EFFICIENT MOBILE SENSING FOR LARGE-SCALE SPATIAL DATA ACQUISITION

EFFICIENT MOBILE SENSING FOR LARGE-SCALE SPATIAL DATA ACQUISITION

BY

YONGYONG WEI, M.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Doctor of Philosophy (2021) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Efficient Mobile Sensing for Large-Scale Spatial Data Acquisition
AUTHOR:	Yongyong Wei M.Sc. (Computer Science), University of Chinese Academy of Sciences, Beijing, China
SUPERVISOR:	Dr. Rong Zheng
NUMBER OF PAGES:	xxv, 153

Lay Abstract

A variety of applications such as environmental monitoring require to collect large-scale spatial data like air quality, temperature and humidity. However, it usually incurs dramatic costs like time to obtain those data, which is impeding the deployment of those applications. To reduce the data collection efforts, we consider two mobile sensing schemes, i.e, mobile robotic sensing and mobile crowdsourcing. For the former scheme, we investigate how to plan paths for mobile robots given limited travel budgets. For the latter scheme, we design a crowdsourcing platform and study user behavior through a real word data collection campaign. The proposed solutions in this thesis can benefit large-scale spatial data collection tasks.

Abstract

Large-scale spatial data such as air quality of a city, biomass content in a lake, Wi-Fi Received Signal Strengths (RSS, also referred as fingerprints) in indoor spaces often play vital roles to applications like indoor localization. However, it is extremely labor-intensive and time-consuming to collect those data manually. In this thesis, the main goal is to develop efficient means for large-scale spatial data collection.

Robotic technologies nowadays offer an opportunity on mobile sensing, where data are collected by a robot traveling in target areas. However, since robots usually have a limited travel budget depending on battery capacity, one important problem is to schedule a data collection path to best utilize the budget. Inspired by existing literature, we consider to collect data along informative paths. The process to search the most informative path given a limited budget is known as the informative path planning (IPP) problem, which is NP-hard. Thus, we propose two heuristic approaches, namely a greedy algorithm and a genetic algorithm. Experiments on Wi-Fi RSS based localization show that data collected along informative paths tend to achieve lower errors than that are opportunistically collected.

In practice, the budget of a mobile robot can vary due to insufficient charging or battery degradation. Although it is possible to apply the same path planning algorithm repetitively whenever the budget changes, it is more efficient and desirable to

avoid solving the problem from scratch. This can be possible since informative paths for the same area share common characteristics. Based on this intuition, we propose and design a reinforcement learning based IPP solution, which is able to predict informative paths given any budget. In addition, it is common to have multiple robots to conduct sensing tasks cooperatively. Therefore, we also investigate the multi-robot IPP problem and present two solutions based on multi-agent reinforcement learning.

Mobile crowdsourcing (MCS) offers another opportunity to lowering the cost of data collection. In MCS, data are collected by individual contributors, which is able to accumulate a large amount of data when there are sufficient participants. As an example, we consider the collection of a specific type of spatial data, namely Wi-Fi RSS, for indoor localization purpose. The process to collect RSS is also known as site survey in the localization community. Though MCS based site survey has been suggested a decade ago [80], so far, there has not been any published large-scale fingerprint MCS campaign. The main issue is that it depends on user's participation, and users may be reluctant to make a contribution. To investigate user behavior in a real-world site survey, we design an indoor fingerprint MCS system and organize a data collection campaign in the McMaster University campus for five months. Although we focus on Wi-Fi fingerprints, the design choices and campaign experience are beneficial to the MCS of other types of spatial data as well.

The contribution of this thesis is two-fold. For applications where robots are available for large-scale spatial sensing, efficient path planning solutions are investigated so as to maximize data utility. Meanwhile, for MCS based data acquisition, our real-world campaign experience and user behavior study reveal essential design factors that need to be considered and aspects for further improvements.

To my family

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor Dr. Rong Zheng, who gave me tremendous support and guidance during my PhD study. I believe I am fortunate to have the opportunity to join the Wireless System Research Group (WiSeR) led by her. Dr. Zheng is knowledgeable and smart. Whenever I met challenges and got stuck in my research, she could always give me some valuable suggestions and encourage me to think and dig deeper. Besides research skills, Dr. Zheng also creates opportunities to enhance presentation and writing skills for WiSeR members. I learned a lot through the weekly meetings, the presentations in WiSeR group seminars and will never forget the careful and insightful feedback she gave to me when revising my papers.

Second, I would like to thank my supervisory committee members Dr. Fei Chiang and Dr. Hassan Ashtiani. I am grateful for their insightful suggestions and comments during my supervisory meetings. Dr. Chiang's feedback on the crowdsourcing project is valuable for me to enhance and present it. Dr. Ashtiani is an expert in machine learning and his constructive suggestions inspired me a lot on the path planning and reinforcement learning project. I would also like to all the folks in the WiSeR group. The group activities of boating, bowling and hiking are excellent stress relievers with lots of fun.

Last but not least, I want to thank my family. In particular, my wife has stood by me in the past four years. Without her unconditional supports and encouragements, this dissertation is not possible. I appreciate all she has done so I can focus on my research. I would also like to thank my son, for his laughter that makes my life brighter.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Abbreviations

GP	Gaussian Process
GPS	Global Positionning System
IPS	Indoor Positionnining System
RSS	Received Signal Strength
AP	Access Point
FP	Fingerprint
MCS	Mobile Crowdsourcing
POI	Point of Interest
IMU	Inertial Measurment Unit
IPP	Informative Path Planning

MIPP	Multi-robot Informative Path Planning
MI	Mutual Information
RG	Recursive Greedy algorithm
GA	Genetic Algorithm
TSP	Travelling Salesman Problem
RL	Reinforcment Learning
MDP	Markov Decision Process
MMDP	Multi-agent Markov Decision Process
RL	Reinforcement Learning
MARL	Multi-agent Reinforcement Learning
MAS	Multi-agent System
OP	Orienteering Problem
MBCR	Marginal Benefit-Cost Ratio
STSP	Steiner Travelling Salesman Problem
RO	Random Orienteering
ERO	Edge based Random Orienteering
SLAM	Simultanenous Localization and Mapping
RNN	Recurrent Neural Network

LCP Least Cost Path

JAL Joint Action Learning

IQL Independent Q-learning

Chapter 1

Introduction

1.1 Motivation and Background

A variety of applications rely on the acquisition of large-scale spatial data such as air quality or humidity in urban areas. To collect these data, researchers have proposed to deploy Wireless Sensor Networks (WSNs) in the target areas, where the data are collected by the sensors and transmitted to a centralized server. Although WSNs are ideal for long-term and real-time environmental monitoring, due to their huge infrastructure costs and setup overhead, they are not suitable for applications that only need to collect data infrequently or on demand. One such example is Wi-Fi based indoor localization [5]. Given a wireless access point (AP), the received signal strengths at a fixed location tend to follow a stable distribution if there are no environmental changes like moving pedestrians or new obstacles. In [48], the authors observe that the distribution of the RSS is often left-skewed Gaussian. Therefore, Wi-Fi RSS has been widely investigated as location dependent features (which is also known as Wi-Fi fingerprints) to train localization models. In general, the Wi-Fi RSS

data only need to be collected infrequently when there are environmental changes or infrastructure changes like some APs are removed.

