks are transmitted between vehicles on a shared channel, therefore task offloading is limited by the capacity constraints of the channel. Furthermore, channel access is dictated by the DCF, which makes throughput prediction more challenging than in cellular networks due to contention on the channel.

Next, we discuss current research on vehicle-to-vehicle computational offloading based on the way these approaches model the communications channel. Accurate modeling of the communications is necessary to account for the loss of scheduled tasks due to either inadequate links between vehicles or queue congestion at vehicles. While some prior works have addressed the unique characteristics of the vehicular 802.11 PHY layer for purposes of task offloading [SLT21, ZPX18, ZTW19], no current works have considered the unique characteristics of both 802.11 MAC and PHY layer for these purposes. The unique aspects of the PHY layer are vehicular mobility, which makes the channel highly variable, and propagation characteristics, which influence the path loss [VBT15]. The MAC layer characteristics are specified by the 802.11 standard in use, however common features are operation in unlicensed bands and distributed contention-based channel access. Modelling the impact of DCF on the throughput experienced by the vehicles has not been adequately addressed in prior works for purposes of cooperative computing. In most research, the impact of contention on achievable throughput is not considered at all [BBG21, KMH20]. On the other hand, in some papers contention is modeled as a constant reduction in achievable data rates and defined by the number of nearby vehicles and their transmission probabilities [ZLG19, WLZ22, ZTW19]. However, since multiple vehicles participate in cooperative computing, the number of active vehicles and their transmission probabilities depends on how task offloading is scheduled. Therefore, the impact of channel contention cannot be treated as independent of task offloading decisions. We incorporate accurate models of the DCF into our approach and consider how contention between task traffic flows will impact the maximum number of tasks that can be offloaded in the VMC.

Second, majority of current vehicular distributed computing papers adopt a task model that assumes a fixed and predetermined number of tasks that are then allocated to vehicles/servers for processing [CJL15, SLT21, FLW17]. However, for sensing-based applications such as cooperative perception, we assume that their QoS can be improved by additional compute resources available through cooperative computing. For example, experimental implementation studies of cooperative perception systems report that a major bottleneck in the frame rate at which cooperative perception systems can operate is the vehicular computational power [QAB18]. A similar consideration has been made in papers on task offloading for visual-based driving assistance systems [ZPX18, ZCM19], where task arrival rate was predetermined, but the complexity of each task adapts to the amount of computing resources available. The main difference between our paper and [ZPX18, ZCM19] is that we also aim to address the unique characteristics of 802.11 MAC and PHY with regards to cooperative computing.

4.3 System model

In this section, we describe the models of cooperative computing and communications in a VMC. The table of important parameters and variables introduce in this section is included in Table 4.1.

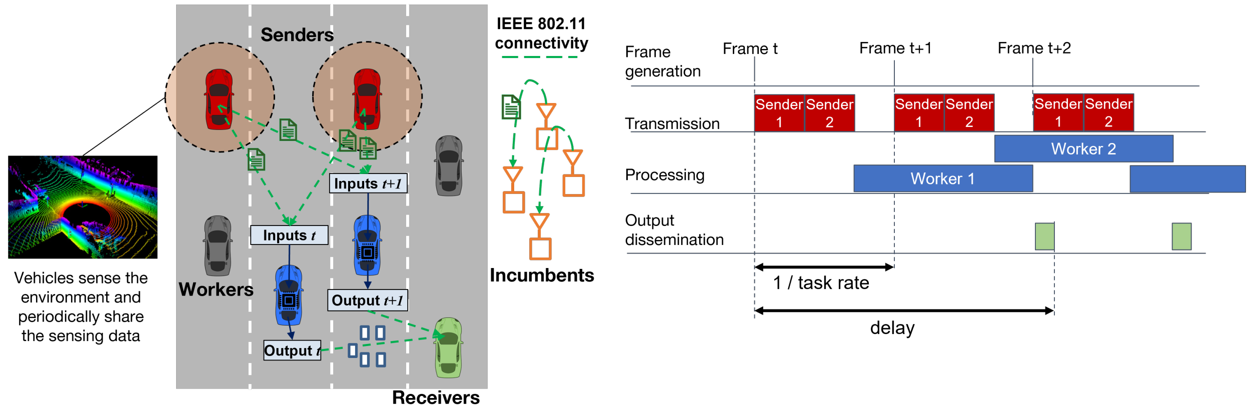


Figure 4.1: Cooperative computing in a vehicular micro cloud for sensing-based vehicular application.

4.3.1 Vehicular micro cloud model

The cars in the micro cloud with sensing capabilities that collect sensor data which is used as the input for computing tasks are called senders. The set of all sender cars in the VMC is denoted by $\mathcal{S}$. We define a set of worker cars $\mathcal{W}$ that can process cooperative computing tasks. The cars that take on the role of workers have computing resources available that can be used for processing of tasks. The cars in the micro cloud that are interested in and receive the outputs of the computed tasks are called the receivers and denoted by $\mathcal{R}$. The sets $\mathcal{S}$, $\mathcal{W}$ and $\mathcal{R}$ can be overlapping, i.e., the cars may have multiple roles from amongst the sender, receiver and worker role at the same time. The roles are illustrated in Fig. 4.1. Furthermore, senders, workers and receivers, together with other V2V capable cars form a stationary VMC. A stationary VMC is formed by cars that are present in particular fixed geographical area [HSH17]. The cars can enter and exit the area at any time and therefore the set of vehicles in VMC changes over time. One vehicle is elected to be a leader whose main functions are to coordinate other vehicles in running an application such as cooperative perception, driving assistance systems, storage and retrieval of data from the cloud.

4.3.2 Task streams model

Each computing task is part of a stream of tasks indexed by k that are generated at one or more senders. A particular vehicular application can consist of one or more task streams. The senders for each task stream are denoted by $s \in \mathcal{S}_k$. Each task stream has Q_k possible QoS levels: $1, \ldots, Q_k$. Without the loss of generality, let us assume that $Q_k = Q \ \forall k$. We use the binary variable $q_{k,l} \ \forall k, l$ to denote the QoS level, where $q_{k,l} = 1$ if the task stream is served at QoS level l and 0 otherwise. Since a task stream can only be served at one particular level, $\sum_l q_{k,l} = 1 \ \forall k$. The variables $q_{k,l} \ \forall k, l$ will be utilized in the problem formulation in Sec. 4.4. The task rate at quality of service l is $r_{k,l}$, task input data size is $b_{k,l}$ and task computing load is $c_{k,l}$. Parameters $r_{k,l}$, $b_{k,l}$ and $c_{k,l}$ are non-decreasing with respect to l. Each task has a maximum delay tolerance τ_k. The delay is measured from the instance when a task is generated until the task is processed and its outputs are delivered to the receivers. $\mathcal{R}_k$ is the set of receiver cars that are interested in receiving the output of the processed tasks of task stream k. To simplify our analysis, we assume that the output data size is negligible compared to the input data size and ignore its overhead.

4.3.3 Task stream model applied to cooperative perception

We explain how our task streams would apply to cooperative perception as the example application, illustrated in Fig. 4.1. Cooperative vehicular perception systems seek to expand a vehicle's field of view by enabling cars to share the data from their environment perception sensors via wireless communication and thereby allowing vehicles to achieve a global view of the traffic environment [COV22]. We focus on cooperative perception systems based on raw sensor data sharing [CTY19], often referred to as early fusion cooperative perception. Experimental implementation studies of cooperative perception systems report that a major bottleneck in the frame rate at which cooperative perception systems can operate is the vehicular computational power [QAB18]. In traditional cooperative perception systems, computation is done independently at each vehicle. Using cooperative computing, computing workload can be split amongst multiple vehicle workers, who process sensor frames and then deliver the outputs of their processing to other interested vehicles, $\mathcal{R}_k$. Computing workload is split between workers such that each worker will process a portion of the tasks in the task stream. Cooperative computing can then enable applications such as cooperative perception to operate at a higher QoS, which translates to higher $r_{k,l}$, $c_{k,l}$ or $b_{k,l}$.

Cooperative perception tasks will have multiple sender vehicles $s \in \mathcal{S}_k$ of input data. Depending on the type of sensor data, there are different types of computing tasks that might need to be performed. For example, on 3D data, generated by radar or LIDAR, processing would include object detection and localization. The data size of the sensor frames would correspond to $b_{k,l}$, computational processing of the sensor frames would have a load $c_{k,l}$ and frame rate would be equivalent to $r_{k,l}$. The set $\mathcal{S}_k$ contains cars that have sensing capabilities such as LIDAR or radar. The outputs of sensor data processing are normally locations and bounding boxes of the objects in the environment. Tasks have a delay tolerance τ_k, past which the information contained in the sensor data is out of date with the environment.

4.3.4 Computing model

The total amount of shareable compute power per worker is f_w CPU cycles per second. Each worker w will have a separate queue per each task stream k it processes, which will be allocated $x_{k,w,l} \in [0,1]$ share of the worker's computing power if the QoS level of k is l. Since a task stream k can only operate at one QoS level l, a particular worker w cannot allocate resources to multiple QoS levels l for some k, i.e $x_{k,w,l_1} > 0 \wedge x_{k,w,l_2} > 0 \implies l_1 = l_2$. For simplicity, we assume that cooperative computing operations will not be memory constrained on the vehicles, although that extension would be feasible in our resource assignment and scheduling framework. The per-task and per-worker computing delay at QoS l is $t^c_{k,w,l} = \frac{c_{k,l}}{f_w x_{k,w,l}}$. Each worker will process $z_{k,w,l}$ tasks per second of the task stream, which is upper bounded by the allocated resources $z_{k,w,l} \leq \lfloor \frac{f_w x_{k,w,l}}{c_{k,l}} \rfloor$. The sum rate across all workers must meet the task stream processing rate: $\sum_w z_{k,w,l} = q_{k,l} r_{k,l}$. This joint processing of the task stream is what enables a higher QoS compared to non-cooperative computing.

4.3.5 Communications model

We assume that all senders and workers are capable of V2V communication and are using the hardware from an IEEE 802.11 standard. The IEEE 802.11p standard is the 802.11 standard developed specifically for vehicular communications [MDW12], but modern vehicles are frequently equipped with Wi-Fi hardware which could also be used for inter-vehicle communications. Since the vehicles will be networking in ad-hoc mode without a central access point, channel contention is achieved using the 802.11 DCF. Specifically, we assume that the Enhanced Distributed Channel Acccess (EDCA) mechanism for contention is used, which was introduced in 802.11e standard [KRS17] and has been present in all subsequent 802.11 standards. The EDCA is based on the DCF and supports multi-priority traffic.

4.3.5.1 Overview of the EDCA Mechanism

The EDCA mechanism defines different access categories (ACs), e.g. Voice-VO, Video-VI, Best effort-BE, and Background-BK. Each AC has a queue independently contending for transmission with its own parameters. Let T_{AIFS} denote the duration of the AIFS, i.e., the idle period after a busy period. In the EDCA mechanism, if a channel is sensed idle when a packet arrives at an AC queue in a station and keeps idle for T_{AIFS} , the packet will be transmitted. Otherwise, if the channel is sensed busy, the station will first start up a backoff counter with the initial value set to one randomly selected from $[0, CW]$. During the backoff period, if the channel is sensed busy, the backoff counter will be frozen at the current value. After the channel becomes idle and keeps idle for T_{AIFS} , the counter will be decremented by 1. When the backoff counter becomes zero, the packet will be transmitted.

If the station does not receive an acknowledgment (ACK) packet in a given time, the packet will be retransmitted. At each retransmission, the value of CW will first be doubled, and then a new backoff procedure is initiated. The CW can be increased up to a maximum value $CW^{\max}$.

4.3.5.2 Task transmission delay model

In order for a task from stream k to be computed at worker w, task input data must be delivered from the all senders $\mathcal{S}_k$. This will be coordinated by the leader vehicle. Input data at QoS l provided by each sender will have a bit size $b_{k,l}$. We assume that transmission frame size is equal across all vehicles and equal to D_{tx} bits and the minimum back-off window per sender is W_s. Each transmitted frame has an overhead of D_h bits as part of the frame. If the input data size $b_{k,l}$ is greater than the payload of a single frame $D_{tx} - D_h$, then input data will be split across $\hat{b}_{k,l} = \lceil \frac{b_{k,l}}{D_{tx}-D_h} \rceil$ frames. The links between senders and workers have an achievable data rate $B_{s,w}$ in bits per second. The maximum bandwidth is the data rate at which a sender can reliably transmit to the worker once it gains access to the channel and is a function of the SNR between them. Therefore, transmission duration for a single task is $\hat{b}_{k,l} L_{s,w}$, where $L_{s,w} = \frac{D_{tx}}{B_{s,w}}$. Task computation at worker w starts when input data from all senders $\mathcal{S}_k$ is delivered, therefore the total expected transmission delay per task is $t^{tx}_{k,w,l} = \sum_{s \in \mathcal{S}_k} \hat{b}_{k,l} L_{s,w}$.

4.3.5.3 Channel capacity

The amount of tasks that can be offloaded is limited due to capacity limits of the channel. Channel capacity can be further limited by incumbent transmissions occurring on the same channel that can be represented as a set of active flows $\mathcal{I}$. Incumbent transmissions can occur due to other cars in the micro cloud transmitting data unrelated to cooperative computing or due to roadside devices utilizing the channel. We currently focus only on interference from other devices with 802.11 hardware. Such incumbent interference can be expected in both 802.11p and WiFi based vehicular networking. Each incumbent flow $i \in \mathcal{I}$ has a frame transmission rate $\tilde{z}_i$, an average per frame transmission duration $\tilde{L}_i$ and a minimum backoff window $\tilde{W}_i$. Each sender will receive a portion of the channel time, which we refer to as the service time $Y_s \in [0,1]$. Y_s depends on the sender's own activity and activity of all other nodes on the channel. Any node in 802.11 networks may also have multiple traffic flows in different AC queues, which can be treated as their own independent flows. If the sender throughput demand $\sum_l \sum_w z_{k,w,l}\hat{b}_{k,l}L_{s,w}$ exceeds Y_s, congestion occurs at the sender and task transmission delay will drastically increase due to a large queuing delay. Therefore, for successful cooperative computing, it is necessary to avoid congestion at all of the senders by ensuring $Y_s \geq \sum_l \sum_w z_{k,w,l}\hat{b}_{k,l}L_{s,w}$ $\forall k, \forall s \in \mathcal{S}_k$. Prediction of Y_s is challenging due to complex interactions between senders' flows and other transmissions on the channel. We will consider two different approaches for congestion control as part of our resource assignment and scheduling algorithm.

4.3.5.4 Achievable data rate model

The achievable data rate $B_{s,w}$ between sender s and worker w is primarily dependent to signal-to-noise ratio $\text{SNR}_{s,w}$ between them. Given a particular $\text{SNR}_{s,w}$ between a sender and a worker, the senders will apply a modulation and coding scheme (MCS) such that reliable transmission can take place. The MCS will determine the achievable data rate $B_{s,w}$ in bps. The achievable data rate $B_{s,w}$ between sender s and worker w is related to signal-to-noise ratio $\text{SNR}_{s,w}$ between them. Given a particular $\text{SNR}_{s,w}$, the senders will apply a MCS such that reliable transmission can take place. The MCS will determine the achievable data rate $B_{s,w}$ in bps. For example, in the 802.11p protocol, there are 8 possible MCSs with the following respective data rates: 3 Mbps, 4.5 Mbps, 6 Mbps, 9 Mbps, 12 Mbps, 18 Mbps, 24 Mbps and 27 Mbps. We denote the set of data rates by $\mathcal{B}$, which will depend on the utilized IEEE 802.11 standard. Newer IEEE 802.11 standards support multiple input multi output (MIMO) communications, for which the transmission rate is not selected simply based on the SNR, but also based on other channel-quality indicators. For simplicity, however, we will focus on standards where reliable data rate prediction is achievable based on SNR only.

Let the distance between a sender s and a worker w be $d_{s,w}$. Furthermore, we assume that the height of the antenna placement at a vehicle i is known and we denote it by h_i. We assume line-of-sight conditions and use the two-ray interference model to predict the large-scale path-loss characteristics. This model was experimentally validated for line-of-sight vehicular communications [SD11]. The expression for path loss in log-scale is given by:

$$\begin{gathered}P_l(d_{s,w}) = 20\log\left(4\pi\frac{d}{\lambda}\left|1+\Gamma_{\perp}e^{i\varphi}\right|^{-1}\right),\ \text{substituting}\\ \varphi = 2\pi\frac{d_{los}-d_{ref}}{\lambda}, \Gamma_{\perp} = \frac{\sin\theta_i - \sqrt{\epsilon_r - \cos^2\theta_i}}{\sin\theta_i + \sqrt{\epsilon_r - \cos^2\theta_i}}\\ d_{los} = \sqrt{d_{s,w}^2 + (h_t - h_r)^2}, d_{ref} = \sqrt{d_{s,w}^2 + (h_t + h_r)^2}\\ \sin\theta_i = (h_t + h_r)/d_{ref}, \cos\theta_i = d_{s,w}/d_{ref}.\end{gathered} \tag{4.1}$$

The constant λ is the wavelength corresponding to the channel center frequency. For more accurate path loss modelling, geometry-based modelling approaches could be applied such as ray-tracing models or simplified ray-tracing models [SEG11]. Small scale fading is normally captured as additive noise to large-scale fading $P_l(d_{s,w})$, with some well-known distribution such as Gaussian, Weibull or Nakagami [VBT15]. However, for simplicity, we exclude such modelling from our system model and the approach. Let P_t, G_t, G_r and N_0 be transmitted power, transmitter gain, receiver gain and noise power in dB, then the SNR between sender s and worker w is $\text{SNR}_{s,w} = P_t + G_t + G_r - P_l(d_{s,w}) - N_0$. Proposed methods for prediction of $B_{s,w}$ and $\text{SNR}_{s,w}$ in VMCs will be discussed in Sec. 4.7.

In the next section, we formulate the resource assignment and scheduling problem for cooperative computing of task streams in sensing-based vehicular applications.

Table 4.1: Important notation used in the paper and their meanings.