However, the lack of low-cost data collection strategies has been impeding relevant applications. For instance, Wi-Fi based localization has been proposed for around two decades, but until now there are no well known successful commercial fingerprint-based localization solutions. According to a rough estimation in [64], it takes around three hours to measure the signal strengths at the corridor of a $69m \times 54m$ floor, even with only one scan at each predefined location.

In general, spatial data in close proximity are highly correlated. Nearby sites tend to yield similar measurements. This relationship can be usually modeled by Gaussian Processes (GPs) with appropriate hyper-parameters. Measurements at sites that are not observed can be predicted through the GP models. Therefore, it is not necessary to collect the data exhaustively, but only data from a representative set of locations are required. As a result, data collection efforts can be reduced if we only consider a subset of locations. Inspired by the sensor placement optimization work in [57], we aim to collect data at locations that are informative. The ultimate goal is that given the measurements at those selected locations, the uncertainty of the prediction is minimized. This can be achieved by optimizing the mutual information (MI) between the variables at observed and un-observed locations. The criteria of MI has been widely used by researchers and practitioners when considering *where* to collect spatial data [8].

Another aspect that is helpful in reducing data collection efforts is *how* to collect the data. With the advancement of technologies, two opportunities arise. First, nowadays, there are many off-the-shelf low-cost mobile robotic platforms available

like autonomous vehicles, drones, and rovers [20]. These technologies make it possible to automate the data collection process by mobile robots, which can directly save human efforts. Second, many smartphones are equipped with a variety of sensors. Thus, another opportunity is to involve individual participants in data collection campaigns, which is known as mobile crowdsourcing. In this scenario, a data collection task is divided among the crowd, which can indirectly address the labor-intensive issue. Although a single participant may only spend limited efforts and contribute a small amount of data, the eventual result will be promising when there are plenty of participants.

In this thesis, both two types of data collection strategies are investigated.

1.1.1 Mobile Robotic Sensing

In mobile robotic sensing, we leverage existing hardware design, navigation and localization technologies from commercial-of-the-shelf platforms and open-source libraries. We mainly focus on planning informative paths for the mobile robots since they are battery powered and have limited travel budgets. When they are moving along the planned paths, two types of data will be collected: environmental measurements and their locations. The obtained data can be utilized to build GP models to characterize the spatial data distribution of target environments.

In order to find informative paths(called Informative Path Planning, or IPP), we first define a graph based on the topology of the target area. This can be achieved by using points of interests as vertices, and an edge exists if two vertices are directly reachable. In the case of an open area, a simple grid graph can be constructed. Compared with optimizing paths directly in a continuous physical space, one major

advantage of graph-based path planning is that paths are confined to vertices and edges on the graph. This makes it easy to rule out obstacles. In essence, the optimization goal is to decide which subset of vertices to visit and in which order to visit them. A related problem is the Orienteering Problem [103] (OP), where each vertex is associated with a reward. In OP, given a start and terminal vertex, one needs to search a path to collect the most reward with a limited travel budget. The main difference between IPP and OP is the definition of rewards. In IPP, rewards are obtained when a robot moves along edges, and they are not additive.

In this thesis, we first consider the IPP problem when the budget is known and fixed. We prove that IPP is NP-hard by reducing an instance of OP to an instance of IPP within polynomial time. Therefore, we propose several heuristic based approaches so as to achieve acceptable trade-offs between computation complexity and optimality.

However, in practice, budget variations are common due to different battery capacities, charging statuses or timing requirements. Although it is feasible to repetitively apply the same algorithm to IPP with fixed budget when budgets change, it is more efficient and desirable to avoid searching from scratch. Intuitively, given the same target area, informative paths with different budgets may share common characteristics. For instances, re-visiting vertices that have been visited tends to achieve less reward, while visiting new vertices tends to yield more reward. A novel reinforcement learning based path planning solution is proposed that can predict informative paths for different budgets.

Lastly, having multi-robots conducting large-scale spatial data acquisition cooperatively can reduce task completion time dramatically. Thus, another important

problem we investigate is multi-robot cooperatively path planning. The major challenge in extending the single-agent reinforcement learning based path planning solution to the multi-robot scenario is that the action space is exponential with respect to the number of robots. To address this issue, we develop a credit assignment based solution and a sequential roll-out based solution.

1.1.2 Mobile Crowdsourcing

Mobile crowdsourcing is another popular data acquisition strategy in recent years due to explosive smartphone adaptions. Through mobile crowdsourcing, the burden of data collection is transferred from service providers to individuals. One major challenge is how to incentivize participants to participate in data collection campaigns.

In the area of indoor localization, researchers have proposed various crowdsourcing based indoor localization solutions. Some works focus on *how* to use the crowdsourced data to build localization models, and others emulate crowdsourcing by a small number of controlled volunteers(e.g., often from the researcher's lab). Due to operational challenges and costs such as platform development and participants recruitment, no prior real-world crowdsourcing based site survey campaign has been published in literature.

To investigate potential problems that may arise in crowdsourcing based data acquisition and study how participants will react, it is necessary to organize a crowdsourcing campaign in the real-world. For this purpose, we need to develop a fingerprint crowdsourcing platform, an incentive mechanism, a recruitment management method and ways to impart necessary knowledge to users so they can perform tasks in the campaign. In this thesis, we present an end-to-end design and analysis of such

a real world campaign. Although the campaign aims to collect Wi-Fi fingerprints, the proposed methodologies are applicable in collecting other types of spatial data in indoor environments.

1.2 Contributions

The main components of the thesis are summarized in Fig. 1.1. Two mobile sensing schemes, namely, mobile robotic sensing and mobile crowdsourcing are investigated for spatial data acquisition.

For robotic sensing, we focus on planning informative paths for robots given budget constraints. The problem is shown to be NP-hard and two heuristics based algorithms are proposed, namely, a Greedy algorithm and a Genetic Algorithm (GA). The Greedy algorithm picks the next way-point to visit based on the ratio of marginal reward and marginal cost. In GA, we customize chromosomes, selection and mutation operations to solve the IPP problem. As an application example, we conducted experiments for Wi-Fi fingerprint collection. Results show that the Greedy algorithm has the shortest run time, but suffers from poor performance in utility or localization accuracy. In contrast, GA has good performance consistently in all scenarios though at the expense of higher computation complexity than the Greedy approach. The baseline algorithms by extending known algorithms for OP suffer from either high computation complexity for large areas, sub-optimal performance, or both.