Variables/Parameters	
Vehicular micro cloud	
w	Worker index
$\mathcal{W}$	Set of workers
f_w	Computing power at worker w
s	Sender index
$d_{s,w}$	Distance between s and w
Task streams	
k	Task stream index
l	QoS level
$\mathcal{S}_k$	Set of senders in task stream k
$q_{k,l}$	QoS assignment variables
$r_{k,l}$	Task stream rate at QoS level l
$c_{k,l}$	Compute load at QoS level l
$b_{k,l}$	Input data size at QoS level l
τ_k	Task delay tolerance l
Computing and communications	
$x_{k,w,l}$	Compute resources assigned by w to process k at QoS l
$z_{k,w,l}$	Processing rate at w of tasks k at QoS l (tasks per sec.)
C_{IDLE}	Average idle time on the channel
$B_{s,w}$	Achievable data rate between s and w (bps)
$L_{s,w}$	Transmission duration of a single frame between s and w
$\hat{b}_{k,l}$	Number of frames per task of task stream k at QoS l
$t^{tx}_{k,w,l}$	Transmission delay of a single task at w of tasks k at QoS l
Incumbent flows	
i	Incumbent flow index
$\tilde{L}_i$	Frame duration
$\tilde{z}_i$	Frame rate

4.4 Resource assignment and scheduling for cooperative computing

In our framework, resource assignment and scheduling is executed periodically at a T_{assign} time interval to meet the changes in computing demands and computing resources in the micro-cloud. Let us assume that T_{assign} is a fixed parameter at this point. We will discuss extensions to adaptively select T_{assign} in Sec. 4.8.

Resource assignment and scheduling is performed at the leader vehicle that holds the information about computing demands and computing resources in the VMC. Our main objective is to maximize the sum QoS value across all task streams in the VMC $\sum_{k,l} lq_{k,l}$, where l is the QoS level. The main optimization variables are computing resource assignment

$x_{k,w,l}\ \forall k, w, l$, scheduled QoS $q_{k,l}\ \forall k, l$ and processing rate $z_{k,w,l}\ \forall k, w, l$.

Given certain scheduled QoS $q_{k,l}\ \forall k, l$ and $z_{k,w,l}\ \forall k, w, l$, queue congestion may occur at one or more senders. The set of viable $q_{k,l}\ \forall k, l$ and $z_{k,w,l}\ \forall k, w, l$ can be represented by a set of constraints:

$$g_i(z_{1,1,1}, \ldots, z_{K,W,L}, q_{1,1}, \ldots, q_{K,L}) \leq 0\ \forall i \in \{1, \ldots, G\} \tag{C1.1}$$

Where functions $g_i(\cdot)$ are some abstract, potentially non-convex functions of $q_{k,l}\ \forall k, l$ and $z_{k,w,l}\ \forall k, w, l$.

We formulate the resource assignment and scheduling problem as a mixed-integer programming (MIP) problem:

$$\max_{x_{k,w,l}, z_{k,w,l}, q_{k,l}} \sum_{k,l} l q_{k,l} \tag{P1}$$

s.t. (C1.1)

$$\frac{c_{k,l}}{f_w x_{k,w,l}} + t^{tx}_{k,w,l} \leq \tau_k \text{ if } x_{k,w,l} > 0,\ \forall k, w, l \tag{C1.2}$$

$$\sum_w z_{k,w,l} = q_{k,l} r_{k,l}\ \forall k, l \tag{C1.3}$$

$$z_{k,w,l} \leq \frac{f_w x_{k,w,l}}{c_{k,l}}\ \forall k, w, l \tag{C1.4}$$

$$x_{k,w,l} \geq 0\ \forall k, w, l \tag{C1.5}$$

$$\sum_l \sum_k x_{k,w,l} \leq 1\ \forall w \tag{C1.6}$$

$$\sum_l q_{k,l} \leq 1\ \forall k \tag{C1.7}$$

$$q_{k,l} \in \{0, 1\}\ \forall k, l \tag{C1.8}$$

$$z_{k,w,l} \in \{0, 1, 2, \ldots, r_{k,l}\}\ \forall k, w, l \tag{C1.9}$$

The achievable $\sum_{k,l} l q_{k,l}$ is limited by the computation and communication constraints. Constraints (C1.3) guarantee that the total task computing rate across workers meets the QoS rate. Constraints (C1.2) ensure that the sum of the computing delay and the task transmission delay $t^{tx}_{k,w,l}$ is smaller than the maximum task processing delay τ_k. Constraints (C1.4) limit the task computing rate per vehicle to be upper bounded based on the allocated compute resources. Constraints (C1.6) are the capacity constraint on computing resources per vehicle. Constraints (C1.5), (C1.7), (C1.8) and (C1.9) impose feasible values on the optimization variables. Finally, the set of constraints (C1.1) ensures that the sender queues will not become congested when offloading tasks.

There are several challenges in solving the problem (P1). The first challenge lies in developing methods for prediction of congestion and formulating the predicted set of viable $q_{k,l}$ $\forall k, l$ and $z_{k,w,l}$ $\forall k, w, l$ as a set of constraints that will replace (C1.1). The second is prediction of the delay $t^{tx}_{k,w,l}$, which relies on knowing the achievable data rates $B_{s,w}$ between senders and workers. Finally, given that we can predict $B_{s,w}$ $\forall s, w$ and we have developed methods for congestion prevention, we must develop methods to solve the problem (P1) optimally and efficiently or seek approximate solutions that can quickly obtain a suboptimal yet feasible solutions.

Resource assignment and scheduling determines how many tasks will be processed by each worker over the next T_{assign} seconds, which is given by $T_{\text{assign}} z_{k,w,l}$ $\forall k, w, l$. However, this still does not determine which specific tasks in the task stream will be processed by which specific workers. We refer to this as the task assignment. The specific assignment matters since inefficient assignment may cause a queuing delay at one or more workers. The assignment of tasks to workers is performed using a round-robin heuristic policy, which we found to minimize queuing at the workers. Since this assignment is not the main focus of this paper, we will postpone the description of the policy until Sec. 4.9.

In sections 4.5 and 4.6, we introduce two congestion control approaches and formulate them as two different sets of constraints that will replace (C1.1) in (P1). Based on the congestion control approaches used, we will obtain two different resource assignment and scheduling approaches.

4.5 resource assignment and scheduling with bottleneck congestion control

In this section, we present a method for congestion control that can be implemented as a single linear integer constraint that replaces (C1.1) in (P1). We then formulate the resulting optimization problem as an MILP problem and solve it using branch and bound methods to obtain a solution for resource assignment and scheduling in Sec. 4.5.2. The complexity of branch and bound approaches scales exponentially with the number vehicles participating in cooperative computing and the number of task streams, so they may not be feasible to run in real time when the number of vehicles participating in cooperative computing is high. For these scenarios, we develop an approximate resource assignment methods that runs in polynomial time. in Sec. 4.5.3.

4.5.1 Congestion control constraints

In the absence of transmissions due to cooperative computing, we assume that the channel may still be utilized by incumbent devices. Specifically, the channel will be occupied by incumbent transmissions for a portion of the time that we refer to use as the busy time and idle for a portion of the time that we refer to as the idle time, which is denoted by $C_{\text{IDLE}} \in [0, 1]$. We make a simplifying assumption that due to the size of the VMC all vehicles are in the interference range of the incumbent devices, therefore the idle time observed by all the vehicles in the VMC is the same. We propose the following congestion control approach. If the total load on the channel due to offloaded tasks exceeds the idle time C_{IDLE}, then this model predicts that at least one of the senders will be congested. Congestion can then be avoided by meeting the following constraint:

$$\sum_{k}\sum_{l}\sum_{s\in\mathcal{S}_k}\sum_{w\neq s} z_{k,w,l}\hat{b}_{k,l}L_{s,w} \leq C_{\text{IDLE}} \tag{C2.1}$$

For this approach, it is necessary to predict the achievable data rates between vehicles $B_{s,w}$ in order to predict $L_{s,w}$. Moreover, it is necessary to estimate C_{IDLE}. To obtain C_{IDLE}, idle time is independently estimated by each vehicle in the micro cloud. Then, the average of idle times observed at each vehicle is used as the estimate for C_{IDLE}.

Since this congestion control approach can be represented using a single linear constraint, it is easy to implement into resource assignment and scheduling, as we will demonstrate in the following section. However, it may not be always be accurate for prediction of congestion. This is because it does not capture the effects of contention for the channel between computational offloading flows and incumbent flows, but instead treats them as independent from each other. Furthermore, as more computational offloading flows are added, the channel contention between them reduces the efficiency of the DCF.

4.5.2 Exact solution methods

First, we derive an approach based on MILP that optimally solves the resulting resource assignment and scheduling problem. In this congestion control approach, the constraint (C2.1) replaces the set of constraints (C1.1). The constraint is linear with integer optimization variables and this is the case for most other constraints in the resulting problem. The only exception are the constraints (C1.2), which are non-convex. These can be reformulated as linear integer constraints. We can replace the constraints (C1.2) by the following constraints.

$$\frac{c_{k,l}}{f_w} + x_{k,w,l}t^{tx}_{k,w,l} \leq x_{k,w,l}\tau_k + (1 - x^b_{k,w,l})\frac{c_{k,l}}{f_w} \ \forall k, w, l \tag{C2.2}$$

$$x_{k,w,l} \leq x^b_{k,w,l} \ \forall k, w, l \tag{C2.3}$$

$$x^b_{k,w,l} \in \{0,1\}\ \forall k, w, l \tag{C2.4}$$

In the above constraints, we created a set of auxiliary integer optimization variables $x^b_{k,w,l}\ \forall k, w, l$. The optimization problem we arrive at then is:

$$\begin{aligned} \max_{x_{k,w,l}, z_{k,w,l}, q_{k,l}, x^b_{k,l}} \quad & \sum_{k,l} l q_{k,l} \qquad \text{(P2)} \\ \text{s.t.} \quad & \text{(C1.3) to (C1.9), (C2.1) to (C2.4)} \end{aligned}$$

This is now an MILP problem, which can be solved using solvers that are based on branch-and-bound methods [Wol20]. While such approaches will provide us with an optimal solution, they may not always be efficient enough to execute in real-time because their complexity is exponential in the worst case. The number of variables and constraints scales with the number of workers and task streams. When the number of workers and task streams is large, we can expect that solving the problem using exact methods may be infeasible. Therefore, for such cases, we will seek approximate methods and use the exact solution methods to quantify the upper bounds on the gains of cooperative computing.

4.5.3 Approximate solution methods

Even though the problem (P2) is an MILP problem, some MILP problems can still be optimally solved in polynomial time. In this section, we will show that the problem (P2) is NP-hard, which tells us that the optimal solution can indeed only be found in exponential time and motivates us to look for approximate methods.

Theorem 1. *The problem (P2) to find the optimal computing resource assignment and scheduling such that sum QoS is maximized is NP-hard.*

Proof. The key idea of the proof is to find a representative NP-hard problem that can be reduced to a special case of our problem. We will consider the following special case of (P2).

First, we assume that the number of levels across all task streams is $Q_k = 1$. In that case, we can drop the subscript l from all the parameters and variables that have it in (P2). Next, let the rate across all task streams be $r_k = 1$. Furthermore, we introduce a parameter $\alpha_{k,w} = \frac{c_k}{f_w\left(\tau_k - t^{tx}_{k,w}\right)}$, which is equal to the minimum amount of computing resources $x_{k,w}$ that need to be allocated such that the delay constraints (C2.1) to (C2.4) are met. Under the assumption that $r_k = 1$, when a worker is assigned to task stream k, the minimum amount of computing resources $x_{k,w}$ that have to be invested to satisfy the constraint (C1.4) are $\beta_{k,w} = \frac{c_k}{f_w}$. One can select the input parameters of (P2) such that $\beta_{k,w} \geq \alpha_{k,w}\ \forall k, w$. In this special case, the delay constraints can be ignored, because we know they will not be the binding constraints. Since delay constraints are ignored, we can exclude the variables $x_{k,w} \forall k, w\ x^b_{k,w} \forall k, w$ from optimization and model computing capacity constraints as $\sum_k z_{k,w} \frac{c_k}{f_w} \leq 1\ \forall w$. The special case of (P2) that we described is summarized below:

$$\max_{z_{k,w}, q_k} \quad \sum_k l q_k \tag{SP2}$$

$$\sum_w z_{k,w} = q_{k,l}\ \forall k$$

$$\sum_k \sum_{s \in \mathcal{S}_k} \sum_{w \neq s} z_{k,w} \hat{b}_k L_{s,w} \leq C_{\text{IDLE}} \tag{SP2.1}$$

$$\sum_k z_{k,w} \frac{c_k}{f_w} \leq 1\ \forall w \tag{SP2.2}$$

$$q_k \in \{0, 1\}\ \ \forall k$$

$$z_{k,w} \in \{0, 1\}\ \ \forall k, w$$

The problem (SP2) is an example of multi-resource bottleneck generalized assignment problem (GAP) which is known to be NP-hard [KA12]. In multi-resource bottleneck GAP there is a set of tasks (task streams in (SP2)), set of agents (workers in (SP2)), and a set of resources ($\frac{c_k}{f_w}$ in (SP2)) used by the agents to perform these tasks. Each task must be assigned to one agent. The total amount of resources used by an agent cannot exceed the amount of resource available to it (constraint SP2.2). Furthermore, total weighted load across all agents cannot exceed a certain threshold (constraint SP2.1).

Since (SP2) is the special case of the problem (P2) and we demonstrated that it is NP-hard, then the problem (P2) is also NP-hard. □

Next, we describe an approximate solution method for solving (P2). The proposed approximate approach is based on an altered continuous relaxation of (P2):

$$\max_{x_{k,w,l}, z_{k,w,l}, q_{k,l}} \quad \sum_{k,l} l q_{k,l} \tag{P3}$$

$$\text{s.t.} \quad \text{(C1.3) to (C1.7), (C2.1)}$$

$$z_{k,w,l} \geq 0 \ \forall k, w, l \tag{C3.1}$$

$$q_{k,l} \geq q^{lb}_{k,l} \ \forall k, l \tag{C3.2}$$

$$q_{k,l} \leq q^{ub}_{k,l} \ \forall k, l \tag{C3.3}$$

$$x_{k,w,l} \geq x^{lb}_{k,w,l} \ \forall k, w, l \tag{C3.4}$$

$$x_{k,w,l} \leq x^{ub}_{k,w,l} \ \forall k, w, l \tag{C3.5}$$

The problem (P3) is a linear programming (LP) problem but it is not a direct continuous relaxation of (P2), which would be obtained by simply replacing the constraints (C1.8) and (C1.9) by constraint (C3.1) and (C3.2). Initially, $q^{lb}_{k,l} = 0 \ \forall k, l$, $x^{lb}_{k,w,l} = 0 \ \forall k, w, l$, $x^{ub}_{k,w,l} = 1 \ \forall k, w, l$ and $q^{ub}_{k,l} = 1 \ \forall k, l$. In (P3) we also remove the delay constraints (C2.3) and (C2.4). Furthermore, we added constraints (C3.4) and (C3.5) which will be utilized to ensure that the delay constraints are not violated. The proposed approximate method will repeatedly solve the problem (P3) and use the obtained continuous solutions for $x_{k,w,l}$, $z_{k,w,l}$, $q_{k,l}$ to solve for resource assignment and scheduling.

The proposed approximate solution algorithm is summarized in Algorithm 1. In line 1, we obtain an initial continuous solution. Then, in the main loop we perform greedy exploration over $q_{k,l} \ \forall k, l$ and solve for $x_{k,w,l} \ \forall k, w, l$ and $z_{k,w,l} \ \forall k, w, l$. At the start of each loop we select

Algorithm 1: Approximate solution algorithm for (P2)

```
    Input: $l, f_w, \alpha_{k,w,l}, c_{k,l}, r_{k,l}, C_{\text{IDLE}}\ \forall k, w, l$
    Output: $x_{k,w,l},\ z_{k,w,l},\ q_{k,l}\ \forall k, w, l$
 1  $x^{lb}_{k,w,l} \leftarrow 0\ \forall k, w, l;\ x^{ub}_{k,w,l} \leftarrow 1\ \forall k, w, l$
 2  $q^{ub}_{k,l} \leftarrow 1\ \forall k, l;\ q^{lb}_{k,l} \leftarrow 0\ \forall k, l$
 3  Solve (P3) to obtain $x_{k,w,l},\ z_{k,w,l},\ q_{k,l}\ \forall k, w, l$
 4  while $\sum_k \sum_l q^{lb}_{k,l} \le K$ do
 5      $\hat{k}, \hat{l} = \text{argmax}_{k,l}\, l$ s.t. $q^{ub}_{k,l} = 1$ and $q^{lb}_{k,l} = 0$
 6      $q^{lb}_{\hat{k},\hat{l}} \leftarrow 1;\ q^{ub}_{\hat{k},l} \leftarrow 0\ \forall l \neq \hat{l}$
 7      Select $\mathcal{V} \subset \mathcal{W}$ with $V$ highest $x_{\hat{k},w,\hat{l}}$ and such that $\left(1 - \sum_{k\neq\hat{k}} \sum_l x_{k,w,l}\right) \ge \alpha_{\hat{k},w,\hat{l}}$
 8      $x^{lb}_{\hat{k},w,\hat{l}} \leftarrow \alpha_{\hat{k},w,\hat{l}}\ \forall w \in \mathcal{V}$
 9      $x^{ub}_{\hat{k},w,\hat{l}} \leftarrow 0\ \forall w \in \mathcal{W} \setminus \mathcal{V}$
10      if (P3) is feasible then
11          Solve (P3) to update $x_{k,w,l},\ z_{k,w,l},\ q_{k,l}\ \forall k, w, l$
12      else
13          $q^{lb}_{\hat{k},\hat{l}} \leftarrow 0;\ q^{ub}_{\hat{k},\hat{l}} \leftarrow 0;\ q^{ub}_{\hat{k},l} \leftarrow 1\ \forall l \neq \hat{l}$
14          $x^{lb}_{\hat{k},w,\hat{l}} \leftarrow 0\ \forall w \in \mathcal{V}$
15      end
16  end
17  Perform randomized rounding on $z_{k,w,l}\ \forall k, w, l$ while meeting the constraints of (P3)
```

a pair $(\hat{k}, \hat{l})$ with highest l, under the constraints $q^{ub}_{k,l} = 1$ and $q^{lb}_{k,l} = 0$. The constraints limit selection to (k, l) pairs that have not been explored before and task streams whose QoS has not been determined yet. In line 6, the algorithm assigns the QoS for task stream $\hat{k}$ to be $\hat{l}$. In line 7, the algorithm selects V workers that will be assigned to task stream $\hat{k}$. However, workers are not necessarily fully assigned to task stream $\hat{k}$ and may be shared with other task streams. The worker assignment step is necessary to ensure that delay constraints will be met. When selecting workers, the algorithms chooses the ones that have at least $\alpha_{k,w,l}$ resources available. This is necessary to ensure that the worker will have sufficient computing resources available to meet the delay constraints (C2.2), (C2.3) and (C2.4). Among such workers, up to V workers with highest $x_{k,w,l}$ values are selected. Therefore, we use the current solution for $x_{k,w,l}\ \forall k, w, l$ to guide the assignment of workers. The parameter V was selected empirically to maximize the final $\sum_{k,l} lq_{k,l}$ across various tested scenarios. In line 8, for the workers that were assigned, the algorithm sets a lower bound on $x_{\hat{k},w,\hat{l}}$ to ensure delay constraints are met, while all others workers are excluded from processing the task stream $\hat{k}$ in line 9. In line 10, the algorithm checks if the problem (P3) can be solved with updated values for $q^{lb}_{k,l}, x^{lb}_{k,w,l}, x^{ub}_{k,w,l}, q^{ub}_{k,l}$ $\forall k, w, l$. If so, the resource assignment and scheduling solution is updated in line 11. Otherwise, in line 13, we reset the QoS assignment for task stream $\hat{k}$ and ensure that the pair $(\hat{k}, \hat{l})$ will not be explored again in the future loops. In line 14, we release the workers from processing $(\hat{k}, \hat{l})$ if a solution was not feasible. The loop proceeds until all task streams are assigned to some QoS level.