To further improve path planning efficiency when budgets vary, we propose a general reinforcement learning framework for IPP based on recurrent neural networks (RNN). Specifically, we model the path planning problem as a sequential decision process, and present a state encoding scheme and a reward function design. A novel

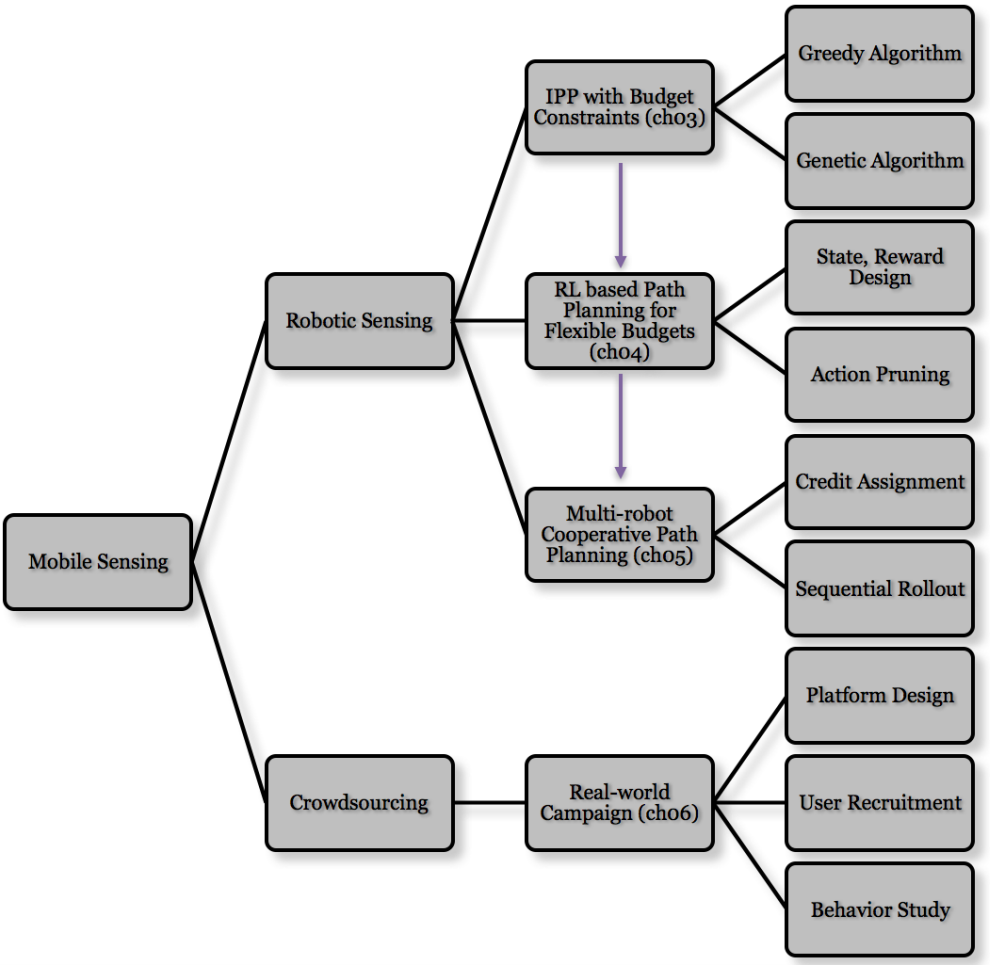


Figure 1.1: Main Contributions and Highlights

action selection method is proposed to accommodate path specifications and improve learning efficiency. With reinforcement learning, we can *learn* the structural characteristics of informative paths and maximize the total future reward, which is equivalent to the utility of a path. Using the learned RL model, informative paths can be *predicted* given any input budget. It improves path planning efficiency dramatically compared with repetitively applying the same IPP algorithm when budgets change.

To address the multi-robot path planning problem, we develop two reinforcement learning based cooperative strategies: independent learning through credit assignment and sequential rollout based learning. Both strategies are highly scalable with respect to the number of robots. Experiment results show that in most cases, the RL based solutions achieve superior or similar performance as a baseline GA-based solution but at only a fraction of running time during inference. Furthermore, when the budgets and initial positions of the robots change, the pre-trained policies can be applied directly.

Lastly, we study the mobile crowdsourcing based sensing scheme. To investigate real-world user response, we launch a large-scale data collection campaign. Specifically, we design and implement a Fingerprint Crowdsourcing System (IFCS). To promote better coverage of target areas, we propose a metric based on the informativeness of participants' collected data to evaluate their contributions. From September 2019 to January 2020, a data collection campaign were conducted in the McMaster University campus. In total, 97 participants signed up for the campaign and 66 of them uploaded fingerprint data. Over 1400 fingerprints had been collected from 24 buildings and 58 floors at the cost of $200. To the best of our knowledge, this work is the first to investigate participant recruitment and behavior analysis for an MCS-based data collection. The results show that an effective incentive and recruitment strategy is essential for a successful campaign. Furthermore, we also find that an appropriate way to teach the participants how to perform the task is also vital since they may not have prior knowledge, although this has not been studied in existing literature.

1.3 Organization

This thesis is organized into the following seven chapters.

- **Chapter 1**: An overview of the motivation and key research questions is presented, followed by a brief introduction of the main components and contributions.
- **Chapter 2**: Preliminaries and related work of this dissertation are introduced, including Gaussian Processes, indoor localization, informative path planning, reinforcement learning and mobile crowdsourcing.
- **Chapter 3**: In this chapter, the informative path planning problem is formulated and two heuristics-based solutions are presented. Experiments show that site survey along informative paths could reduce localization errors given the same budget.
- **Chapter 4**: In this chapter, a reinforcement learning framework for IPP is presented, which can be applied to predict informative paths when budgets change.
- **Chapter 5**: The problem of multi-robot cooperative path planning is formulated. An independent Q-learning approach based on multi-agent credit assignment and a sequential roll out approach are presented and evaluated.
- **Chapter 6**: In this chapter, a real-world crowdsourcing campaign is presented. We discuss in details on the platform design, user recruitment and behavior analysis, and localization experiments.
- **Chapter 7**: Conclusions and future work are discussed in this chapter.

Chapter 2

Preliminaries and Related Work

In this chapter, we first introduce some preliminary knowledge about Gaussian Processes (GPs). GPs are utilized for two purposes in this thesis, namely, building localization models and calculation of informativeness. We then introduce related work on informative path planning, reinforcement learning, mobile crowdsourcing and indoor localization.