At the end of the loop, we will obtain a feasible integer assignment for $q_{k,l}$ $\forall k, l$ in (P3) and a feasible assignment for $x_{k,w,l}$ $\forall k, w, l$ in (P3). However, the solution $z_{k,w,l}$ $\forall k, w, l$ is not integer, even though it is feasible in (P3). Accordingly, the algorithm will perform randomized rounding on $z_{k,w,l}$ $\forall k, w, l$, while keeping $q_{k,l}$ $\forall k, l$ fixed. With probability $\min(\mu(z_{k,w,l} - \lfloor z_{k,w,l} \rfloor), 1)$, $z_{k,w,l}$ is rounded up, otherwise $z_{k,w,l}$ is rounded down. The parameter $\mu \geq 1$ was selected empirically to maximize the final $\sum_{k,l} lq_{k,l}$. If rounding $z_{k,w,l}$ up, it is possible that $x_{k,w,l}$ will need to be increased to meet the constraint (C1.4). However, rounding up of $z_{k,w,l}$ is performed only if the constraints (C1.6) and (C2.1) are not violated. After performing rounding on $z_{k,w,l}$ $\forall k, w, l$, the constraint (C1.3) may not be met for some task streams. For these task streams, $z_{k,w,l}$ may need to be lowered for some workers if the total task computing rate is too high ($\sum_w z_{k,w,l} > q_{k,l} r_{k,l}$). If the total task computing rate is too low for some task streams ($\sum_w z_{k,w,l} < q_{k,l} r_{k,l}$), then QoS must be lowered until $\sum_w z_{k,w,l} \geq q_{k,l} r_{k,l}$. Lowering the QoS will not violate any of the other constraints since we assume that if $l_1 < l_2$, then $c_{k,l_1} \leq c_{k,l_2}$, $b_{k,l_1} \leq b_{k,l_2}$ and $r_{k,l_1} \leq r_{k,l_2}$. Next, we discuss the characteristics of Algorithm 1. A solution $q_{k,l}$ $\forall k, l$, $x_{k,w,l}$ $\forall k, w, l$ and $z_{k,w,l}$ $\forall k, w, l$ obtained with Algorithm 1 is a feasible solution for (P2). This is because any feasible solution of (P3) with integer $q_{k,l}$ $\forall k, l$ and $z_{k,w,l}$ $\forall k, w, l$ and with appropriate $x^{ub}_{k,w,l}, x^{lb}_{k,w,l}$ $\forall k, w, l$ is feasible in (P2). $x^{ub}_{k,w,l}$ $\forall k, w, l$ and $x^{lb}_{k,w,l}$ $\forall k, w, l$ are appropriately assigned in Algorithm 1 to meet the delay constraints (C2.1) to (C2.4). Furthermore, Algorithm 1 has a polynomial time complexity because in the worst case it solves the LP problem (P3) $(KL+1)$ times. The Algorithm 1 is not guaranteed to obtain the optimal solution for (P2), however it tends to produce solutions that lead to high $\sum_{k,l} lq_{k,l}$ because of the greedy approach to assigning QoS (lines 5 and 6) and the proposed strategy of assigning workers to task streams (line 7). The performance of Algorithm 1 will be evaluated through extensive simulations.

4.6 Resource assignment and scheduling with per-flow congestion control

In this section, we present an alternative resource assignment and scheduling method based on a more accurate model of congestion, which addresses the drawbacks of the congestion model presented in Sec. 4.5. While the resource assignment and scheduling method we will present is more reliable in preventing congestion, it is computationally more complex and requires more information about incumbent interference parameters. We first develop an approach for resource assignment and scheduling based on MILP in Sec. 4.6.2 and then propose approximate methods in Sec. 4.6.3

4.6.1 Congestion control constraints

The proposed congestion control constraints are based on a throughput prediction approach for 802.11 networks based on EDCA that was introduced in [YK07]. The approach is based on modeling channel contention as a Markov process. The model assumes that all active nodes can be in two states: saturated and non-saturated. A saturated node always has backlogged packets while a non-saturated node often has an empty queue. We denote the set of all saturated nodes by $\mathcal{N}_1$ and the set of non-saturated nodes by $\mathcal{N}_2$. In order to explain the model more easily, we temporarily exclude the incumbent flows from the analysis. We introduce a set of variables $\hat{z}_s = \sum_l \sum_{w \neq s} z_{k,w,l} \hat{b}_{k,l}$ $\forall k, \forall s \in \mathcal{S}_k$ which denote the frame transmission rate per sender and a set of variables $\hat{L}_s\ \forall k, \forall s \in \mathcal{S}_k$ which denote the average frame transmission duration from the sender to all of the workers. $\hat{L}_s$ is defined as:

$$\hat{L}_s = \frac{\sum_l \sum_w z_{k,w,l} L_{s,w}}{\sum_l \sum_w z_{k,w,l}} \quad \forall k, \forall s \in \mathcal{S}_k. \tag{4.6}$$

Let $\hat{\delta}_s$ be a binary variable equal to 1 if $\hat{z}_s > 0$, and 0 otherwise. Then, following the approach in [YK07], we define a variable called congestion level:

$$\eta\left(\mathcal{N}_1, \mathcal{N}_2\right) = \frac{\sum_{s \in \mathcal{N}_1} \hat{\delta}_s \hat{L}_s / W_s}{1 - \sum_{s \in \mathcal{N}_2} \hat{z}_s \hat{L}_s / C_{\text{cap}}}. \tag{4.7}$$

where C_{cap} denotes the maximum total service time in a fully saturated network: $\sum_s Y_s \leq C_{\text{cap}}$. The value of C_{cap} is normally set to 0.9 and tends to be constant regardless of the number of flows and their characteristics. If a sender s is saturated, i.e. $s \in \mathcal{N}_1$, then s will obtain a service time $Y_s = \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1, \mathcal{N}_2) W_s}$. Otherwise, if a sender s is not saturated, it will receive a service time equal to its demand: $Y_s = \hat{z}_s \hat{L}_s$. In the latter case, the sender queues will not be congested which is the outcome that we want for successful cooperative computing. This prediction relies on knowing the sets $\mathcal{N}_1$ and $\mathcal{N}_2$. To solve for $\mathcal{N}_1$ and $\mathcal{N}_2$, we can rely on the fact that when a valid solution for $\mathcal{N}_1$ and $\mathcal{N}_2$ has been found $\hat{z}_s \hat{L}_s \geq \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1, \mathcal{N}_2) W_s}\ \forall s \in \mathcal{N}_1$ and $\hat{z}_s \hat{L}_s < \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1, \mathcal{N}_2) W_s}\ \forall s \in \mathcal{N}_2$. The following approach was proposed in [YK07] to solve for $\mathcal{N}_1$ and $\mathcal{N}_2$. First, the flows are sorted based on $\hat{z}_s W_s$ from highest to lowest. Initially, all senders are placed in $\mathcal{N}_2$. Then, starting from the highest $\hat{z}_s W_s$ sender, we sequentially add senders into $\mathcal{N}_1$, until $\hat{z}_s \hat{L}_s \geq \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1, \mathcal{N}_2) W_s}\ \forall s \in \mathcal{N}_1$ and $\hat{z}_s \hat{L}_s < \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1, \mathcal{N}_2) W_s}\ \forall s \in \mathcal{N}_2$. At that point, the solution for $\mathcal{N}_1$ and $\mathcal{N}_2$ is found. It is straightforward to add incumbent flows into the approach we just described, as they would be treated identically as the sender flows. We will denote the set of saturated incumbent flows by $\mathcal{N}_3$ and the unsaturated set by $\mathcal{N}_4$.

The desired outcome for cooperative computing is for no senders to be saturated, since this will prevent queue congestion. In other words, the set $\mathcal{N}_1$ should be empty. However, implementing the throughput prediction method from [YK07] as a set of constraints in (P1) is challenging because this method is based on flows with predetermined rates, whereas the purpose of (P1) is to solve for the rate values. Instead, we propose the following strategy to ensure that the set $\mathcal{N}_1$ will be empty. If none of the senders are saturated, then there is no valid solution for $\mathcal{N}_1$ and $\mathcal{N}_2$ such that there are senders in $\mathcal{N}_1$. To ensure this, we set $\hat{z}_s\hat{L}_s < \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_3,\mathcal{N}_4)W_s}$ $\exists s \in \mathcal{N}_1$, regardless of the set $\mathcal{N}_1$ being chosen. Let the set $\mathcal{N}$ be the set of all possible combinations for $(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_3,\mathcal{N}_4)$. Then, for each combination $(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_3,\mathcal{N}_4)$, we add the set of constraints $\hat{z}_s\hat{L}_s < \frac{\hat{L}_s C_{\text{cap}}}{\eta(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_1,\mathcal{N}_2)W_s}$ $\forall s \in \mathcal{N}_1$ into (P1), out of which only one must be met. After inserting the expression for $\eta\left(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_3,\mathcal{N}_4\right)$, this inequality is written as:

$$\frac{\hat{z}_s W_s}{C_{\text{cap}}} < \frac{1 - \sum_{j\in\mathcal{N}_2} \hat{z}_j \hat{L}_j / C_{\text{cap}} - \sum_{i\in\mathcal{N}_4} \tilde{z}_i \tilde{L}_i / C_{\text{cap}}}{\sum_{j\in\mathcal{N}_1} \hat{\delta}_j \hat{L}_j / W_j + \sum_{i\in\mathcal{N}_3} \tilde{L}_i / \tilde{W}_i} \tag{4.8}$$

The number of constraints of the type in Eq. 4.8 that needs to be added to (P1) are $O(2^{|\mathcal{S}|+I})$. Therefore, this approach is not suitable in circumstances with a large number of senders or incumbent flows because the computational complexity of solving (P1) may be too high for real-time operation. In the following sections, we will develop exact and approximate solution methods for solving for resource assignment and scheduling with these congestion constraints.

4.6.2 Exact solution methods

In the previous section, we showed that the set of constraints that must be added to (P1) is of the type given in Eq. 4.8. This equation contains the variable $\hat{L}_s$, which was defined in Eq. 4.6. However, the expression in Eq. 4.6 cannot be represented as a linear integer constraint, which is important for formulating this problem as an MILP. We wish to formulate the resource assignment and scheduling problem as an MILP because the solvers for these problems are well developed and on average have lower than exponential computational complexity. First, we will replace the variables $\hat{L}_s$ with static parameters $\bar{L}_s$, which are defined as follows:

$$\bar{L}_s = \frac{\sum_w \mathbf{1}_{(B_{s,w}>0)} L_{s,w}}{\sum_w \mathbf{1}_{(B_{s,w}>0)}} \quad \forall k, \forall s \in \mathcal{S}_k. \tag{4.9}$$

$\mathbf{1}_Q$ is a function that outputs 1 if the logical expression Q is true, and 0 otherwise. Hence, the average frame transmission duration at sender s, $\hat{L}_s$, is approximated as the average duration of a single frame transmission to all workers for which the expected achievable data rate from sender s is non-zero.

The expression in Eq. 4.8 also cannot be formulated as a linear integer constraint due to the quadratic summation term $\frac{\hat{z}_s W_s}{C_{\text{cap}}} \left(\sum_{j \in \mathcal{N}_1} \hat{\delta}_j \hat{L}_j / W_j \right)$. We will use the big-M method to convert the expression intro a linear integer constraint:

$$\begin{aligned} \frac{\hat{z}_s W_s}{C_{\text{cap}}} \left(\sum_{j \in \mathcal{N}_1} \bar{L}_j / W_j + \sum_{i \in \mathcal{N}_3} \tilde{L}_i / \tilde{W}_i \right) \leq 1 - \\ \sum_{j \in \mathcal{N}_2} \hat{z}_j \bar{L}_j / C_{\text{cap}} - \sum_{i \in \mathcal{N}_4} \tilde{z}_i \tilde{L}_i / C_{\text{cap}} + \sum_{j \in \mathcal{N}_1} (1 - \hat{\delta}_j) M \\ \forall \left(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4 \right) \in \mathcal{N}, \ \exists s \in \mathcal{N}_1 \end{aligned} \tag{C4.1}$$

$$\hat{z}_s = \sum_l \sum_{w \neq s} z_{k,w,l} \hat{b}_{k,l} \ \forall k, \forall s \in \mathcal{S}_k \tag{C4.2}$$

The constant M is chosen appropriately such that the constraint is guaranteed to become non-binding when any of the senders in $\mathcal{N}_1$ are not active, i.e. when $\hat{\delta}_j = 0$. We can now formulate resource assignment and scheduling as an MILP problem based on the advanced congestion constraints as:

$$\begin{aligned} \max_{x_{k,w,l}, z_{k,w,l}, q_{k,l}, x^b_{k,l}, z_s} \quad & \sum_{k,l} l q_{k,l} \\ \text{s.t.} \quad & (\text{C1.3}) \text{ to } (\text{C1.9}), (\text{C2.2}) \text{ to } (\text{C2.4}) \\ & (\text{C4.1}) \text{ and } (\text{C4.2}) \end{aligned} \tag{P4}$$

Similar to problem (P2), this problem can be solved using solvers based on branch-and-bound methods. While such approaches will provide us with an optimal solution, they may not always be efficient enough to execute in real-time because their complexity is exponential in the worst case. Therefore, for such cases, we will seek approximate methods and use the exact solution methods to quantify the upper bounds on the gains of cooperative computing.

4.6.3 Approximate solution methods

Since the problem (P4) is similar to (P2), we can expect that it is also NP-hard. For the sake of brevity, we omit the whole proof and only explain the basic idea behind proving this. Using the same strategy as in Theorem 1, it can be shown that GAP is reducible to a special case of this problem. The only difference between (P4) and (P2) are the congestion constraints (C4.1) and (C4.2). However, a special case can be selected that has the following properties: $\frac{\bar{L}_s}{C_{\text{cap}}} \ll 1 \; \forall s$ and $\frac{\tilde{L}_i}{C_{\text{cap}}} \ll 1 \; \forall i$, which guarantees that the constraints (C4.1) are not binding and can be omitted from the special case. Then, the proof would follow in a similar manner as the proof of Theorem 1. Since the problem (P4) is NP-hard, we are motivated to look for approximate solution methods.

The approximation approach that we will propose is based on utilizing the solution obtained for (P2) and lowering the QoS per task stream until we meet the congestion constraints described in Sec. 4.6.1. This method is based on the fact the problems (P2) and (P4) only differ in their congestion constraints. Furthermore, this approach is based on the assumption that if $l_1 < l_2$, then $c_{k,l_1} \leq c_{k,l_2}$, $b_{k,l_1} \leq b_{k,l_2}$ and $r_{k,l_1} \leq r_{k,l_2}$. Therefore, if QoS of a task stream is lowered starting from a feasible solution, then only constraints that can be potentially violated in (P2) are (C1.3).

The proposed way of accomplishing this is summarized in Algorithm 2. In line 1, the algorithm first obtains a solution for (P2) using the exact or approximate methods described in Sec. 4.5. We can expect that the exact solution of (P2) will be faster to obtain using MILP methods than the exact solution of (P4) if the number of senders $|\mathcal{S}|$ is high, since that determines the number of constraints in (C4.1). Otherwise, approximate methods for solving (P2) can be used as the starting point. Then, in the main loop, we first introduce temporary variables $\tilde{z}_{k,w,l}, \tilde{q}_{k,l}$ $\forall k, w, l$, which will be used to store a solution for each task stream such that QoS level is lowered by 1. Then, in the inner loop we obtain a solution for task processing rate $\tilde{z}_{k,w,l}$ $\forall k, w, l$ per worker that supports the lowered $\tilde{q}_{k,l}$ $\forall k, l$. The solution is obtained in line 7, where the algorithm determines the positive reduction in task processing rate per vehicle $\Delta_w \forall w$ that will also minimize the cumulative transmission duration on the channel $\sum_{s \in \mathcal{S}_{\hat{k}}} \sum_{w \neq s} (z_{\hat{k},w,l} - \Delta_w) \hat{b}_{\hat{k},\hat{l}-1} L_{s,w}$. This minimization problem in line 7 can be solved by minimizing task processing rate in the order of the workers with the highest $L_{s,w}$. Cumulative transmission duration is correlated with congestion, so minimizing it makes it more likely that the constraints (C4.1) and (C4.2) are met at the end of the loop. Then, after solving for $\tilde{z}_{k,w,l}$ $\forall k, w, l$, we lover the QoS by one level for the task stream $\tilde{k}$ (selected in line 11) such that cumulative transmission duration on the channel will be minimized. The main loop continues until the constraints (C4.1) and (C4.2) are met.

Since Algorithm 2 updates the solution of (P2) in a manner such that the constraints of (P2) are not violated, while meeting the constraints (C4.1) and (C4.2), the obtained solution is also feasible in (P4). The Algorithm 2 complexity is determined by the complexity of the solution method used to obtain a starting solution in line 1, which can be exponential if MILP is used and polynomial if Algorithm 1 is used. The solution obtained using Algorithm 1 is not optimal, but since the starting solution obtained in line 1 maximizes sum QoS, the output solution can be expected to have a high objective value. The gaps between Algorithm 2 and exact solutions methods developed in the previous section will be quantified through extensive simulations.

Algorithm 2: Approximate solution algorithm for (P4)

```
   Input: $l, f_w, \alpha_{k,w,l}, c_{k,l}, r_{k,l}, C_{\text{IDLE}}\ \forall k, w, l$
   Output: $x_{k,w,l},\ z_{k,w,l},\ q_{k,l}\ \forall k, w, l$
1  Solve (P2) to obtain $x_{k,w,l},\ z_{k,w,l},\ q_{k,l}\ \forall k, w, l$ ;
2  while constraints (C4.1) and (C4.2) are not met do
3      $\tilde{z}_{k,w,l} \leftarrow 0\ \forall k, w, l$;
4      $\tilde{q}_{k,l} \leftarrow 0\ \forall k, w, l$;
5      for $\hat{k} \in 1, \ldots, K$ do
6          $\hat{l} \leftarrow \text{argmax}_l\, q_{\hat{k},l}$;
7          $\Delta_1, \ldots, \Delta_W = \text{argmin}_{\Delta_1,\ldots,\Delta_W} \sum_{s\in\mathcal{S}_{\hat{k}}} \sum_{w\neq s}(z_{\hat{k},w,l} - \Delta_w)\hat{b}_{\hat{k},\hat{l}-1}L_{s,w}$ s.t.
             $\sum_w (z_{\hat{k},w,\hat{l}-1} - \Delta_w) = r_{\hat{k},\hat{l}-1}$;
8          $\tilde{z}_{\hat{k},w,\hat{l}-1} \leftarrow z_{\hat{k},w,\hat{l}} - \Delta_w\ \forall w$;
9          $\tilde{q}_{\hat{k},w,\hat{l}-1} \leftarrow 1$;
10     end
11     $\tilde{k} \leftarrow \text{argmax}_k \sum_{s\in\mathcal{S}_{\hat{k}}} \sum_l \sum_{w\neq s}(z_{\hat{k},w,l} - \tilde{z}_{\hat{k},w,l})\hat{b}_{\hat{k},l}L_{s,w}$;
12     $z_{\tilde{k},w,l} \leftarrow \tilde{z}_{\hat{k},w,l} \forall l, w$ ;
13     $q_{\tilde{k},l} \leftarrow \tilde{q}_{\hat{k},l} \forall l$ ;
14 end
```

4.7 Achievable data rate prediction

Resource assignment and scheduling methods introduced in sections 4.5 and 4.6 rely on the knowledge of $B_{s,w}\ \forall s, w$. In this section, we address the methods for prediction of $B_{s,w}$.

First, let us assume that we have a possibly imperfect prediction of SNR from time $t = 0$ when an assignment is performed to time $t = T_{\text{assign}}$ when the assignment is updated again. Let the SNR prediction as a function of time be denoted by $\gamma_{s,w}(t)$. We can assume that an adaptive modulation mechanism is implemented at the senders, which detects the changes in the SNR to adjust the MCS. However, we assume that the senders will have a more accurate knowledge of the SNR at time t when selecting MCS compared to the prediction $\gamma_{s,w}(t)$ used for resource assignment and scheduling. This is because a sender s can continuously update its estimate of the SNR to worker w based on messages exchanged with each other. On the other hand, the prediction $\gamma_{s,w}(t)$ relies on information up time $t = 0$. Next, let us assume that we have a model $P_{\mathrm{SR}}(\mathrm{SNR}, D_{tx}, b)$, which provides us with the expected frame decoding probability as a function of frame size D_{tx}, SNR and the modulation datarate b. The model $P_{\mathrm{SR}}(\mathrm{SNR}, D_{tx}, b)$ is specific to the physical layer in a particular IEEE 802.11 standard. For example, frame decoding probability models for physical layer based on orthogonal frequency division multiplexing (OFDM), used in 802.11p and other 802.11 standards, can be found in [Mil03]. Furthermore, let p_{SR} be the target frame decoding probability. The achievable data rate is calculated using the following expression:

$$B_{s,w} = \min_{t \in [0, T_{\mathrm{assign}}]} \left(\max_{b \in \mathcal{B}} \left(\mathbf{1}_{P_{\mathrm{SR}}(\gamma_{s,w}(t), D_{tx}, b) \geq p_{\mathrm{SR}}} b \right) \right). \tag{4.11}$$

Therefore, achievable data rate is calculated as the minimum reliable instantaneous data rate from $t = 0$ to $t = T_{\mathrm{assign}}$ based on the predicted SNR $\gamma_{s,w}(t)$ and in the absence of other transmissions. We take the minimum over $[0, T_{\mathrm{assign}}]$ to ensure that at no point during the period $[0, T_{\mathrm{assign}}]$, the task transmission delay will exceed the value expected during resource assignment.

The main challenge in achievable data rate prediction is obtaining an accurate prediction of $\gamma_{s,w}(t)$. Developing methods for prediction of $\gamma_{s,w}(t)$ is not the main focus of this paper. Instead, we will rely on existing approaches to predict connectivity between vehicles and compare them in terms of reliability for the purposes of our cooperative computing framework.

4.7.0.1 Beaconing SNR prediction

The first approach is a beaconing-based approach that has been utilized and analyzed in many instances for vehicular networking and safety applications [SJS14]. The vehicles in the micro cloud transmit beaconing messages every T_b seconds. T_b is typically in the range between 0.1 s and 1 s. For simplicity, we assume that the size of the beaconing messages is small enough that it does not interfere with cooperative computing transmissions. Each vehicle keeps track of the received beacons and if a sender s has not received a beacon from worker w in the past T_d seconds, then they are assumed to be disconnected. Otherwise, the SNR extracted from the most recent message between s and w prior to $t = 0$ is used as the predicted SNR for the next T_{assign} seconds. The SNR information collected at each vehicle about its neighbors can then be relayed to the leader vehicle in the VMC. This makes the prediction for $\gamma_{s,w}(t)$ static, which may not be accurate enough in scenarios with fast moving vehicles where the SNR rapidly changes in the time period $[0, T_{\text{assign}}]$.

4.7.0.2 Deadreckoning SNR prediction

An alternative approach that has been utilized in the past is based on dead reckoning, whereby future vehicle locations are predicted based on their past locations and their velocities. The predicted vehicle locations are utilized in combinations with channel modelling to predict the SNR [ZTW19]. The distance between a sender and a worker changes as a function of time $d_{s,w}(t)$. We assume that the vehicles are periodically disseminating basic safety messages at period T_b, which include the current location $\mathbf{p}_i$ of a vehicle i and its velocity $\mathbf{v}_i$. Position and velocity information extracted from the basic safety messages can then be relayed to the leader vehicle in the VMC. Let Δt_i be the age in seconds of the last safety message transmitted by vehicle i at time $t = 0$. The estimated distance between a sender s and a worker w obtained using dead reckoning is $\hat{d}_{s,w}(t) = ||(\mathbf{p}_s + \mathbf{v}_s(\Delta t_s + t)) - (\mathbf{p}_w + \mathbf{v}_w(\Delta t_w + t))||_2$. Then, using the model for path loss and SNR models in Sec. 4.3.5.4, the SNR can be predicted using the dead reckoning separations $\hat{d}_{s,w}(t)$.

4.8 Resource assignment and scheduling duration adaptation

By using methods for SNR prediction, we can predict how the achievable data rates between vehicles will change over time. This was necessary to account for changes in the achievable data rate between vehicles between each resource assignment and scheduling, since it is done periodically. SNR prediction can be achieved with the dead reckoning method for SNR prediction that we described in Sec. 4.7, for example. Next, we show how by using the SNR prediction methods it is possible to adaptively select the duration of a resource assignment and scheduling T_{assign}.

Ideally, T_{assign} should be as large as possible to reduce the computing overhead of finding the optimal resource assignment and scheduling and to reduce the overhead of transmitting the information needed for resource assignment and scheduling and cooperative computing in the VMC. However, increasing T_{assign} should not come at the expense of decreasing the QoS of task streams.

We formulate this problem as follows. Let us assume that T_{assign} can be selected from a discrete set of A values $\mathcal{T} = \{T^{(1)}_{\text{assign}}, \dots, T^{(A)}_{\text{assign}}\}$. Depending on selected $T^{(a)}_{\text{assign}}$, the achievable data rate between two vehicles is denoted by $B_{s,w,a}$, which can be obtained using Eq. 4.11. Let $t^{tx}_{k,w,l,a}$ be the task transmission duration if the assignment duration is $T^{(a)}_{\text{assign}}$, $t^{tx}_{k,w,l,a} = \sum_{s\in\mathcal{S}_k} \hat{b}_{k,l}\frac{D_{tx}}{B_{s,w,a}}$. Then, we can extend the problem (P1) to include adaptation of assignment duration as follows:

$$\max_{v_a, x_{k,w,l}, z_{k,w,l}, q_{k,l}} \sum_{k,l} l q_{k,l} + \epsilon \sum_{a} \frac{T^{(a)}_{\text{assign}}}{T^{(A)}_{\text{assign}}} v_a \tag{P5}$$

$$\text{s.t. (C1.2) to (C1.9)}$$

$$\frac{c_{k,l}}{f_w x_{k,w,l}} + t^{tx}_{k,w,l,a} \leq \tau_k$$

$$\text{if } (x_{k,w,l} > 0 \wedge v_a = 1)\ \forall k, w, l, a \tag{C5.1}$$

$$\sum_a v_a = 1 \tag{C5.2}$$

$$v_a \in \{0, 1\}\ \forall a \tag{C5.3}$$

In (P5), we have introduced a set of binary variables $v_a\ \forall a$. $v_a = 1$ if the resource assignment and scheduling duration that is selected is $T^{(a)}_{\text{assign}}$, and 0 otherwise. The objective function has an additional term $\epsilon \sum_a \frac{T^{(a)}_{\text{assign}}}{T^{(A)}_{\text{assign}}} v_a$ that incentivizes larger assignment duration. We set $\epsilon < 1$ to ensure that increasing the duration of resource assignment and scheduling never comes at the expense of decreasing the sum QoS.

We described how assignment duration adaption can be incorporated into to the generic resource assignment and scheduling problem (P1). The same extension can be applied to the resource assignment and scheduling problems (P2) and (P4). The exact solution methods to solve (P2) and (P4) are based on these problems being MILP problems, hence the addition of resource adaptation into (P2) or (P4) should not change this. This can be accomplished using the big-M method. First, let us show how the delay constraints (C2.2) should be updated:

$$\frac{c_{k,l}}{f_w} + x_{k,w,l} t^{tx}_{k,w,l,a} \leq x_{k,w,l}\tau_k + (1 - x^b_{k,w,l})\frac{c_{k,l}}{f_w} + M_1(1 - v_a) \quad \forall k, w, l, a \quad \text{(C2.2*)}$$

where the constant M_1 is large enough to ensure that the a delay constraint will not be binding unless $v_a = 1$. The same procedure can be applied to the other constraints in (P2) and (P4) that depend on the achievable data rate: (C2.1) and (C4.1). For the sake of brevity, we omit the full explanation.

We next describe how to adapt Algorithms 1 and 2, for approximately solving (P2) and (P4) respectively, to determine the optimal resource assignment and scheduling duration as well as resource assignment and scheduling. We utilize the fact that these algorithms are computationally inexpensive and run them for each resource duration in $\{T^{(1)}_{\text{assign}}, \ldots, T^{(A)}_{\text{assign}}\}$ independently to obtain A solutions for $x_{k,w,l}$, $z_{k,w,l}$, $q_{k,l}$. Then, we can evaluate each solution with respect to the objective function in (P5) and select the optimal resource assignment and scheduling and its duration.

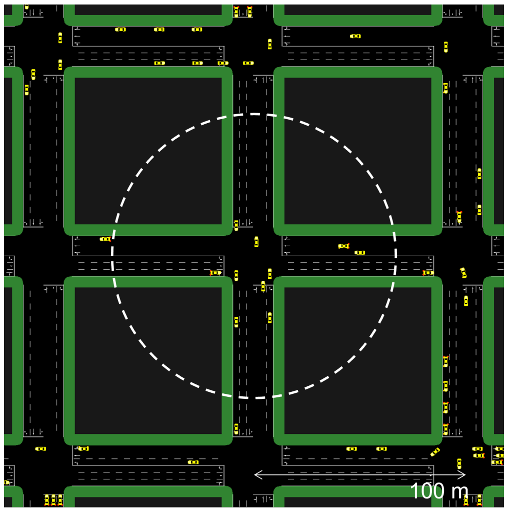


Figure 4.2: Simulated intersection where VMC is formed by the cars entering within its radius.

4.9 Results

4.9.1 Simulation environment

We evaluate our approaches in VMCs based on IEEE 802.11p standard. We chose IEEE 802.11p standard due to the availability of well developed vehicular networking simulators for this standard. Our simulation framework that utilizes Veins vehicular networking simulator [SEB19] and a custom Python-based simulator on top of Veins that runs cooperative computing applications and the resource assignment and scheduling algorithm. The MAC layer of IEEE 802.11p has some notable differences compared to 802.11 WiFi standards, which include channel switching between control and service channels, lack of acknowledgement messages and lower data rates due to channel bandwidths being 10 MHz (compared to 20 MHz in WiFi) [ES12]. To make the conclusions obtained from our results applicable to 802.11 WiFi standards, we enabled acknowledgements and disabled channel switching in the simulations. Veins vehicular networking simulator utilizes SUMO simulator that provides realistic simulations of urban vehicular mobility [BBE11]. Each Monte Carlo iteration lasts 100 seconds. Both incumbent and cooperative computing traffic utilize frame size of 10 Kb and priority category AC_BK. The definitions of priority categories in 802.11p can be found in [ES12]. We implemented adaptive MCS selection in Veins, which relies on perfect knowledge of the SNR between transmitters and receivers. We use ideal an MCS selection scheme to exclude the effects of imperfect MCS on the results. Retransmissions were enabled on the MAC layer and up to 4 retransmissions of a frame are allowed if frame errors occur.

Using the SUMO simulation framework, we simulate a 6 by 6 Manhattan grid where each block is 100 m long. An VMC is formed by vehicles that are within 80 meters from the center of one of the inner intersections in the Manhattan grid. However, if a vehicles is assigned to process a task stream it may continue to do so even after leaving the VMC, so long as it remains connected to the senders. All cars in the VMC can act as workers and all have equal computing capabilities. The intersection is shown in Fig 4.3. We generate three different traffic scenarios with three different traffic densities in the Manhattan grid. Namely, the number of cars in the grid is set to 200, 100 and 50. The average number of cars in the VMC in each of these scenarios is shown in Fig. 4.3. Since the average speed of vehicles is linked to the traffic density, we also show the average speed in the VMC in Fig. 4.3.

Four static incumbent nodes are randomly placed in the VMC radius as controlled sources of incumbent interference. Two incumbents act as transmitters and two act as receivers. Simulated incumbent devices are used to emulate interference that would normally occur due to other vehicles or roadside devices utilizing the channel. The incumbent frame rate is varied across different simulation results between 0 and 80 frames per second. The average portion of the time on the channel occupied by the incumbents prior to any cooperative computing taking place is shown in Fig. 4.4. The MCS of incumbent transmissions is static and set to 6 Mbps.

Next, we describe the characteristics of application task streams. Each task stream only has one sender, i.e. $|\mathcal{S}_k| = 1$. Any car that enters the VMC is assigned to be a sender at probability 0.2. Each task stream has 6 QoS levels, which are related to the task processing

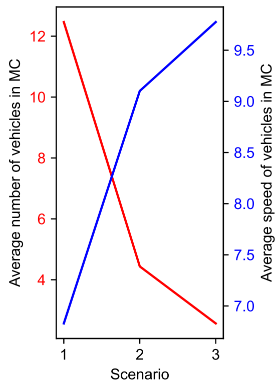


Figure 4.3: Average speed and number of vehicles in the VMC across three different scenarios

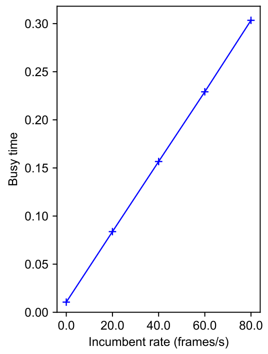


Figure 4.4: Busy time on the channel across different incumbent transmission rate levels.

rate $r_{k,l}$. The task input data size $b_{k,l}$ and compute load $c_{k,l}$ are constant across task stream QoS levels. Alternative task stream definitions are possible, where QoS is linked to $b_{k,l}$ or $c_{k,l}$ but we do not consider those in the simulation results. Two types of task streams with two kinds of input data will be considered: image data processing with $b_{k,l} = 1$ Mb and radar point cloud data with $b_{k,l} = 0.2$ Mb. For image data processing processing, rates are defined as $r_{k,l} = 5l \ \forall l = 1, \dots, 6$ and for point cloud processing the rates are $r_{k,l} = 10l \ \forall l = 1, \dots, 6$. We set lower task rates per QoS level for image data processing since high task rates would not be feasible due to channel capacity constraints of 802.11p. These types of input data appear in applications such as cooperative perception or augmented reality. In practice, image data size may exceed 1 Mb without compression but due to the limitations of the 802.11p PHY layer data rates, we restrict ourselves to this size. The maximum latency per task is set to $\tau_k = 0.5$ s. Instead of varying $c_{k,l}$ and f_w separately across simulations, it is sufficient to vary their ratio $\frac{f_w}{c_{k,l}}$ because these parameters always appear as a ratio in expressions for compute delay and computing rate (see Sec. 4.3.4). The resource assignment and scheduling duration is set to $T_{\text{assign}} = 1$ s, unless stated otherwise.