2.1 Gaussian Processes

A GP is a collection of random variables, any finite number of which follow a joint Gaussian distribution [87]. Specifically, a GP is defined by a mean function $m(\mathbf{x})$ and a covariance (also know as kernel) function $k(\mathbf{x}_p, \mathbf{x}_q)$. Let f is a function drawn from this GP, for any finite subset of variables $\mathbf{x}_1, ..., \mathbf{x}_n$, the respective function values

$f(\mathbf{x}_1), ..., f(\mathbf{x}_n)$ have a joint multivariate Gaussian distribution,

$$\mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}_1) \\ \vdots \\ m(\mathbf{x}_n) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & ... & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & ... & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}\right). \tag{2.1.1}$$

The mean function m can be any valid function. The covariance (kernel) matrix K is generated by the covariance function and is positive semidefinite. One frequently adopted covariance function is the exponential kernel function

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 exp(-\frac{||\mathbf{x}_p - \mathbf{x}_q||}{l}), \tag{2.1.2}$$

where σ_f^2 is the maximum covariance between variables and l is the length scale term that controls the smoothness. GPs model distributions over functions instead of variables, and they are widely used in modeling spatial processes.

Gaussian Process Regression is a frequently utilized non-parametric regression method. Let a training dataset be $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$. For notation simplicity, the training data could also be represented using feature matrix X and the corresponding measurement vector $\mathbf{y}$. Suppose the measurements contain a Gaussian noise term ϵ, and $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Let f is a function drawn from a GP prior, the i-th measurement is then given by

$$y_i = f(\mathbf{x}_i) + \epsilon_i. \tag{2.1.3}$$

The noise term can also be merged into the covariance function,

$$cov(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{pq}, \tag{2.1.4}$$

where $\delta_{pq} = 1$ if $p = q$ and zero otherwise.

GP based regression is a widely used non-parametric regression method. Let X_* denotes a feature matrix, and the corresponding unobserved values be $\mathbf{y}_*$. Under the GP assumption, given the training data $\mathcal{D}$, we have

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right), \tag{2.1.5}$$

where $m(X)$ and $m(X_*)$ are the mean value vector for the training and unobserved data respectively. $K(X, X)$ denotes the covariance matrix evaluated for all pairs of training locations, and similarly for $K(X, X_*)$ and $K(X_*, X_*)$. By the property of conditional multivariate Gaussian distributions, we have

$$\mathbf{y}_* | X, \mathbf{y}, X_* \sim \mathcal{N}(\mu_*, \Sigma_*), \tag{2.1.6}$$

where

$$\mu_* = m(X_*) + K(X_*, X)(K(X, X) + \sigma_n^2 I)^{-1}(\mathbf{y} - m(X)), \tag{2.1.7}$$

and

$$\Sigma_* = K(X_*, X_*) - K(X_*, X)(K(X, X) + \sigma_n^2 I)^{-1} K(X, X_*). \tag{2.1.8}$$

Also noting that $\mathbf{y} \sim \mathcal{N}(m(X), K(X, X) + \sigma_n^2 I)$, we can obtain the log marginal

likelihood,

$$\begin{aligned} log\ p(\mathbf{y}|X) = &-\frac{1}{2}(\mathbf{y}-m(X))^T(K(X,X)+\sigma_n^2 I)^{-1}(\mathbf{y}-m(X)) \\ &-\frac{1}{2}log|K(X,X)+\sigma_n^2 I| - \frac{n}{2}log 2\pi. \end{aligned} \tag{2.1.9}$$

By maximizing this log marginal likelihood term, the hyper-parameters of the covariance function and the noise variance could be estimated accordingly.

In this thesis, we assume the kernel of the underlying GP of the environmental phenomenon is isotropic [56], i.e, given all the hyper-parameters Θ of the kernel, the covariance of two random variables only depends on their distances, formally, $k(\mathbf{x}_p, \mathbf{x}_q) = k_\Theta(||\mathbf{x}_p - \mathbf{x}_q||)$. This is a special case of the so-called stationary kernel where the covariance depends on the difference between two inputs, i.e, $k(\mathbf{x}_p, \mathbf{x}_q) = k_\Theta(\mathbf{x}_p - \mathbf{x}_q)$. For stationary kernels, they are invariant to translations in the input space, and for isotropic kernels they are invariant to all rigid motions [87].

2.2 Informative Path Planning

The goal of IPP is to plan a path such that the utility (informativeness) of data collected along the path is maximized. The informativeness can be measured by mutual information (MI). In [36], static sensors are deployed for spatial monitoring, and the authors use MI as the metric for optimal sensor location selection . In [93], similarly MI is adopted as the metric to search informative paths for mobile robots.

One major challenge of IPP is that the problem is NP-hard [108]. Thus, it is difficult to achieve optimality and efficiency at the same time for a large scale problem. A few existing solutions to IPP rely on the Recursive Greedy (RG) algorithm [15]. The

basic idea is to exhaustively consider all possible combinations of intermediate vertices and budgets, and then apply the algorithm recursively on the smaller sub-problems. IPP solutions based on RG can be found in [8, 93]. Specifically, [8] also considers rewards from edges besides vertices. To reduce the high computation complexity of RG, [93] proposes the use of spatial decomposition to create a coarse graph by grouping vertices into cells. The algorithm is then applied on the cell-based coarse graph. Unfortunately, planning paths based on a surrogate coarse graph may not give the optimal solution for the original IPP problem.

Evolutionary strategies have also been investigated for IPP. The authors in [70] model the path planning process as a control policy and proposed a heuristic strategy by incrementally constructing a policy tree. In [41, 84], the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is utilized to optimize paths in a continuous space. To reduce the search space, paths are constrained using control points and constructed with splines. After path planning, in the robot deployment stage, re-planning and adaptive sampling are also considered so as to focus on regions of interests. The basic idea is to omit locations where the predicted observation values (based on GP) are below a certain threshold.

Some works make further assumptions that each vertex can be visited only once, and rewards can be obtained only at the vertices. Under such assumptions, IPP can be decomposed to two steps: subset (vertices) selection and path construction. Once the set of vertices are determined, a traveling salesman problem (TSP) solver can be utilized to construct a path with the minimum cost. In [4], a randomized algorithm is presented, where vertices are randomly added or removed repetitively to track the best subset. Similarly, in [69], way-points are added incrementally and a TSP solver

is utilized to generate paths. However, in practical IPP scenarios, the aforementioned assumptions do not hold, since i) vertices could be visited multiple times, particularly when the graph is not complete; ii) rewards can be obtained when an agent moves along the edges with sensors recording the spatial data.

Due to the NP-hardness, existing solutions suffer from sub-optimal performance or incur a high computation complexity. In addition, when budgets change, even with the same start and terminal locations, the path planning process needs to be executed from scratch, which further degrades efficiency.

2.3 Reinforcement Learning

Under the framework of RL [47, 96], an agent interacts with the environment. The process can be formulated by a Markov Decision Process (MDP) $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$, where

- $\mathcal{S}$ is a finite set of states;
- $\mathcal{A}$ is a finite set of actions;
- $\mathcal{T}$ is a state transition function[1] defined as $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$;
- $\mathcal{R}$ is a reward function defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathbb{R}$ is a real value reward signal.