4.9.2 Resource assignment and scheduling approaches

We consider the following approaches for cooperative computing of task streams:

- Non-cooperative computing: All task streams are processed at the receivers. This approach serves as a benchmark to quantify the benefits of cooperative computing compared to independent task processing.
- MAC 1: This is the short-hand name that we use for resource assignment and scheduling approach explained in Sec. 4.5.2.
- MAC 2: This is the assignment approach explained in Sec. 4.6.2.
- MAC 1 Approx: This is the approach explained in Sec. 4.5.3 and is a polynomial time approximation of MAC 1 approach. For the parameters in Algorithm 1, we set $V = 2|\mathcal{W}|/K$ and $\mu = 2$.
- MAC 2 Approx: This is the approach explained in Sec. 4.6.3 and is a polynomial time approximation of MAC 2 approach. We use the solution obtained using MAC 1 Approx as the starting point in Algorithm 2.

The purpose of the latter four approaches is to quantify the benefits of cooperative computing. Furthermore, these approaches have different levels of computational complexity and need different levels of information about incumbent interference. MAC 1 Approx and MAC 2 Approx are the fastest since they run in polynomial time. MAC 1 and MAC 2 are slower since they rely on MILP solvers, but MAC 1 is expected to be faster than MAC 2 because it has significantly fewer constraints. MAC 1 and MAC 1 Approx only need to know C_{IDLE} to operate, whereas MAC 2 and MAC 2 Approx need to know detailed parameters of incumbent interference, as explained in Sec. 4.6.

4.9.3 Task assignment

Resource assignment and scheduling methods determines how many tasks will be processed by each worker over the next T_{assign} seconds, which is given by $T_{\text{assign}} z_{k,w,l}\ \forall k, w, l$. However, this still does not determine which specific tasks in the task stream will be processed by which specific workers. We use a round-robin assignment algorithm that we empirically established to minimize the queuing delays at the workers. Note that each worker w will have a separate queue per each task stream k it processes, which will be allocated $x_{k,w,l}$ share of the worker's computing power if the QoS level of k is l. The assignment policy operates as follows. Per each resource assignment and scheduling, the round-robin algorithm loops over each of the workers that had been sorted randomly and assigns each of the $T_{\text{assign}} r_{k,l}$ tasks to one worker per loop unless that worker has already been assigned $T_{\text{assign}} z_{k,w,l}$ tasks. The looping over workers ends once all $T_{\text{assign}} r_{k,l}$ tasks have been assigned.

4.9.4 Gains of cooperative computing under different resource assignment and scheduling approaches

Next, we evaluate all approaches in terms of obtained QoS and their congestion control capabilities. First, we present results for point cloud processing across different levels of incumbent interference in Fig. 4.5 and with $\frac{f_w}{c_{k,l}} = 10$. For this set of results, the SNR predictions $\gamma_{s,w}(t)$ are error-free. This is accomplished by doing a mock run before each Monte Carlo iteration and recording the SNR values between vehicles over time. We do this to remove the effects of incorrect SNR prediction on task loss.

The average QoS results shown in Fig. 4.5 correspond to the average scheduled QoS per task stream in the VMC. For all proposed resource assignment and scheduling algorithms, there is an increase in QoS compared to the non-cooperative benchmark, which maps to a higher task processing rate per stream. However, even though a task stream may be assigned a particular QoS level during resource assignment and scheduling, that level may not

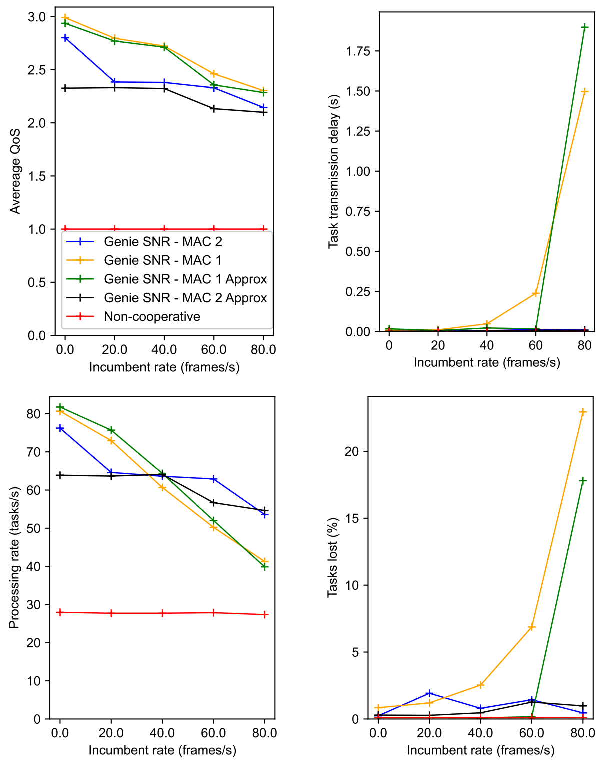


Figure 4.5: Results for point cloud processing for different resource assignment and scheduling approaches and under different levels of incumbent activity.

be supported by the wireless channel. Indeed, this is what we observe with MAC 1 and MAC 1 Approx approaches in the task transmission delay results. For higher incumbent rates, these approaches cause congestion at one or more senders which is reflected in large task transmission delay. On the other hand, when resource assignment and scheduling is performed using MAC 2 and MAC 2 Approx approaches, the task delay remains approximately constant because these approaches have more accurate congestion control mechanisms.

In our simulations, we declare any task that is not processed within the deadline τ_k as lost. The cumulative task processing rate of tasks processed before the deadline is shown in Fig. 4.5, along with the percentage of lost tasks. Note that a small percentage of tasks are lost even with MAC 2 and MAC 2 Approx approaches, which occurs due to collisions on the wireless channel instead of congestion. Such loss could be minimized by introducing redundancies into task transmissions, however this is beyond the scope of this paper.

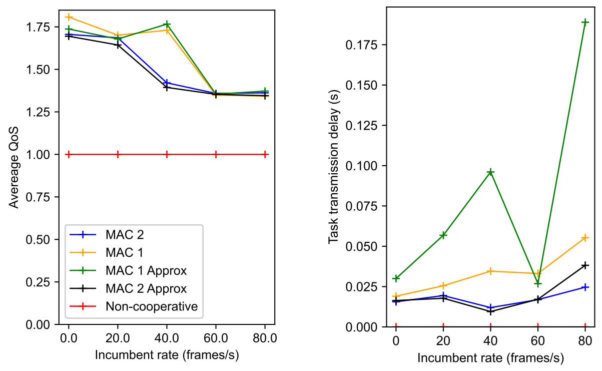


Figure 4.6: Results for image processing for different resource assignment and scheduling approaches and under different levels of incumbent activity.

Furthermore, the approximate methods MAC 1 Approx and MAC 2 Approx perform very similarly to their exact counterparts in terms of QoS and congestion control accuracy. This is valuable since these methods have a polynomial time complexity and can therefore be reliably used as a substitute for their exact counterpart when the number of workers and task streams is large.

Next, in Fig. 4.6, we show results for image processing with $\frac{f_w}{c_{k,l}} = 5$. For these results, we scale the ratio $\frac{f_w}{c_{k,l}}$ down to 5 to match the lower task rates $r_{k,l}$ of the image processing tasks. First, we observe that due to larger input data size, the QoS gains relative to the non-cooperative benchmark are smaller compared to processing of point cloud data. This occurs because fewer tasks can be transmitted over the channel due to capacity constraints. Overall, we notice similar trends as in Fig. 4.5. The approaches MAC 1 and MAC 1 Approx are more greedy in scheduling task streams however their congestion control accuracy is only reliable in absence of incumbent interference. Nevertheless, these approaches remain useful as part of MAC 2 Approx algorithm, where they provide the initial solution.

Next, in Fig. 4.7, we present results for point cloud processing across the three traffic scenarios with $\frac{f_w}{c_{k,l}} = 10$. The incumbent rates are set to 40 frames per second. First, we can observe that the average QoS per task stream remains steady across traffic scenarios, but the total number of processed task decreases from left to right. This occurs because

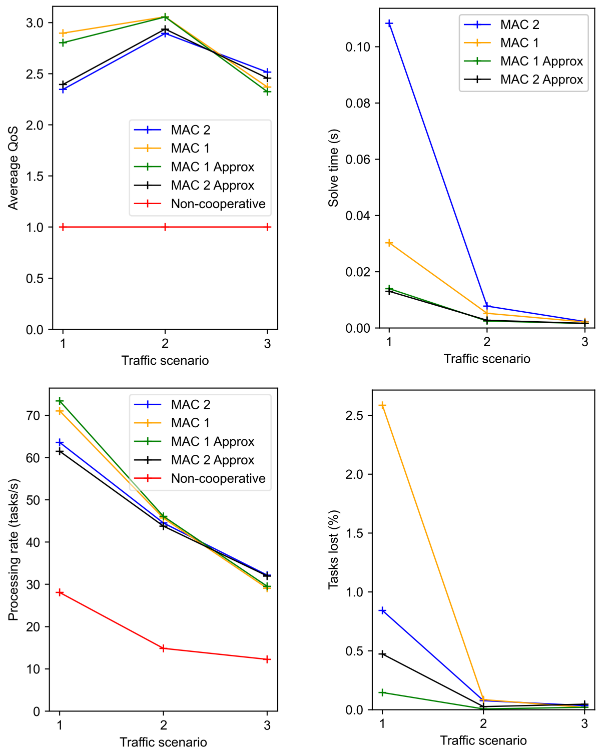


Figure 4.7: Results for point cloud processing for different resource assignment and scheduling approaches in different traffic scenarios.

due to a decrease in traffic density from left to right, the number of task streams or the computing demand decreases. However, the computing resources available per task stream remain similar which is why the average QoS does not change from left to right. The loss rate decreases from left to right because the channel is less utilized so collisions and congestion are less likely to occur. Finally, in Fig. 4.7 we also show the time taken on average for each resource assignment and scheduling algorithm to arrive at a solution. These results are obtained on a workstation with a AMD Ryzen Threadripper PRO 5975WX 32-Core CPU. The solution time decreases from left to right since the number of workers and task streams decreases. Furthermore, as expected, approaches MAC 2 and MAC 1 have a higher complexity than the polynomial-time algorithms MAC 1 Approx and MAC 2 Approx. However, it is important to highlight that these results are dependent on the workstation hardware and also on the parameters of the resource assignment and scheduling problem.

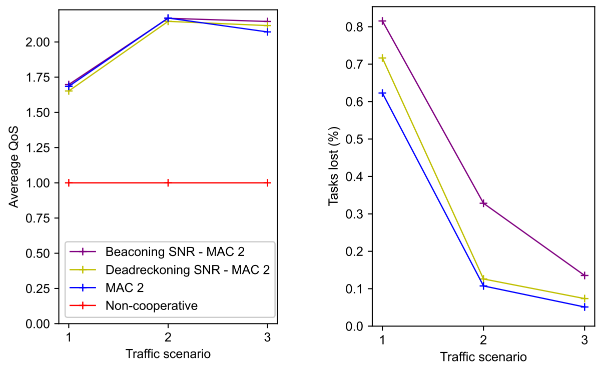


Figure 4.8: Results for image processing for different achievable data rate prediction methods when $T_{\text{assign}} = 1$ s.

4.9.5 Evaluation of achievable data rate prediction methods

Next, we compare the approaches for achievable data rate prediction introduced in Sec. 4.7 against achievable data rate prediction based on perfectly known SNR. The results are shown in figures 4.8 and 4.9, where $T_{\text{assign}} = 1$ s and $T_{\text{assign}} = 5$ s, respectively. We vary T_{assign} since achievable data rate prediction becomes more challenging with higher T_{assign} due to the need to predict the SNR further into the future. We also run simulations in different traffic scenario in figures 4.8 and 4.9 because the vehicle speed increases with lower density, which also makes the achievable data rate prediction more challenging. For these results the incumbent rates are 0 and the resource assignment and scheduling method used was MAC 2, which is chosen to minimize the loss due to congestion.

For $T_{\text{assign}} = 1$ s, both beaconing and dead reckoning methods perform similarly to the ideal case with perfectly known SNR, which is labeled as MAC 2 in the figures. However, the task loss is slightly higher when using beaconing, because this approach does not perform well in scenarios with fast moving vehicles. For $T_{\text{assign}} = 5$ s, the beaconing method becomes inadequate, which is evident as in the task loss results in Fig. 4.9. This occurs because over longer T_{assign} vehicles move significantly and SNR values obtained using beaconing become outdated. On the other hand, the dead reckoning method performs well even for longer T_{assign}.

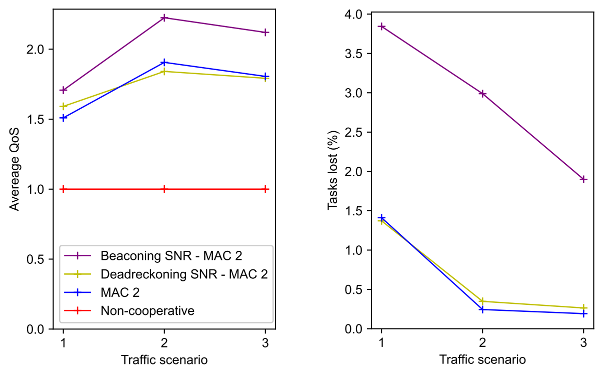


Figure 4.9: Results for image processing for different achievable data rate prediction methods when $T_{\text{assign}} = 5$ s.

4.9.6 Adaptation of resource assignment and scheduling duration

For all the presented results so far, T_{assign} was a predetermined parameter. However, using the methods described in Sec. 4.8, T_{assign} can be adaptively selected based on the state of the VMC. The results with adaptive T_{assign} selection and with point cloud task streams are shown in Fig. 4.10. To obtain these results, we use $\mathcal{T} = \{1, 2, \ldots, 10\}$ and use all three SNR prediction approaches discussed so far. Since the incumbent rates are set to 0, we use MAC 1 approach for resource assignment and scheduling. First, we observe that the average selected T_{assign} is roughly constant across different traffic scenario. This is somewhat surprising since the average vehicle speed increases from left to right, so it would be expected that T_{assign} should decrease. This can be explained by the fact that resource assignment and scheduling algorithm can match workers and senders that are moving in the same direction, so resource assignment and scheduling remains valid for a longer period of time. Furthermore, the total processing rate remains the same as in Fig. 4.7, where shorter $T_{\text{assign}} = 1s$ was used, which means computing resources remain efficiently used. This is not the case for assignment based on beaconing since a significant number of tasks are lost due to inaccurate SNR prediction.

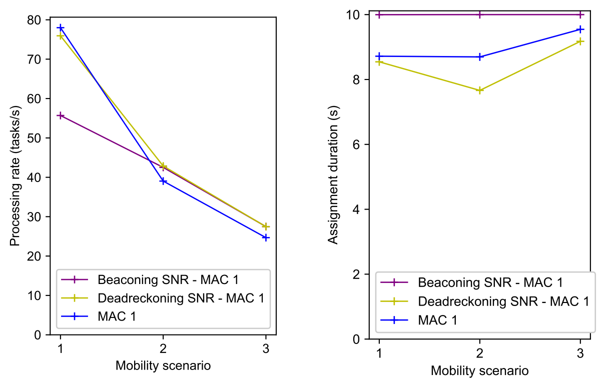


Figure 4.10: Results for point cloud processing in different traffic scenarios with adaptive selection resource assignment and scheduling duration T_{assign}.

4.10 Summary

In this work, we focus on the use case of cooperative computing to support sensing-based computationally-intensive and delay-constrained vehicular applications such as cooperative perception, augmented reality, driving assistance and navigation. To this end, we develop resource assignment and scheduling methods that will maximize the QoS for computing applications given the available vehicles in the VMC. Our proposed approaches adapt to link quality changes between vehicles and prevent congestion on 802.11 channels even in the presence of incumbent interference. The proposed approaches or their approximate alternatives have a low computational complexity such that they can be used in real-time. Furthermore, the approaches adapt to mobility of vehicles to control the frequency of resource assignment and scheduling updates. The proposed approaches are evaluated in realistic simulators of vehicular networking based on IEEE 802.11p standard with a MAC layer that was adapted to closely match 802.11 WiFi standards. This makes the conclusions that we make using our simulation results applicable to 802.11 WiFi standards. The simulation results demonstrate the importance of accurate congestion prediction for reliable cooperative computing. This is most notable in presence of incumbent transmissions, which can originate from other vehicles in the micro cloud or due to roadside devices. We further validate the performance of our methods in different traffic scenarios and demonstrate consistent gains in QoS compared to non-cooperative computing. Next, we evaluated two practical methods for prediction of SNR between vehicles in terms of their viability for use in cooperative computing. Using these SNR prediction methods we show how our approach can be extended to adaptively control the resource assignment and scheduling update frequency.

CHAPTER 5

Hybrid Vehicular and Cloud Distributed Computing: A Case for Cooperative Perception

In this work, we propose the use of hybrid offloading of computing tasks simultaneously to edge servers (vertical offloading) via LTE communication and to nearby cars (horizontal offloading) via V2V communication, in order to increase the rate at which tasks are processed compared to local processing. Our main contribution is an optimized resource assignment and scheduling framework for hybrid offloading of computing tasks. The framework optimally utilizes the computational resources in the edge and in the micro cloud, while taking into account communication constraints and task requirements. While cooperative perception is the primary use case of our framework, the framework is applicable to other cooperative vehicular applications with high computing demand and significant transmission overhead. The framework is tested in a simulated environment built on top of car traces and communication rates exported from the Veins vehicular networking simulator. We observe a significant increase in the processing rate of cooperative perception sensor frames when hybrid offloading with optimized resource assignment is adopted. Furthermore, the processing rate increases with V2V connectivity as more computing tasks can be offloaded horizontally.