To solve the MDP with RL, a policy π is required for decision making. The policy can be deterministic or stochastic. A deterministic policy is defined as $\pi(s) : \mathcal{S} \to \mathcal{A}$, i.e., given the state, the policy outputs the action to take for the following step.

[1] In this thesis we consider deterministic transitions.

At each time step t, the environment is at a state $s_t \in \mathcal{S}$. The agent makes a decision by taking an action $a_t = \pi(s_t) \in \mathcal{A}$. It then receives an immediate reward signal r_t and the state moves to $s_{t+1} \in \mathcal{S}$. The goal of RL is to find a policy π such that the total future reward

$$R_t = r_t + \gamma r_{t+1} + ... + \gamma^{T-t} r_T \tag{2.3.1}$$

is maximized, where $\gamma \in [0, 1]$ is a discount factor that controls the weight of future reward and T is the last action time.

There are two main types of approaches towards RL, namely value-based and policy-based approaches. Representative algorithms in the two categories are Q-learning and policy gradient, respectively. Q-learning aims to learn a function mapping between state-actions and q-values. Once the Q-function is learned, the corresponding policy can be induced. On the other hand, policy gradient formalizes and learns the policy directly through gradient ascent. Researchers have also combined value-based and the policy-based approaches, and proposed the actor-critic [54] methods. A critic (value-based) is trained to evaluate the actions selected by the actor (policy-based). In many cases, actor-critic methods have better convergence compared with policy gradient methods such as the reinforce algorithm.

In recent years, with the advancement of deep neural networks [61], deep reinforcement learning [65] emerged as an effective RL paradigm, and it has been applied in a variety of applications such as games, robotics and traffic signal control. In particular, two works [6, 52] attempt to apply RL in combinatorial optimization. In [6], the authors focus on the TSP and utilize a pointer network to predict the distribution

of vertex permutations. Parameters of the network are optimized using policy gradient with negative tour lengths as reward signals. In [52], a Q-learning approach is presented. Graph embedding techniques are leveraged for graph representation, and the solution is evaluated through Minimum Vertex Cover, Maximum Cut and TSP.

2.4 Mobile Crowdsourcing

In a mobile crowdsourcing system, participants are responsible to consciously meet application requests by deciding when, what, where, and how to perform required tasks. They contribute data and take an active part in the task allocation and assignment process, which are usually managed by a central authority with different methodologies. In some applications, a teaching phase is needed where users first acquire the required technical skills to accomplish a specific task. For instance, collecting Wi-Fi fingerprints properly may be difficult to inexperienced users, leading to inaccurate and erroneous data gathered.

To be effective, an MCS campaign requires a significant contribution from users, who sustain costs in terms of time and battery spent in the sensing process [12]. The required effort is particularly relevant in participatory systems, where users may be reluctant to contribute and thus incentive mechanisms are crucial to motivate them. Main categories of incentive mechanisms include entertainment, service, and money [46, 117]. Entertainment approaches incentivize people by turning tasks into games, while service-oriented ones reward with services offered by the platform in exchange for contributions. Lastly, monetary incentives provide participants payments for their contributions [68, 109].

Many studies have been conducted in using MCS to solve problems in our daily

lives. Liu *et al.* in [66] presented Third-Eye, which is an MCS system for air quality monitoring. Mobile phones with cameras are utilized to take outdoor photos, which are further used to estimate the air quality with computer vision techniques. In [118], the authors proposed a vehicular MCS system for pothole profiling. The system relies on the built-in inertial sensors of vehicle-carried phones to estimate the locations, lengths and depths of potholes. The result is sent to drivers and the road maintenance sector. CrowdX [17] is proposed to enhance the automatic construction of indoor floor plans with crowdsourced traces. TrailSense [53] can automatically infer whether trail segments are risky for hiking by analyzing sensor data from hikers' smartphones. Acer *et al.* [1] argued that urban-scale MCS could leverage the daily routines of mobile workforces that roam around an urban area. Wang *et al.* [104] developed the CrossCheck symptom prediction system, which is used to monitor patients' trajectory of psychiatric symptoms. In [16], the authors presented TurnsMap, which is an MCS system for classifying protected/unprotected left turns using IMU sensor data from on-board mobile devices in a moving car.

For Wi-Fi based indoor localization, researchers have also proposed crowdsensing based site survey. Both opportunistic sensing and participatory sensing approaches have been investigated. Very few work with limited scales has been done to conduct indoor fingerprint site survey campaign using MCS. Park *et al.* developed a system called OIL [80], which populates a Wi-Fi RSS fingerprint database with user provided fingerprints over time. A Voronoi diagram-based method was developed for conveying localization uncertainty and increasing coverage. The authors also incorporated a clustering-based method that automatically discards erroneous user inputs through outlier detection in the signal space. The evaluation was carried out in a single

building with nine floors. 19 people were invited to participate and were provided with tablets by the researchers. After 9 days, the average localization reached 5.3 m in the building.

UJIIndoorLoc [99] and [67] are two crowdsourced fingerprint datasets. These datasets provide opportunities to investigate localization algorithms using the crowdsourced fingerprints. Specifically, [67] is a benchmark dataset for "crowdsourced" fingerprints. The dataset is collected by eight volunteers with 21 Android devices. The devices were used by different persons so as to mimic a crowdsourced data gathering scenario. Strictly speaking, it does not reflect many characteristics of real-world MCS campaigns, because user behaviors are highly unpredictable in terms of when, how and where data are collected.

MCS based survey puts a burden on participants, and it can be challenging to recruit sufficient participants. Further, as analyzed in [82], the characteristics of crowdsourced fingerprints can degrade localization performance. For instance, because devices used in site survey are probably different from the devices to be localized, a problem called device heterogeneity [81] can arise. Moreover, location labels may be incorrect due to mistakes of the participants. As a result, despite extensive research together with various proposed solutions, Wi-Fi fingerprint MCS is still not widely adopted for site survey.

2.5 Indoor Localization

Although the Global Positioning System (GPS) achieved a great success in the last decade and lots of applications are built on top of GPS, in indoor spaces, there are still no mature localization solutions. Existing indoor localization approaches can be

broadly divided into two categories, namely infrastructure-based and infrastructure-free approaches. Specifically, solutions in the first category need dedicated hardware such as anchor nodes [106] to help determine target locations. On the other hand, infrastructure-free solutions use existing infrastructure and require no extra hardware. Among infrastructure-free approaches, Wi-Fi RSS based indoor localization is attractive due to the wide availability of Wi-Fi Access Points (APs) in indoor environments and has been extensively investigated.