5.1 Introduction

Cooperative vehicular perception systems seek to expand a vehicle's field of view by enabling cars to share the data from their environment perception sensors via wireless communication and thereby allowing vehicles to achieve a global view of the traffic environment. However, granted a high speed wireless communication channel, experimental implementation studies of cooperative perception systems report that the main bottleneck in the frame rate at which cooperative perception systems can operate is the vehicular computational power [QAB18].

Cooperative perception and similar computationally demanding and time-sensitive vehicular applications can benefit from additional computing power. Edge computing is a promising paradigm where computing resources are placed in close proximity of end users, which are usually mobile. Vehicles can offload some of the computational tasks to edge servers which they can reach via an LTE connection. The results of the processed tasks are then be sent back from the edge servers to the interested cars on the road. We refer to this type of offloading as *vertical offloading*. The vehicle cloudification framework, a paradigm that involves forming virtual cloud servers from vehicles in proximity of each other on the road, can enable *horizontal offloading*. In horizontal offloading, the computing resources and the coordination of the vehicular micro cloud can be utilized for task offloading via V2V communication. The feasibility of forming stable vehicular micro clouds on the road has been confirmed in [HJD17].

In addition to the rate at which sensor frames are processed, there are other considerations that need to be made when it comes to cooperative perception systems. Firstly, the processing delay of a frame must not be too long, otherwise the processing results would be out of date relative to the actual state of the traffic environment. When offloading tasks, the processing delay includes data frame transmission delay and computing delay. Secondly, we need to assume that there is a limit on the amount of data that can be transmitted over the cellular and V2V links. For the cellular data exchange, the practical reason for this is that there is normally a monetary cost related to the utilization of the cellular link and that there might also be a limit on how much the cellular providers will allow the network to be loaded by the transmission of vehicular perception data, since it could negatively impact the quality of service for other cellular users. Likewise, there is a practical limit on how much data transfer can occur over V2V links, since the data transfer of cooperative perception data could cause congestion and disrupt other services that rely on the shared V2V channel.

Approaches to offloading of vehicular computing tasks to the edge or to the nearby cars have been proposed in the literature before. A framework for task allocation in horizontal offloading was proposed in [HUH19]. However, in this work, the delay due to transmission of the computing task data has not been considered, therefore this particular framework cannot be applied on cooperative perception or other computing tasks that would have a large transmission overhead when offloaded. Another comprehensive framework for task offloading was developed in [FLW17]. While [FLW17] considers the data transfer of computing tasks, it focuses only on horizontal task offloading. A mechanism for both horizontal and vertical task offloading was proposed in [ZPX18]. While the system proposed in [ZPX18] could in theory be applied to cooperative perception or other data intensive computing tasks, it does not give any consideration to cellular/V2V traffic overhead limitations.

In this chapter, we propose *hybrid horizontal and vertical offloading* of time-sensitive cooperative perception computing tasks with the goal of increasing the rate at which these tasks are processed. We design a resource allocation and task scheduling algorithm that maximizes the rate at which tasks are processed, while considering the limits on the amount of data that can be transferred, the delay constraints and the rate at which data frames can be transmitted.

The rest of this chapter is organized as follows. In Section 5.2, we describe the model of the system that we assume in the development of our algorithm. The Section 5.3 outlines our design goals and the details of the proposed algorithm. In the Section 5.4 we report the main results from the testing of our approach via simulation. The Section 5.5 provides the main conclusions of this work.

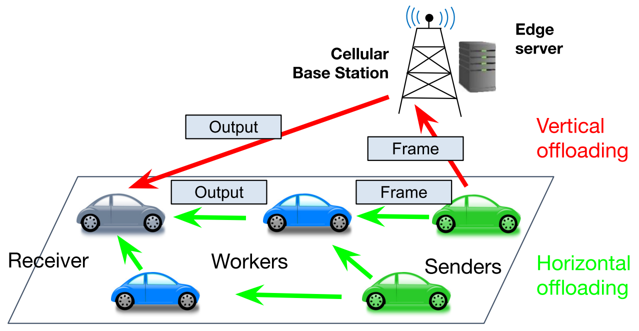


Figure 5.1: Overview of the system model.

5.2 System model

In this section, we describe the model of vehicular micro cloud and edge computing resources and the model of computing tasks for cooperative perception used in the remainder of this chapter.

5.2.1 Vehicular and edge computing resources

We consider a segment of a road or an intersection occupied by a set of cars $\mathcal{C}$, as illustrated in Fig. 5.1. A subset of cars $\mathcal{S} \in \mathcal{C}$ act as senders sharing their sensor data with other vehicles on the road. Each sender has a receiver set of cars $\mathcal{R}_s \in \mathcal{C}$, $\forall s \in \mathcal{S}$, interested in the data from the senders. We assume that all senders and receivers are V2V capable. Furthermore, the senders and receivers, together with other V2V capable cars form a stationary vehicular micro cloud $\mathcal{C}'$. A stationary micro cloud is formed by cars that are present in particular fixed geographical area [HSH17]. The cars can enter and exit the area at any time and therefore the set $\mathcal{C}'$ changes over time. On the other hand, the set of edge servers is constant over time.

Senders and receivers can connect to a set of edge servers $\mathcal{B}$ via LTE connection through the nearby base stations (BSs). While we assume that all cars have cellular communications capability, only a fraction of the vehicles is capable of V2V communication, based on the current trends in car industry.

We define a set of worker nodes $\mathcal{W} \in \mathcal{B} \cup \mathcal{C}'$ that can be utilized for offloading of the processing of perception frames from the senders. The cars that take on the role of workers have computing resources available that can be used for processing of cooperative perception frames. We assume that there exist some form of incentive mechanism or agreement between car manufacturers that would lead to sharing of computing resources and sensor data between cars. The sets $\mathcal{S}$, $\mathcal{W}$ and $\mathcal{R}_1, ..., \mathcal{R}_{|\mathcal{S}|}$ can be overlapping, i.e., the cars may have multiple roles from amongst the sender, receiver and worker role at the same time.

We denote the computational power, measured in Hz, of a node i (either a car or an edge server) as F_i. Furthermore, we denote the communication rate between two nodes, i and j, as $R_{i,j}$. We use an indicator variable $D_{i,j}^{\mathrm{LTE}} \in \{0, 1\}$, that is equal to 1 if communication between nodes i and j is cellular, i.e. one node of the nodes is a car and other is an edge server. Another indicator variable, $D_{i,j}^{\mathrm{V2V}} \in \{0, 1\}$, is used to capture whether the communication between the nodes i and j is V2V, i.e., both of the nodes are vehicles. Only $D_{i,j}^{\mathrm{V2V}}$ or only $D_{i,j}^{\mathrm{LTE}}$ can be equal to 1 for any given pair of nodes.

5.2.2 Computing tasks for cooperative perception

A cooperative perception system relies on a variety of computing tasks that need to be performed before the sensor data from the senders can become useful to the receivers. Depending on the type of the sensor data, there are different types of computing tasks that might need to be performed. For example, on 3D data, generated by radar or lidar, feature extraction and localization needs to be performed using simultaneous localization and mapping (SLAM) algorithms at either the sender (the source of sensor data) or receiver cars (cars interested in the information provided by the sender). For image data, the computing tasks include feature detection and perspective transformation. We describe a computing task l by a tuple of parameters $S_l = \{d_l, c_l, s_l, \mathcal{R}_l, t_l, \tau_l\}$, where:

- d_l is the data size of the sensor frame to be processed

- c_l is the computational load measured in CPU cycles per second
- s_l is the source sender of the task
- $\mathcal{R}_l$ is the set of receiver nodes that are interested in receiving the output of the task
- t_l is the time instance when the sensor frame is captured
- τ_l is the maximum delay. The delay is measured from t_l until the task is processed and its output delivered to the receiver

We assume that every task l belongs to a particular task type (TT) $k \in \{1, ..., K\}$ that has a unique set of parameters $A_k = \{d_k, c_k, s_k, \mathcal{R}_k, \tau_k\}$. Therefore, for every task l, $\{d_l, c_l, s_l, \mathcal{R}_l, \tau_l\} \in \{A_k : k = 1, ..., K\}$. For example, a TT can be feature extraction on a 3D point cloud coming from a particular sender s_k, with a delay tolerance τ_k, input frame size d_l and a receiver set $\mathcal{R}_k$. Computationally demanding tasks can create a bottleneck in how many frames per second can be processed and shared between the sender and receiver cars. Assuming that computation is the bottleneck, the senders will generate sensor frames at the rate at which they can be processed. The sensor generation rate determines a TT arrival rate, and therefore task arrival time.

5.3 Proposed framework

In this section, we propose an optimization framework for offloading of vehicular computing tasks to the edge and the vehicular micro cloud.

At the beginning of each period, there is a demand for completion of certain set of TTs, $\{A_k : k = 1, ..., K\}$. We maximize the number of tasks/frames of all TTs that are processed over the next period T. Our algorithm is a two step algorithm. In the first step, we assign the computation resources $\{F_w : w \in \mathcal{W}\}$ and communication resources $\{R_{i,j} : i, j \in \mathcal{C} \cup \mathcal{B},\ i \neq j\}$ to particular TTs. The resource assignment determines how many tasks of each TT will be processed at each individual worker. Assuming that computation is the bottleneck, the senders will generate data frames to meet the achieved processing rate. In the second stage, we schedule data frame/task generation and assign where each task is processed since several workers may be processing the tasks of the same type.

The optimization is done in a centralized manner. We assume that one of the cars in the vehicular micro cloud, potentially the micro cloud leader, or one of the edge servers performs the resource assignment and task scheduling based on the knowledge of communication rates $R_{i,j}$ and computing powers F_i of cars and edge servers. Since the state of a traffic scenario changes over time, the offloading decisions need to be updated periodically. Our algorithm is applied periodically at interval T to obtain the resource assignment and scheduling, which is then shared with all nodes.

5.3.1 Resource assignment

We pose the resource assignment problem as a non-linear optimization problem and then approximate it as a mixed integer linear optimization problem.

In the resource assignment stage, we solve for three optimization variables:

- $X_{k,w} \in [0,1]\ \forall k, w$, that determines the share of computing resources allocated by each worker w for completion of tasks of type k
- $Y^{\mathrm{LTE}}_{s,k,w} \in [0,1]\ \forall s, k, w$, that determines the share of time resources the sender s will spend on the transmission of data frames of TT k to worker w via cellular network
- $Y^{\mathrm{V2V}}_{s,k,w} \in [0,1]$, that, similarly to $Y^{\mathrm{LTE}}_{s,k,w}$, determines the allocation of V2V communication resources.

The delay of processing a task of a certain type k needs to be less than τ_k. We assume that only two types of delay are significant: the delay of transmitting the data frame to the worker if a task computation is offloaded and the delay of task processing. The data size of the task output is assumed to be negligible. The maximum rate of transmission between a sender of TT k, s_k, and a worker w_k is $R_{s_k,w}$. However, given that the sender only spends a fraction of time, $Y_{s_k,k,w}$, transmitting that particular TT to worker w, the effective rate of transmission is $R_{s_k,w}Y_{s_k,k,w}$. The delay of transmitting a data frame of a task of type k to worker w is $\frac{d_k}{R_{s_k,w}Y_{s_k,k,w}}$. The delay of processing a task at worker w is $\frac{c_k}{F_w X_{k,w}}$, where $F_w X_{k,w}$ is the effective computing rate of a TT k at w. We define the resource assignment problem as a non-linear program:

$$\max_{X_{k,w},Y^{\text{LTE}}_{s,k,w},Y^{\text{V2V}}_{s,k,w}} \quad \sum_k \sum_w \left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor \tag{P1}$$

$$\text{s.t.} \quad d_k \left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor \leq TR_{s_k,w}\big(D^{LTE}_{s_k,w}Y^{LTE}_{s_k,k,w} + D^{V2V}_{s_k,w}Y^{V2V}_{s_k,k,w}\big) \tag{C1}$$

$$\sum_k \sum_w D^{LTE}_{s_k,w} d_k \left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor \leq U^{\text{LTE}}T \tag{C2}$$

$$\sum_k \sum_w D^{V2V}_{s_k,w} d_k \left\lfloor \frac{F_w X_{k,w}}{c_k} \right\rfloor \leq U^{\text{V2V}}T \tag{C3}$$

$$\frac{d_k}{R_{s_k,w}Y^{V2V}_{s_k,k,w}} + \frac{c_k}{F_w X_{k,w}} \leq \tau_k \text{ if } X_{k,w} \neq 0 \text{ and } D^{V2V}_{s_k,w} = 1 \tag{C4}$$

$$\frac{d_k}{R_{s_k,w}Y^{LTE}_{s_k,k,w}} + \frac{c_k}{F_w X_{k,w}} \leq \tau_k \text{ if } X_{k,w} \neq 0 \text{ and } D^{LTE}_{s_k,w} = 1 \tag{C5}$$

$$\sum_k X_{k,w} \leq 1, \quad \sum_w \sum_k Y^{LTE}_{s_k,k,w} \leq 1, \quad \sum_w \sum_k Y^{V2V}_{s_k,k,w} \leq 1 \tag{C6, C7, C8}$$

The objective function counts the total number of tasks or data frames that will be processed over the current period T. The floor function $\lfloor \cdot \rfloor$ rounds its argument down to the nearest integer, since only the whole number of tasks that are processed is important.

The Constraint (C1) is applied per each TT k and it limits the rate at which data frames can be transmitted by the effective communication rate $R_{s_k,w}Y^{LTE}_{s_k,k,w}$ if the communication is cellular or the rate $R_{s_k,w}Y^{V2V}_{s_k,k,w}$ if the communication is V2V. The Constraint (C2) limits the amount of data transmitted through the cellular network by an upper bound $U^{\text{LTE}}T$ and the Constraint (C3) limits the amount of data transmitted through V2V by an uppper bound $U^{\text{V2V}}T$. The parameters U^{LTE} and U^{V2V} are defined by the operator of the system to precisely limit the amount of LTE and V2V traffic.

The constraints (C4) and (C5), applied per each TT k, are the delay constraints and they only need to be satisfied for workers w that process tasks of type k, i.e., $X_{k,w} \neq 0$. The delay constraint (C4) applies to data frames that are transmitted via V2V communication and the delay constraint (C5) applies to transmissions that occur over cellular. The Constraint (C6) is applied per each worker and ensures that shares of all computing resources per worker add up to one. Similarly, the constraints (C7) and (C8) are applied per each unique sender s and they ensure that the shares of transmission time per sender add up to 1.

To the best of authors' knowledge, the optimization problem we arrive at cannot be readily solved by any standard optimization techniques, therefore we linearize it by making the necessary approximations. Once the optimization problem is linearized, it can be readily solved by using any mixed integer linear programming solver.

Linearization of the optimization problem

A non-linear expression that appears several times in our problem formulation is $\left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor$. To approximate this expression by a linear function, we introduce an integer variable $V_{k,w} \in \mathbb{Z}^+$. We add two additional constraint sets:

$$V_{k,w} \leq \frac{TF_w X_{k,w}}{c_k}$$

$$\frac{TF_w X_{k,w}}{c_k} - 0.999 \leq V_{k,w}$$

We then replace the expression $\left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor$ by the variable $V_{k,w}$ everywhere in problem formulation.

Next, we linearize the delay constraints (C4) and (C5). We introduce a set of helper constants $\alpha^{(n)} = n/N$, for $n = 0, ..., N$, where N is a positive integer hyperparameter. We also introduce a set of helper binary variables $U_{k,w}^{(n)} \in \mathbb{Z}_2, \forall n, k, w$. To simplify the exposition, we only show how one of the delay constraints can be linearized. To accomplish our goal, we introduce the following constraints that will replace the delay constraint

$$\left(1 - U_{k,w}^{(n)}\right) - \alpha^{(n)} \tau_k \frac{F_w}{c_k} X_{k,w} \leq 0 \ \forall n$$

$$\left(1 - U_{k,w}^{(n)}\right) - \left(1 - \alpha^{(n)}\right) \tau_k \frac{R_{s_k,w}}{d_k} Y_{s_k,k,w}^{\text{LTE}} \leq 0 \ \forall n$$

$$U_{k,w}^{(1)} + U_{k,w}^{(2)} + \cdots + U_{k,w}^{(N)} \leq N - I(X_{k,w} > 0)$$

where $I(X_{k,w} > 0)$ is the indicator function equal to 1 if $X_{k,w} > 0$ and 0 otherwise. This indicator function can easily be linearized using the same approach as with the expression $\left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor$. The greater the value of hyperparameter N selected, the more accurate our approximation becomes.

5.3.2 Task scheduling

The resource assignment determines how many tasks will be processed by each worker over the current period T. Given $X_{k,w}$, the number of tasks of type k processed by worker w is $M_{k,w} = \left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor$ and the total number of tasks of type k that will be processed is $L_k = \sum_w M_{k,w}$. Given that workers can process L_k tasks of a particular TT k and assuming that that the computation is the bottleneck, the arrival rate of sensor frames and, hence, the tasks will meet the processing rate. Expressed mathematically, the arrival times of tasks $l = 1, ..., L_k$ are $t_l = \frac{(t_l - 1)T}{L_k}$. The assignment of tasks to workers is performed using a heuristic policy. We use a round-robin assignment algorithm that we empirically established to minimize the queuing delays in transmission queue at the sender and the processing queue at the worker. Let $Z_{l,w} \in \mathbb{Z}_2$ be an assignment variable equal to 1 if the task l is assigned to worker w and 0 otherwise. The heuristic round robin assignment policy is described in the pseudo-code below: The policy is applied per each TT separately. The algorithm

Algorithm 3: Round-robin task assignment for a TT k

```
Result: $Z_{l,w}$
1  $Z_{l,w} \leftarrow 0 \ \forall w \in \mathcal{W}, \forall l = 1, ..., L_k$
2  $M_{k,w} \leftarrow \left\lfloor \frac{TF_w X_{k,w}}{c_k} \right\rfloor \ \forall w \in \mathcal{W}$
3  $L_k \leftarrow \sum_w M_{k,w}$
4  $l \leftarrow 0$
5  while $l < L_k$ do
6      for $w \in \mathcal{W}$ do
7          if $M_{k,w} \neq 0$ then
8              $Z_{l,w} \leftarrow 1$
9              $M_{k,w} \leftarrow M_{k,w} - 1$
10             $l \leftarrow l + 1$
11         end
12     end
13 end
```

continuously loops over each of the workers that are sorted randomly and assigns each of the L_k tasks to one worker in each loop iteration unless that worker has already been assigned $M_{k,w}$ tasks. The looping over workers ends once all L_k tasks have been assigned.

5.4 Simulation results

In this section, we analyze the performance of the proposed approach for horizontal and vertical computing offloading for cooperative perception in a simulated traffic environment. We simulate an intersection in the Luxembourg SUMO traffic (LuST) scenario [CFE15]. The simulated intersection is located at 49°36'34.7"N, 6°07'09.7"E and the micro cloud area radius is 150 m. We assume that all V2V capable cars that enter the circular area become members of the vehicular micro cloud. The traffic is simulated for one hour between 8 AM and 9 AM.

A share η_S of cars in the micro cloud are randomly selected to be senders and a share η_R are randomly selected to be receivers. All the cars in the micro cloud, including the senders, together with one edge server form the set of workers.

The cars communicate to each other via DSRC V2V communication while the communication to the edge server is done via LTE. In our simulations, we model the LTE and V2V communication links as pipes with some rate $R_{i,j}$. In the LUST scenario, using Veins [SGD10], we simulate the cars broadcasting beaconing signals via DSRC three times per second, which we use to estimate the signal to noise plus interference ratio (SINR) between cars in the area. The SINR is mapped to DSRC communication rates based on the mappings reported in [JCD08]. The LTE communication rate to the edge server is modeled as a Gaussian random variable $\mathcal{N}(\mu_R, \sigma_R)$. We assume that the knowledge of the communication rates $R_{i,j}$ is available to the node that calculates the optimal resource assignment. In practice, these values would need to be obtained via some distributed or centralized rate prediction algorithm

The computing power F_i of sender cars is $\mu_C^{(1)}$. From the remaining cars in the micro cloud, 70% also have the computing power $\mu_C^{(1)}$ while the remainder of the cars have the computing power $\mu_C^{(2)}$, where $\mu_C^{(2)} > \mu_C^{(1)}$. The computation power $\mu_C^{(2)}$ corresponds to high-end vehicles that have abundant computation power compared to regular vehicles. Finally, the computing power of the edge server is $\mu_C^{(3)}$. We assume that all nodes report their computing power to the node in charge of performing resource assignment.

We assume that there is one TT per sender per each optimization period and that all of the TTs have the parameters d_k, c_k and τ_k in common. We obtain a reference for the values of these parameters from some reported traces of computer vision computing tasks [KRS17]. The simulations are performed for two generic types of computing tasks, one representing image processing tasks such as edge detection, and one representing radar point cloud processing tasks such as a SLAM operation. The point cloud type has a higher computational load c_k than the image type but a lower data size d_k. The maximum latency τ_k for sensor frame processing can not be too high as the positions of the vehicles on the road change rapidly. The values of parameters used in simulations are given in Tables I and II.

As a benchmark for our resource assignment algorithm, we use random resource assign-

Table 5.1: Computing and communications resources and parameters used in simulations

Parameter	Value	Parameter	Value
T	1 s	η_S	0.2
$[\mu_C^{(1)}, \mu_C^{(2)}, \mu_C^{(3)}]$	[1 5 10] GHz	η_R	0.2
μ_R	50 Mb/s	U^{V2V}	∞
σ_R	5 Mb/s	U^{LTE}	24 Mb/s

Table 5.2: Parameter of image and point cloud processing tasks used in simulations

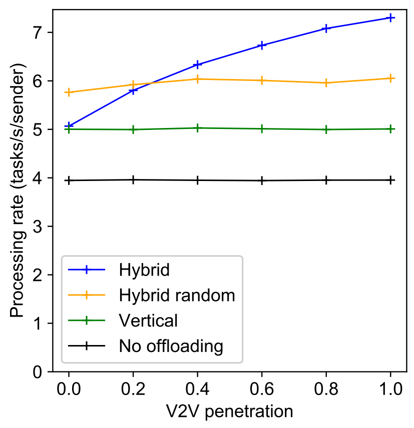
Parameter	c_k	d_k	τ_k
Image	1E9	20 KB	0.6 s
Point cloud	2E8	400 KB	0.6 s

ment. With random resource assignment, all senders are assigned to process their own TTs and each of the remaining workers and the edge servers are randomly assigned to a TT at maximum capacity. The transmission resources $Y_{s_k,k,w}$ at each sender are equally split across the workers that process its TTs. Naturally, with this random assignment approach we cannot guarantee that the constraints (C1-C5) will be satisfied.

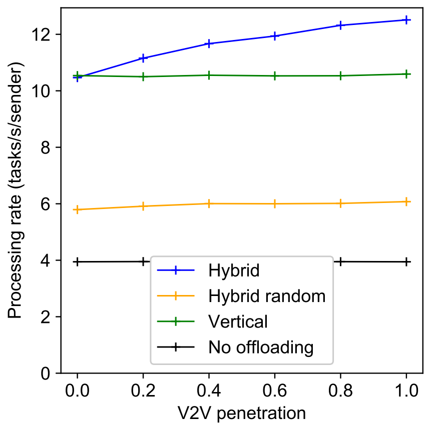
5.4.1 Image processing

We first analyze and compare the offloading approaches for image processing tasks. The performance metric is the number of tasks per second per sender that are processed within the allowed latency τ_k. The results are shown in Fig. 5.2. with a limit $U^{\text{LTE}} = 24$ MB/s on the amount of LTE transmissions and without any limit on LTE traffic.

The LTE uplink limit can restrict the number of tasks that can be offloaded to the edge over a period T. The results in Fig. 5.2a demonstrate what the performance may look like if the uplink limit becomes the bottleneck. Vertical offloading does not provide a significant benefit in terms of the processing rate. However, horizontal offloading with the help of vertical offloading (*hybrid offloading*) can still double the processing rate compared to the



(a) With LTE cap



(b) Without LTE cap

Figure 5.2: The processing rate of image tasks.

case of no offloading. The number of tasks processed by horizontal offloading is unaffected by the LTE cap but it does depend on the V2V penetration defined as the share of cars other than senders and receivers that are equipped for V2V communication and hence able to serve as workers. Without an LTE data transmission limit, vertical offloading doubles the processing rate on its own. However, this scenario is unrealistic as it does not recognize the financial cost that can be incurred due to LTE upload.

We observe that with hybrid offloading and random assignment the system resources are underutilized, and the performance only slightly increases with higher penetration. When tasks are offloaded to randomly selected vehicles, the communication rate to the selected vehicles may not always be sufficient to deliver the frames fast enough and so less frames are successfully processed. However, we should note that this is not a one-to-one comparison to our algorithm since with random assignment the uplink limit constraint is not always satisfied, which is why with no V2V penetration the random assignment seemingly performs better in Fig. 2a.

We also analyze how offloading affects the average delay of the processing of the frames. The results are shown in Fig. 5.3 for the case with an LTE traffic limit. The delay consists of transmission delay to the processor, if a task is offloaded, and the compute delay at the

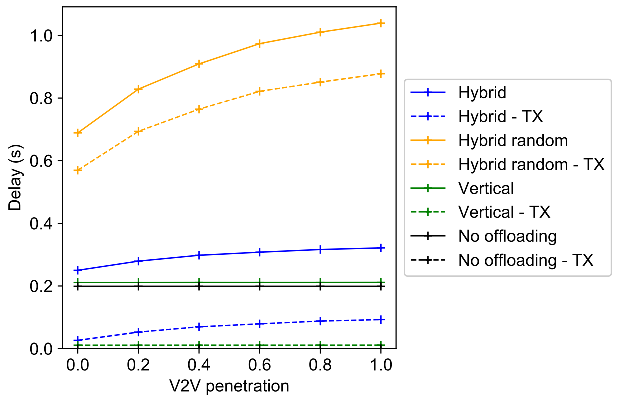


Figure 5.3: The average processing delay, which consists of transmit and computing delay, for image tasks. The transmit delay is represented by the dotted line.

processor. We assume that the transmission delay of the output results is negligible. The average transmission delay of hybrid offloading is higher than that of vertical offloading because a DSRC connection, which is utilized for horizontal offloading, has a lower rate than an LTE connection. The transmission delay increases with V2V penetration since larger share of tasks is offloaded horizontally therefore the average transmission delay is larger. The average compute delay (the transmission delay subtracted from the total delay) moderately decreases with V2V penetration since some tasks are being offloaded to the high-end vehicles. Overall, offloading image tasks is only suitable in the scenarios where it is acceptable to incur an additional processing delay in order to achieve a higher rate. With random assignment, the transmission delay is very significant because data frames spend extensive amount of time in the transmission queue since the transmission rate cannot satisfy the scheduled rate.

5.4.2 Point cloud processing

The processing rate for point cloud tasks is shown in Fig. 5.4. Since the data size of point cloud frames is small, the LTE traffic is already below the cap that we set, and it does not have an impact on the processing rate. Vertical offloading increases the processing rate by around 100% in our scenario. Since the transmission overhead is smaller than that of

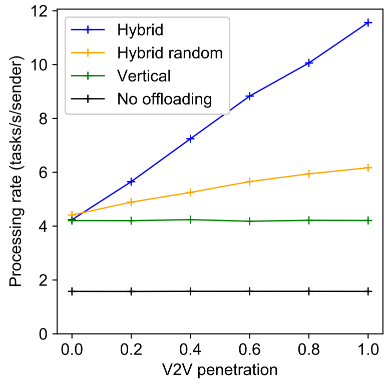


Figure 5.4: The processing rate of point cloud tasks.

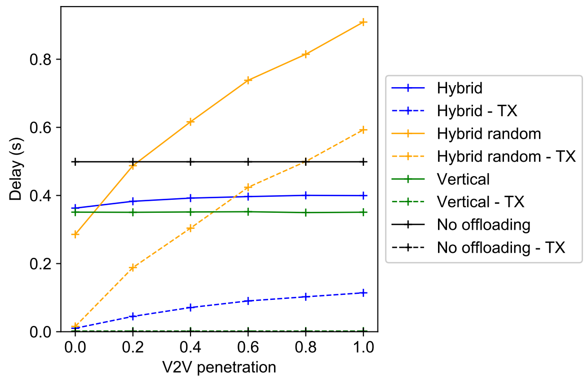


Figure 5.5: The average processing delay, which consists of transmit and computing delay, for point cloud tasks.

image tasks, large gains are possible thanks to horizontal offloading and, overall, multiple-fold gain is obtained with hybrid offloading. Point cloud tasks lend themselves much better to offloading compared to image tasks. This is further supported by the delay profiles shown in Fig. 5.5. Since the compute load of point cloud tasks is significantly larger than that of the image tasks, their processing delay can be significantly reduced by offloading them to more powerful processors. Indeed, we observe a significant decrease in the processing delay with both vertical and hybrid offloading in Fig. 5.5. due to offloading to the edge server and to the powerful high-end cars. Overall, offloading of point cloud tasks increases the processing rate while decreasing the processing delay.

5.5 Summary

In vehicular computing-intensive applications, the task processing rate can be increased by offloading the computing to the local edge servers (vertical offloading) and to the nearby cars (horizontal offloading). We develop an optimized resource assignment and scheduling algorithm for hybrid offloading of computing tasks for cooperative perception that maximizes the rate at which frames are processed, while ensuring that results are delivered within a deadline and also constraining the cellular and V2V communication overhead. The algorithm is tested in a simulated environment based on the LUST traffic scenario and Veins vehicular network simulator. We observe a significant increase in the processing rate of sensor frames when using hybrid offloading compared to the no offloading case or the case with only vertical offloading.

CHAPTER 6

Conclusion

6.1 Summary of Contributions

In this dissertation, we developed cooperative systems and methods that can bring higher communications speeds and larger computing power to mobile devices without relying on expansion of network infrastructure.

In Chapter 2, we developed methods for prediction of channel gain that use environment-specific features, including building maps and channel gain measurements to achieve a high level of prediction accuracy. We assume that measurements are collected using a swarm of coordinated UAVs. We developed two active prediction approaches based on deep learning and Kriging interpolation. We trained and evaluated the two proposed approaches in a ray-tracing-based channel gain simulator. Using channel simulations based on the ray-tracing approach, we demonstrated the importance of active prediction compared to prediction based on randomly collected measurements of channel gain. Furthermore, we showed that using deep learning and 3D maps, we can achieve high prediction accuracy compared to the benchmarks even without knowing the transmitter location. We also demonstrated the importance of coordinated path planning for active prediction when using multiples UAVs compared to UAVs collecting measurements independently in a greedy manner.

In Chapter 3, we used deep reinforcement learning methods to optimize the placement of a UAV communicating to the user on the ground. We consider the case where the ground user location is not known and use topology data to replace statistical models of the channel.

We were able to achieve a high success rate in moving the UAV to a location that has a sufficient SINR with limits on UAV traversal distance and starting from a randomized location. Moreover, our reinforcement learning approach stands out in that it can be applied in previously unexplored urban environment.

In Chapter 4, we considered a cooperative computing paradigm between intelligent vehicles of similar computing power to enable emerging vehicular applications. To this end, we developed resource assignment methods that will maximize the quality of service for computing applications given the available vehicles in the micro cloud. Our proposed approaches adapt to link quality changes between vehicles and prevent congestion on 802.11 channels even in the presence of incumbent interference. Furthermore, the approaches adapt to mobility of vehicles to control the frequency of resource assignment updates. The proposed approaches are evaluated in realistic simulators of vehicular networking based on IEEE 802.11p standard. The simulation results demonstrate the importance of accurate congestion prediction for reliable cooperative computing. This is most notable in presence of incumbent devices. We further validate the performance of our methods in different traffic scenarios and demonstrate consistent gains in quality of service compared to non-cooperative computing. Next, we evaluated two practical methods for prediction of SNR between vehicles in terms of their viability for use in cooperative computing. Using these SNR prediction methods we show how our approach can be extended to adaptively control the resource assignment update frequency.

In Chapter 5, we showed how for vehicular computing-intensive applications, the task processing rate can be increased by offloading the computing to the local edge servers (vertical offloading) and to the nearby cars (horizontal offloading). We develop an optimized resource assignment and scheduling algorithm for hybrid offloading of computing tasks for cooperative perception that maximizes the rate at which frames are processed. The algorithm is tested in a simulated environment based on the LUST traffic scenario and Veins vehicular network simulator. We quantified the increases in the processing rate of sensor frames when using hybrid offloading compared to the case with no vertical offloading or the case with only vertical offloading.

6.2 Future Work

In Chapter 2, we assumed that the location of the ground user is completely unknown for our deep learning based approaches, however it would also be practical to consider scenarios where the ground device is partially localized. The location of the ground user and its uncertainty could be incorporated as an input into the deep learning channel gain prediction algorithm. Prediction and measurement collection in our deep learning approaches was constrained to a 2D plane for simplicity, however a 3D proof of concept of these approaches would be valuable. Furthermore, future studies could focus on experimental validation of the developed active prediction approaches.

In Chapter 3, we focused on the scenarios where the ground user is not moving over the course of UAV exploration and placement optimization. To capture the mobility of ground users, recurrent neural networks can be utilized, which can use sequential measurements to implicitly predict future changes in the channel gain across space. To accomplish this, recurrent neural networks can be used as building blocks for deep reinforcement learning models. Furthermore, in this chapter we assumed that every UAV is assigned to serve a single user. Future work should explore how deep reinforcement learning techniques can be used to simultaneously optimize the locations of multiple UAVs serving multiple users on the ground.

In Chapter 4, the utility in cooperative computing was defined as either the task processing rate allocated to a task stream or the computational complexity at which tasks in the task stream are scheduled to be processed. However, alternative utility metrics are also relevant, such as task processing delay or energy consumption. The 802.11 PHY layer model used in our approach can be updated to incorporate newer advances in this technology, such as multiple-input and multiple-output (MIMO) communications. These advances can potentially improve the performance of vehicular cooperative computing frameworks. Similar extensions could be applied to our work in Chapter 5. Furthermore, the methods in Chapter 5 can be improved by utilizing the approaches for congestion control in vehicular networks developed in Chapter 4.

REFERENCES

[AAG11] Marko Angjelicinoski, Vladimir Atanasovski, and Liljana Gavrilovska. "Comparative analysis of spatial interpolation methods for creating radio environment maps." In *2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers*, pp. 334–337. IEEE, 2011.

[AFG10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. "A view of cloud computing." *Communications of the ACM*, **53**(4):50–58, 2010.

[AMB22] Emmanouil Alimpertis, Athina Markopoulou, Carter T Butts, Evita Bakopoulou, and Konstantinos Psounis. "A Unified Prediction Framework for Signal Maps: Not All Measurements are Created Equal." *IEEE Transactions on Mobile Computing*, 2022.

[ARV12] Andreas Achtzehn, Janne Riihijärvi, Guilberth Martínez Vargas, Marina Petrova, and Petri Mähönen. "Improving coverage prediction for primary multi-transmitter networks operating in the TV whitespaces." In *2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 623–631. IEEE, 2012.

[BBE11] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. "SUMO–simulation of urban mobility: an overview." In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[BBG21] Tayebeh Bahreini, Marco Brocanelli, and Daniel Grosu. "VECMAN: A framework for energy-aware resource management in vehicular edge computing systems." *IEEE Transactions on Mobile Computing*, 2021.

[BDM17] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." In *International conference on machine learning*, pp. 449–458. PMLR, 2017.

[BJF16] Hajer Braham, Sana Ben Jemaa, Gersende Fort, Eric Moulines, and Berna Sayrac. "Spatial prediction under location uncertainty in cellular networks." *IEEE Transactions on Wireless Communications*, **15**(11):7633–7643, 2016.