There are generally two stages in RSS-based localization solutions, namely the *off-line* (site survey) stage and the *on-line* localization stage [29]. In the off-line stage, Wi-Fi RSS measurements are collected at different pre-defined reference locations. This stage is very time-consuming, especially for large buildings or indoor areas. To shorten the site survey process, a few works proposed to use path-based data collections [22, 64]. The basic idea is for users to collect data using a mobile device while walking along a path with known start and terminal locations. Location labels of RSS measurements could then be inferred by the device's built-in Inertial Measurement Unit (IMU) sensors and the associated timestamps.

Once site surveys are done, data collected can be used to build a fingerprint database or machine learning models for localization. A large number of models or algorithms have been proposed for the on-line location inference stage, such as k-nearest neighbor [5, 92], [24, 100], SVM [10, 28], neural network [74] and fingerprint gradient [112]. In [114], the authors combined vision and Wi-Fi fingerprints and achieved sub-meter localization accuracy.

GPs have also been utilized to build localization models. The first work on GP

based localization is GPPS [90]. In GPPS, a GP regression model is trained individually for each AP with its respective calibration dataset. In the inference stage, given the input signals, the target location is estimated to be the position that has the maximum probability to generate the signals. When training the GP models, a linear function is adopted as the mean function. Specifically, the AP location is estimated to be the center of three locations with the strongest signal strengths. For each calibration point, the mean value depends on the distance to the AP. Ferris *et al.* [27] extend GPPS. They combine GP with a Bayesian filter for location estimation to build on a mixed graph / free space representation of indoor environments. The method is shown to work both for Wi-Fi and GSM localization.

Chapter 3

Informative Path Planning with Budget Constraints

In this chapter, we formally introduce the concept of informative path planning for mobile robotic sensing. Two algorithms are presented to plan informative paths given a known budget constraint. Experiments are designed to validate the proposed solutions.

3.1 Problem Formulation

The IPP problem operates on a graph. In practice, the graph can be created depending on the topology of the target area. Next, we first define the general path planning problem and then extend it to IPP.

3.1.1 General Path Planning with a Limited Budget

We formalize a path planning problem on a graph with a tuple $\langle G, v_s, v_t, f(\mathcal{P}), B\rangle$. Specifically,

- $G = (\mathcal{V}, \mathcal{E})$ is a graph, with $\mathcal{V}$ and $\mathcal{E}$ representing the set of vertices and edges, respectively. Each $v \in \mathcal{V}$ is associated with a physical location $\mathbf{x}$. Note that we consider the target area to be 2D shaped, and thus $\mathbf{x}$ represents a 2D coordinate. For each $e \in \mathcal{E}$, there is a cost $c(e)$ (e.g., the length of the edge) for travelling along the edge.

- $v_s \in \mathcal{V}$ is the start location, and $v_t \in \mathcal{V}$ is the expected terminal location. These locations could be the depot of the robot or charging stations.

- $\mathcal{P} = [v_s, ..., v_k, ..., v_t]$ denotes a valid path[1], and $f(\mathcal{P})$ represents the corresponding utility (or reward) of the path. In this thesis, we will use the terms of "utility" and "reward" interchangeably.

- B represents the budget available for the path. We emphasize two scenarios: i) B is known and fixed; ii) B can be variable, e.g., when the robot is not fully charged. Almost all existing path planning algorithms assume the budget is fixed. When the budget changes, a natural solution is to re-run the algorithm with the updated budget. In this chapter we assume the budget is fixed, and later chapters we will discuss solutions to avoid searching from scratch when budgets change.

[1]In graph theory, a path is defined as a sequence of vertices and edges without repeated vertices or edges. To be consistent with existing IPP literature, we allow repetition of vertices on a path, the equivalent of a walk in graph theory.

The cost of a path $\mathcal{P}$ is the sum of edge cost along the path,

$$c(\mathcal{P}) = \sum_{i=1}^{|\mathcal{P}|-1} c(\mathcal{P}[i], \mathcal{P}[i+1]), \tag{3.1.1}$$

where $\mathcal{P}[i]$ is the i-th vertex in $\mathcal{P}$ and $(\mathcal{P}[i], \mathcal{P}[i+1])$ represents the corresponding edge. The objective of path planning is to find the optimal path that satisfies

$$\mathcal{P}^* = \arg\max_{\mathcal{P}\in\Psi} f(\mathcal{P}) \text{ s.t. } c(\mathcal{P}) \leq B, \tag{3.1.2}$$

where Ψ is the set of all paths in G from v_s to v_t.

One classic instance of the general path planning problem is the OP [30, 37, 103]. In OP, each vertex is associated with a reward. Given a budget, the goal is to find a subset of vertices to visit so as to maximize the total collected reward, defined as the sum of rewards from individual vertices.

3.1.2 Informative Path Planning

IPP is special case of the general path planning problem, where the reward function reflects the informativeness of the data collected along paths. Next, we present the details of $f(\mathcal{P})$ for IPP based on GPs and MI.

Assume that the data to be collected are modeled by a GP. Thus, for each $v \in \mathcal{V}$ at a physical location $\mathbf{x}$, the corresponding measurement y_v (e.g., temperature or humidity) is associated with a Gaussian distributed random variable, and all the variables $\mathbf{y}_\mathcal{V}$ at locations of $\mathcal{V}$ follow a joint multivariate Gaussian distribution. For simplicity, we also denote the multivariate Gaussian distribution with $\mathcal{N}(m(X_\mathcal{V}), \Sigma_\mathcal{V})$, where $X_\mathcal{V}$ is a matrix of the locations of $\mathcal{V}$, and $\Sigma_\mathcal{V}$ is the covariance matrix generated

through a covariance (kernel) function.

The differential entropy (also referred to as continuous entropy) of $\mathbf{y}_{\mathcal{V}}$ is

$$H(\mathbf{y}_{\mathcal{V}}) = \frac{1}{2}\ln|\Sigma_{\mathcal{V}}| + \frac{n}{2}(1+\ln(2\pi)), \tag{3.1.3}$$

where n is the number of rows in the matrix.