[CBS18] Vinay-Prasad Chowdappa, Carmen Botella, Juan Javier Samper-Zapater, and Rafael J Martinez. "Distributed radio map reconstruction for 5G automotive." *IEEE Intelligent Transportation Systems Magazine*, **10**(2):36–49, 2018.

[CCM23] Yi Chen, Zheng Chang, Geyong Min, Shiwen Mao, and Timo Hämäläinen. "Joint Optimization of Sensing and Computation for Status Update in Mobile Edge Computing Systems." *IEEE Transactions on Wireless Communications*, 2023.

[CEG19] Md Moin Uddin Chowdhury, Fatih Erden, and Ismail Guvenc. "RSS-Based Q-Learning for Indoor UAV Navigation." *arXiv preprint arXiv:1905.13406*, 2019.

[CFE15] Lara Codeca, Raphaël Frank, and Thomas Engel. "Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research." In *2015 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8. IEEE, 2015.

[CG17] Junting Chen and David Gesbert. "Optimal positioning of flying relays for wireless networks: A LOS map approach." In *2017 IEEE international conference on communications (ICC)*, pp. 1–6. IEEE, 2017.

[CJL15] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. "Efficient multi-user computation offloading for mobile-edge cloud computing." *IEEE/ACM transactions on networking*, **24**(5):2795–2808, 2015.

[COV22] Antoine Caillot, Safa Ouerghi, Pascal Vasseur, Rémi Boutteau, and Yohan Dupuis. "Survey on Cooperative Perception in an Automotive Context." *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[CTY19] Qi Chen, Sihai Tang, Qing Yang, and Song Fu. "Cooper: Cooperative perception for connected autonomous vehicles based on 3d point clouds." In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 514–524. IEEE, 2019.

[CWY22] Guanhua Chai, Weihua Wu, Qinghai Yang, Runzi Liu, and F Richard Yu. "Learning-Based Resource Allocation for Ultra-Reliable V2X Networks With Partial CSI." *IEEE Transactions on Communications*, **70**(10):6532–6546, 2022.

[CZX22] Guangzhen Cui, Weili Zhang, Yanqiu Xiao, Lei Yao, and Zhanpeng Fang. "Cooperative perception technology of autonomous driving in the internet of vehicles environment: A review." *Sensors*, **22**(15):5535, 2022.

[EGG18a] Omid Esrafilian, Rajeev Gangula, and David Gesbert. "Learning to communicate in UAV-aided wireless networks: Map-based approaches." *IEEE Internet of Things Journal*, **6**(2):1791–1802, 2018.

[EGG18b] Omid Esrafilian, Rajeev Gangula, and David Gesbert. "UAV-relay placement with unknown user locations and channel parameters." In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 1075–1079. IEEE, 2018.

[ES12] David Eckhoff and Christoph Sommer. "A multi-channel IEEE 1609.4 and 802.11 p EDCA model for the veins framework." In *Proceedings of 5th ACM/ICST international conference on simulation tools and techniques for communications, networks and systems: 5th ACM/ICST international workshop on OMNet++.(Desenzano, Italy, 19-23 March, 2012). OMNeT*, 2012.

[FLW17] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. "AVE: Autonomous vehicular edge computing framework with ACO-based scheduling." *IEEE Transactions on Vehicular Technology*, **66**(12):10660–10675, 2017.

[Foe18] J Foerster. *Deep multi-agent reinforcement learning*. PhD thesis, University of Oxford, 2018.

[Gud91] Mikael Gudmundson. "Correlation model for shadow fading in mobile radio systems." *Electronics letters*, **27**(23):2145–2146, 1991.

[HB12] Gustavo Hernandez-Penaloza and Baltasar Beferull-Lozano. "Field estimation in wireless sensor networks using distributed kriging." In *2012 IEEE International Conference on Communications (ICC)*, pp. 724–729. IEEE, 2012.

[HJD17] Takamasa Higuchi, Joshua Joy, Falko Dressler, Mario Gerla, and Onur Altintas. "On the feasibility of vehicular micro clouds." In *2017 IEEE Vehicular Networking Conference (VNC)*, pp. 179–182. IEEE, 2017.

[HSH17] Florian Hagenauer, Christoph Sommer, Takamasa Higuchi, Onur Altintas, and Falko Dressler. "Vehicular micro clouds as virtual edge servers for efficient data collection." In *Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services*, pp. 31–35, 2017.

[HUH19] Ghaith Hattab, Seyhan Ucar, Takamasa Higuchi, Onur Altintas, Falko Dressler, and Danijela Cabric. "Optimized Assignment of Computational Tasks in Vehicular Micro Clouds." In *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, pp. 1–6, 2019.

[HXS20] Xu Han, Lei Xue, Fucai Shao, and Ying Xu. "A power spectrum maps estimation algorithm based on generative adversarial networks for underlay cognitive radio networks." *Sensors*, **20**(1):311, 2020.

[Ins22] Fortune Business Insights. "Commercial Drone Market Size, Share | Forecast Report [2028]." *Fortune Business Insights*, 2022.

[JCD08] Daniel Jiang, Qi Chen, and Luca Delgrossi. "Optimal data rate selection for vehicle safety communications." In *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking*, pp. 30–38, 2008.

[JZC12] Y. Jin, Y. D. Zhang, and B. K. Chalise. “Joint optimization of relay position and power allocation in cooperative broadcast wireless networks.” In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2493–2496, March 2012.

[KA12] Özlem Karsu and Meral Azizoğlu. “The multi-resource agent bottleneck generalised assignment problem.” *International Journal of Production Research*, **50**(2):309–324, 2012.

[KB14] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization.” *arXiv preprint arXiv:1412.6980*, 2014.

[KC23] Enes Krijestorac and Danijela Cabric. “Deep Learning Based Active Spatial Channel Gain Prediction Using a Swarm of Unmanned Aerial Vehicles.” *to be submitted to a journal, arXiv preprint arXiv:2310.04547*, 2023.

[KCZ18] Aziz Altaf Khuwaja, Yunfei Chen, Nan Zhao, Mohamed-Slim Alouini, and Paul Dobbins. “A survey of channel modeling for UAV communications.” *IEEE Communications Surveys & Tutorials*, **20**(4):2804–2821, 2018.

[KHC19] Enes Krijestorac, Samer Hanna, and Danijela Cabric. “UAV access point placement for connectivity to a user with unknown location using deep RL.” In *2019 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6. IEEE, 2019.

[KHC21] Enes Krijestorac, Samer Hanna, and Danijela Cabric. “Spatial signal strength prediction using 3D maps and deep learning.” In *ICC 2021-IEEE international conference on communications*, pp. 1–6. IEEE, 2021.

[KLH23] Enes Krijestorac, Chunghan Lee, Takamasa Higuchi, Seyhan Ucar, Onur Altintas, and Danijela Cabric. “Cooperative Computing in Vehicular Micro Clouds for Emerging Vehicular Applications.” *to be submitted to a journal*, 2023.

[KMH20] Enes Krijestorac, Agon Memedi, Takamasa Higuchi, Seyhan Ucar, Onur Altintas, and Danijela Cabric. “Hybrid vehicular and cloud distributed computing: A case for cooperative perception.” In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6. IEEE, 2020.

[KRS17] Ajay Kattepur, Hemant Kumar Rath, and Anantha Simha. “A-priori estimation of computation times in fog networked robotics.” In *2017 IEEE international conference on edge computing (EDGE)*, pp. 9–16. IEEE, 2017.

[KSG08] Andreas Krause, Ajit Singh, and Carlos Guestrin. “Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies.” *Journal of Machine Learning Research*, **9**(2), 2008.

[KSS23] Enes Krijestorac, Hazem Sallouha, Shamik Sarkar, and Danijela Cabric. "Agile Radio Map Prediction Using Deep Learning." In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–2. IEEE, 2023.

[LaV06] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[LBB18] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. "Microscopic traffic simulation using sumo." In *2018 21st international conference on intelligent transportation systems (ITSC)*, pp. 2575–2582. IEEE, 2018.

[LC17] Yue Li and Lin Cai. "UAV-Assisted Dynamic Coverage in a Heterogeneous Cellular System." *IEEE Network*, **31**(4):56–61, 2017.

[LDK11] Kian Hsiang Low, John M Dolan, and Pradeep Khosla. "Active Markov information-theoretic path planning for robotic environmental sensing." In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 753–760, 2011.

[LKG17] Donghoon Lee, Seung-Jun Kim, and Georgios B Giannakis. "Channel gain cartography for cognitive radios leveraging low rank and sparsity." *IEEE Transactions on Wireless Communications*, **16**(9):5953–5966, 2017.

[LLC18] Xiao Liu, Yuanwei Liu, and Yue Chen. "Deployment and Movement for Multiple Aerial Base Stations by Reinforcement Learning." In *2018 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6. IEEE, 2018.

[LLW23] Li Li, Wei Li, Jun Wang, Xiaonan Chen, Qihang Peng, and Wei Huang. "UAV Trajectory Optimization for Spectrum Cartography: A PPO Approach." *IEEE Communications Letters*, 2023.

[LOC16] Pawel Ladosz, Hyondong Oh, and Wen-Hua Chen. "Optimal positioning of communication relay unmanned aerial vehicles in urban environments." In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1140–1147. IEEE, 2016.

[LYK21] Ron Levie, Çağkan Yapar, Gitta Kutyniok, and Giuseppe Caire. "RadioUNet: Fast radio map estimation with convolutional neural networks." *IEEE Transactions on Wireless Communications*, **20**(6):4001–4015, 2021.

[MDW12] Lusheng Miao, Karim Djouani, Barend J van Wyk, and Yskandar Hamam. "Evaluation and enhancement of IEEE 802.11 p standard: a survey." *Mobile Computing*, **1**(1):15–30, 2012.

[Mil03] Len E Miller. "Validation of 802.11 a/UWB coexistence simulation." *National Institute of Standards and Technology (NIST), WCTG white paper*, 2003.

[MKS13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602*, 2013.

[MM12] Mehrzad Malmirchegini and Yasamin Mostofi. "On the spatial predictability of communication channels." *IEEE Transactions on Wireless Communications*, **11**(3):964–978, 2012.

[MM17] Arjun Muralidharan and Yasamin Mostofi. "Path planning for a connectivity seeking robot." In *2017 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6. IEEE, 2017.

[MSB16] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, and Mérouane Debbah. "Efficient Deployment of Multiple Unmanned Aerial Vehicles for Optimal Wireless Coverage." *IEEE Communications Letters*, **20**(8):1647–1650, 2016.

[MSB19a] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, Young-Han Nam, and Mérouane Debbah. "A tutorial on UAVs for wireless networks: Applications, challenges, and open problems." *IEEE communications surveys & tutorials*, **21**(3):2334–2360, 2019.

[MSB19b] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, Young-Han Nam, and Mérouane Debbah. "A Tutorial on UAVs for Wireless Networks: Applications, Challenges, and Open Problems." *IEEE Communications Surveys Tutorials*, **21**(3):2334–2360, 2019. Conference Name: IEEE Communications Surveys Tutorials.

[NZW20] Zhaolong Ning, Kaiyuan Zhang, Xiaojie Wang, Mohammad S Obaidat, Lei Guo, Xiping Hu, Bin Hu, Yi Guo, Balqies Sadoun, and Ricky YK Kwok. "Joint computing and caching in 5G-envisioned Internet of vehicles: A deep reinforcement learning-based traffic control system." *IEEE Transactions on Intelligent Transportation Systems*, **22**(8):5201–5212, 2020.

[QAB18] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. "Avr: Augmented vehicular reality." In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 81–95, 2018.

[QHC23] Bo-Jun Qiu, Cheng-Ying Hsieh, Jyh-Cheng Chen, and Falko Dressler. "TCOA: Triple-Check Offloading Algorithm for Roadside Units and Vehicular Microclouds in 5G Networks and Beyond." *IEEE Access*, 2023.

[Ras03] Carl Edward Rasmussen. "Gaussian processes in machine learning." In *Summer school on machine learning*, pp. 63–71. Springer, 2003.

[rem] "Wireless EM Propagation Software - Wireless InSite." URL: https://www.remcom.com/wireless-insite-em-propagation-software.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

[SCZ16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge computing: Vision and challenges." *IEEE internet of things journal*, **3**(5):637–646, 2016.

[SD11] Christoph Sommer and Falko Dressler. "Using the right two-ray model? A measurement based evaluation of PHY models in VANETs." In *Proc. ACM MobiCom*, pp. 1–3, 2011.

[SEB19] Christoph Sommer, David Eckhoff, Alexander Brummer, Dominik S Buse, Florian Hagenauer, Stefan Joerer, and Michele Segata. "Veins: The open source vehicular network simulation framework." In *Recent advances in network simulation*, pp. 215–252. Springer, 2019.

[SEG11] Christoph Sommer, David Eckhoff, Reinhard German, and Falko Dressler. "A computationally inexpensive empirical model of IEEE 802.11 p radio shadowing in urban environments." In *2011 Eighth international conference on wireless on-demand network systems and services*, pp. 84–90. IEEE, 2011.

[SGD10] Christoph Sommer, Reinhard German, and Falko Dressler. "Bidirectionally coupled network and road traffic simulation for improved IVC analysis." *IEEE Transactions on mobile computing*, **10**(1):3–15, 2010.

[SGD11] Christoph Sommer, Reinhard German, and Falko Dressler. "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis." *IEEE Transactions on Mobile Computing*, **10**(1):3–15, January 2011.

[SGS19] Yuxuan Sun, Xueying Guo, Jinhui Song, Sheng Zhou, Zhiyuan Jiang, Xin Liu, and Zhisheng Niu. "Adaptive learning-based task offloading for vehicular edge computing systems." *IEEE Transactions on vehicular technology*, **68**(4):3061–3074, 2019.

[SJS14] Christoph Sommer, Stefan Joerer, Michele Segata, Ozan K Tonguz, Renato Lo Cigno, and Falko Dressler. "How shadowing hurts vehicular communications and how dynamic beaconing can help." *IEEE Transactions on Mobile Computing*, **14**(7):1411–1421, 2014.

[SLT21] Bodong Shang, Lingjia Liu, and Zhi Tian. "Deep Learning-Assisted Energy-Efficient Task Offloading in Vehicular Edge Computing Systems." *IEEE Transactions on Vehicular Technology*, **70**(9):9619–9624, 2021.

[SRC22] Raju Shrestha, Daniel Romero, and Sundeep Prabhakar Chepuri. "Spectrum surveying: Active radio map estimation with autonomous UAVs." *IEEE Transactions on Wireless Communications*, **22**(1):627–641, 2022.

[TR20] Yves Teganya and Daniel Romero. "Deep Completion Autoencoders for Radio Map Estimation." *arXiv preprint arXiv:2005.05964*, 2020.

[VBT15] Wantanee Viriyasitavat, Mate Boban, Hsin-Mu Tsai, and Athanasios Vasilakos. "Vehicular communications: Survey and challenges of channel and propagation models." *IEEE Vehicular Technology Magazine*, **10**(2):55–66, 2015.

[VGS16] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-learning." In *Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016.

[WLZ22] Zhiwei Wei, Bing Li, Rongqing Zhang, Xiang Cheng, and Liuqing Yang. "OCVC: An overlapping-enabled cooperative vehicular fog computing protocol." *IEEE Transactions on Mobile Computing*, 2022.

[Wol20] Laurence A Wolsey. *Integer programming*, chapter Branch and Bound. John Wiley & Sons, 2020.

[WRC18] Haichao Wang, Guochun Ren, Jin Chen, Guoru Ding, and Yijun Yang. "Unmanned aerial vehicle-aided communications: Joint transmit power and trajectory optimization." *IEEE Wireless Communications Letters*, **7**(4):522–525, 2018.

[WSH15] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling network architectures for deep reinforcement learning." *arXiv preprint arXiv:1511.06581*, 2015.

[WXC07] Junfang Wang, Bin Xie, Kan Cai, and Dharma P Agrawal. "Efficient mesh router placement in wireless mesh networks." In *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, pp. 1–9. IEEE, 2007.

[WXZ18] Qingqing Wu, Jie Xu, and Rui Zhang. "UAV-enabled aerial base station (BS) III/III: Capacity characterization of UAV-enabled two-user broadcast channel." *arXiv preprint arXiv:1801.00443*, 2018.

[YI15] Zhengqing Yun and Magdy F Iskander. "Ray tracing for radio propagation modeling: Principles and applications." *IEEE access*, **3**:1089–1100, 2015.

[YK07] Yaling Yang and Robin Kravets. “Throughput guarantees for multi-priority traffic in ad hoc networks.” *Ad Hoc Networks*, **5**(2):228–253, 2007.

[ZCM19] Chao Zhu, Yi-Han Chiang, Abbas Mehrabi, Yu Xiao, Antti Ylä-Jääski, and Yusheng Ji. “Chameleon: Latency and resolution aware task offloading for visual-based assisted driving.” *IEEE Transactions on Vehicular Technology*, **68**(9):9038–9048, 2019.

[ZFW20] Guoyong Zhang, Xiao Fu, Jun Wang, Xi-Le Zhao, and Mingyi Hong. “Spectrum cartography via coupled block-term tensor decomposition.” *IEEE Transactions on Signal Processing*, **68**:3660–3675, 2020.

[ZLG19] Junhui Zhao, Qiuping Li, Yi Gong, and Ke Zhang. “Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks.” *IEEE Transactions on Vehicular Technology*, **68**(8):7944–7956, 2019.

[ZPX18] Chao Zhu, Giancarlo Pastor, Yu Xiao, Yong Li, and Antti Ylae-Jaeaeski. “Fog following me: Latency and quality balanced task allocation in vehicular fog computing.” In *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–9. IEEE, 2018.

[ZTW19] Jianshan Zhou, Daxin Tian, Yunpeng Wang, Zhengguo Sheng, Xuting Duan, and Victor CM Leung. “Reliability-optimal cooperative communication and computing in connected vehicle systems.” *IEEE Transactions on Mobile Computing*, **19**(5):1216–1232, 2019.

[ZZL16] Y. Zeng, R. Zhang, and T. J. Lim. “Throughput Maximization for UAV-Enabled Mobile Relaying Systems.” *IEEE Transactions on Communications*, **64**(12):4983–4996, December 2016.