Given a path $\mathcal{P} = [v_s, ..., v_k, ..., v_t]$, suppose data will be collected by an agent along the path every d-meter interval (depending on the robot's travel speed and sample frequency). The sample locations can be easily calculated using the coordinates of the vertices. We denote all the sample locations as $X_{\mathcal{S}}$ and the corresponding measurements as $\mathbf{y}_{\mathcal{S}}$. According to the properties of GPs introduced in Chapter 2.1, the posterior distribution of $\mathbf{y}_{\mathcal{V}}$ given $\mathbf{y}_{\mathcal{S}}$ is $\mathcal{N}(\boldsymbol{\mu}', \Sigma')$, with

$$\boldsymbol{\mu}' = m(X_{\mathcal{V}}) + K(X_{\mathcal{V}}, X_{\mathcal{S}})(K(X_{\mathcal{S}}, X_{\mathcal{S}}) + \sigma_n^2 I)^{-1}(\mathbf{y}_{\mathcal{S}} - m(X_{\mathcal{S}})), \tag{3.1.4}$$

$$\Sigma' = K(X_{\mathcal{V}}, X_{\mathcal{V}}) - K(X_{\mathcal{V}}, X_{\mathcal{S}})(K(X_{\mathcal{S}}, X_{\mathcal{S}}) + \sigma_n^2 I)^{-1}K(X_{\mathcal{S}}, X_{\mathcal{V}}). \tag{3.1.5}$$

Here σ_n represents the noise variance of the underlying GP, and $K(X_{\mathcal{V}}, X_{\mathcal{S}})$ is the kernel matrix generated by $k(\cdot,\cdot)$ with pair-wise entries in $X_{\mathcal{V}}$ and $X_{\mathcal{S}}$. The conditional differential entropy of $\mathbf{y}_{\mathcal{V}}$ given the observations $\mathbf{y}_{\mathcal{S}}$ is

$$H(\mathbf{y}_{\mathcal{V}}|\mathbf{y}_{\mathcal{S}}) = \frac{1}{2}\ln|\Sigma'| + \frac{n}{2}(1+\ln(2\pi)). \tag{3.1.6}$$

The MI based reward of the path $\mathcal{P}$ can then be calculated by

$$f(\mathcal{P}) = \mathrm{MI}(\mathbf{y}_{\mathcal{V}}; \mathbf{y}_{\mathcal{S}}) = H(\mathbf{y}_{\mathcal{V}}) - H(\mathbf{y}_{\mathcal{V}}|\mathbf{y}_{\mathcal{S}}). \tag{3.1.7}$$

Note that since differential entropy only depends on the kernel matrix (i.e, the kernel function and $\mathcal{P}$), rewards can be calculated analytically *without* travelling along the actual path and taking real measurements. That is why informative paths can be planned in advance.

However, the kernel function $k(\cdot, \cdot)$ usually has some hyperparameters that need to be estimated. In some application scenarios, such as wireless sensor networks with mobile elements [3], the hyperparameters can be learned using measurements from the existing static sensors. In cases where no prior studies are available like in [8, 15, 36], a round of pilot data collection is conducted for hyperparameter estimation. Given a small set of pilot data $(X_{\mathcal{D}}, \mathbf{y}_{\mathcal{D}})$ collected in advance at locations $X_{\mathcal{D}}$ with measurements $\mathbf{y}_{\mathcal{D}}$, the reward can then be calculated with

$$f_{\mathcal{D}}(\mathcal{P}) = \mathrm{MI}(\mathbf{y}_{\mathcal{V}}; \mathbf{y}_S \cup \mathbf{y}_{\mathcal{D}}) = H(\mathbf{y}_{\mathcal{V}}) - H(\mathbf{y}_{\mathcal{V}} | \mathbf{y}_{\mathcal{S}} \cup \mathbf{y}_{\mathcal{D}}). \tag{3.1.8}$$

Note that although in our problem formulation we assume the environment is in 2D, it is straightforward to extend the formulation to 3D environments. In a 3D environment, each vertex in the graph is associated with a 3D coordinate. The GP can be extended accordingly by assuming the locations of observations are in 3D, and the entropy or mutual information are still determined by the covariance matrices. Examples of applying GPs in a 3D environment can be found in [41].

Given the input of IPP as $< G, v_s, v_t, f(\mathcal{P}), B >$, one naive approach is to enumerate all the valid paths from v_s to v_t and choose the path with the highest $f(\mathcal{P})$. However, brute force search is not computationally feasible for large problem instances because the IPP problem is NP-hard, which will be proven in the next section.

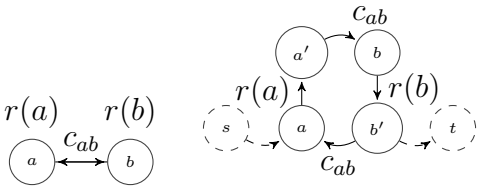


Figure 3.1: An example of the transformation from an OP graph G to an IPP graph G', where the starting node is a and terminating node is b. In the transformation, two dummy nodes with zero reward and cost, s and t are added.

3.1.3 NP-hardness of IPP

IPP can be shown to be NP-hard by reducing an instance of OP to an instance of IPP within polynomial time. OP is defined on a graph $G = (V, E)$, with a vertex reward function $r(v)$ and an edge cost function $c(e)$. The start and terminal vertices are v_s and v_t, and the travel cost is limited by B. The following steps show how to reduce an instance of OP to an instance of IPP[2]:

- A graph $G' = (\mathcal{V}', \mathcal{E}')$ is constructed for IPP. Initially, $\mathcal{V}' = \mathcal{V}$, and then for $v \in \mathcal{V}$, a *shadow* vertex v' is created by making a copy of v.

- To construct $\mathcal{E}'$ in G', for $e_{ab} \in \mathcal{E}$ in OP, we connect $(a, a'), (a', b), (b, b'), (b', a)$ as directed edges.

- In G', we assign the reward associated with each edge (v, v') as $r(v)$ in OP, and all other edges have zero rewards. Furthermore, the cost of (v, v') is set to be 0, and all other edges have the same cost as in OP.

- Two dummy vertices with zero edge cost and zero reward, s and t are added to accommodate the corresponding start and terminal vertices in OP.

[2]We deem the undirected graph G as a bidirected graph.

Fig. 3.1 shows an example transformation for a simple graph. It can be observed that in the converted graph G', for every original node v, the surrounding vertices are shadow vertices. Similarly, for every shadow vertex v', the surrounding vertices are original vertices. This forces the solution to be a sequence of vertices alternating between v and v'. With such a transformation, it is easy to show that the solution in G for OP is optimal if and only if the corresponding solution in G' for IPP is optimal.

Let the optimal IPP solution be $[a, a', b, b', ..., k, k']$ (dummy vertices s and t are omitted, a and k are the start and terminal vertices) in G', then the corresponding optimal OP solution in G is $[a, b, ..., k]$. If the optimal solution for OP is different from this path, e.g., $[a, x, ..., k]$ with a larger reward, we can always construct a better path $[a, a', x, x', ..., k, k']$ in G', which leads to a contradiction with the fact that $[a, a', b, b', ..., k, k']$ is the optimal IPP solution. Conversely, for any optimal OP solution in G, we can similarly construct the optimal IPP solution in G'. Since the reduction can be constructed in polynomial time and OP is NP-hard [51], IPP is also NP-hard.

3.2 Informative Path Planning Algorithms

Due to the NP-hardness, we resort to heuristic algorithms to solve the problem. Specifically, we first present a Greedy algorithm based on the Steiner TSP solver, and then we introduce GA to solve IPP.

3.2.1 Greedy Algorithm

Greedy algorithms are known to achieve constant approximation ratios [36] for some submodular optimization problems, such as observation selection constrained by cardinality and budget [55]. The IPP problem, on the other hand, is more challenging since its budget constraints are defined on a graph with path length. The RG algorithm in [15] is a pseudo-polynomial algorithm with a logarithmic approximation ratio. However, it has a long run time, even for small graphs. In this section, we develop a simple greedy algorithm with low computation complexity which adds vertices incrementally. It cannot achieve a constant approximation ratio since it is subjected to a constraint on the path length [55].

The first step is to define the greedy criteria. Suppose the current path planned is $\mathcal{P}$, and let $\mathcal{V}(\mathcal{P})$ be the vertices in $\mathcal{P}$. For each candidate vertex v_c that is not contained in the current path, the shortest path to traverse $\mathcal{V}(\mathcal{P}) \cup \{v_c\}$ is denoted by $\mathcal{P}_c$. The marginal benefit-cost ratio (MBCR) of extending the current path to v_c is defined as $MBCR(\mathcal{P}, v_c) = \frac{f_D(\mathcal{P}_c) - f_D(\mathcal{P})}{PathLength(\mathcal{P}_c) - PathLength(\mathcal{P})}$. The greedy algorithm then selects the vertex that has the highest MBCR among all remaining vertices. In computing both the utility and the cost of adding an extra vertex, an instance of the Steiner TSP (STSP) from source v_s to terminal v_t is solved.

The Greedy algorithm is outlined in Algorithm 1. The complexity is mainly determined by the Steiner TSP algorithm. For instance, if the algorithm has a complexity of $O(n^2 * 2^n)$ (due to a dynamic programming based TSP algorithm), the complexity of the Greedy algorithm is $O(\frac{B}{e_{min}} * n^3 * 2^n)$, where B is the budget, e_{min} is the minimum edge length and n represents the number of vertices in the graph.

It is easy to construct an example where Greedy performs arbitrarily bad. However, through evaluation, we see that under certain budgets, Greedy can achieve competitive results.

Algorithm 1: Greedy Algorithm

Input : the problem Graph G and the budget B
the start and terminal vertices v_s and v_t
the pilot data set D

Output: the vertex order of the returned path

```
1  P = ShortestPath(v_s, v_t)
2  while PathLength(P) <= B do
3      foreach v_c ∈ V ∧ v_c ∉ V(P) do
4          P_c = SteinerTSP(V(P) ∪ {v_c})
5          MBCR(P, v_c) = (f_D(P_c) − f_D(P)) / (PathLength(P_c) − PathLength(P))
6      end
7      v_best = arg max(MBCR(P, v_c))
8      P_best = SteinerTSP(V(P) ∪ {v_best})
9      if PathLength(P_best) <= B then
10         P = P_best
11     else
12         break
13     end
14 end
15 return P
```

3.2.2 Genetic Algorithm

Genetic algorithm is a powerful and efficient evolutionary algorithm which can be utilized to solve both numerical and combinatorial optimization problems. The main idea is to imitate the process of biological natural selection. Though there are no guarantees that GA will find the global optimal solution, its solution is likely to be close to the global optimum [71]. The mutation mechanism can protect the algorithm from being stuck in a local optimum by diversifying the population.

In GA, a chromosome is utilized to encode a feasible solution to a specific optimization problem. A pool of chromosomes is maintained, which is also known as the population. For each chromosome, a corresponding fitness score is calculated by a fitness function. The initial population is usually randomly generated. After that, an evolutionary process starts iterating. In each iteration, four steps are involved:

- *Selection:* Individuals (also known as parents) from the population are selected based on their fitness scores.
- *Crossover:* The selected parents reproduce new offspring by a crossover process.
- *Mutation:* Some of the offspring are selected for mutation to increase the diversity of the population.
- *Update:* The population is updated by merging the offspring and parents.

After a number of iterations, the chromosome with the highest fitness score is selected as the final solution.

A large body of literature [34, 59, 85] can be found using GA to solve the TSP problem. Specifically, [59] summarized the attempts to solve TSP with different crossover and mutation operators. GA has also been investigated to tackle the orienteering problem [50, 83, 98]. However, in [50, 98], the proposed GA operators require the graph to be complete. The case where the input is an incomplete graph is considered in [50] and vertex revisit is allowed. However, only the first visit receives rewards, which is not the case in IPP.

Next we discuss the details of adapting the GA framework to solve the IPP problem.

3.2.2.1 Encoding and Fitness Function

Since the final solution to IPP is a list of ordered vertices from v_s to v_t, a path-based representation is a natural choice. Specifically, each chromosome represents a solution in the form of $\mathcal{P} = [v_s, v_i, ..., v_j, v_t]$. For each pair of consecutive vertices in the ordered list, there must be an edge connecting them. The fitness function is chosen to be equal to the utility function $f_{\mathcal{D}}$ as discussed in Section 3.1. As such, the fitness score of $\mathcal{P}$ represents how much uncertainty can be reduced by sampling along the path $\mathcal{P}$. This is directly linked to the optimization objective.

3.2.2.2 Initializing Population

When using GA to solve TSP on a fully connected graph, the population can be initialized with random permutations of all the vertices since each vertex is visited only once and the solution is a tour without start and terminal locations. However, when initializing the population for IPP, three constraints must be satisfied. Specifically, i) the start and end vertices are specified; ii) the budget limitation is satisfied; and iii) each vertex is allowed to be visited multiple times.

Algorithm 2 describes the procedure to initialize the population. Starting from v_s, an adjacent vertex to the current vertex is randomly selected and appended until half of the budget is used. The terminal vertex v_t is then connected by the shortest path. If there is still remaining budget, vertices are sampled and inserted. If the generated chromosome by such a scheme exceeds the budget limit, the chromosome is dropped and the procedure restarts.

Algorithm 2: GA Population Initialization

Input : the problem Graph G and the budget B
the start and terminal vertices v_s and v_t
the population size $popsize$

Output: the initial population

```
1  poppool = [];
2  while sizeof(poppool) < popsize do
3      seq = [v_s]
4      do
5          v_last = last vertex in seq
6          v_adj = sample a vertex from v_last's neighbors
7          append v_adj to seq
8          if length(seq + v_adj) > 0.5 * B then
9              delete v_adj from seq
10             break
11     while length(seq) < 0.5 * B;
12     v_mid = last vertex in seq
13     seq2 = shortestpath(v_mid, v_t)
14     do
15         v = sample a vertex from V − (seq ∪ seq2)
16         insert v into seq2
17         /*Note the insertion location is the position which will cause the
             minimum budget increase. If there is no direct edges between two
             vertices, shortest path is utilized*/
18         if length(seq2) > 0.5 * B then
19             delete v from seq2
20             break
21     while length(seq2) < 0.5 * B;
22     chromosome = seq + seq2[1 :]
23     add chromosome to poppool
24 end
25 return poppool
```

3.2.2.3 Selection and Crossover

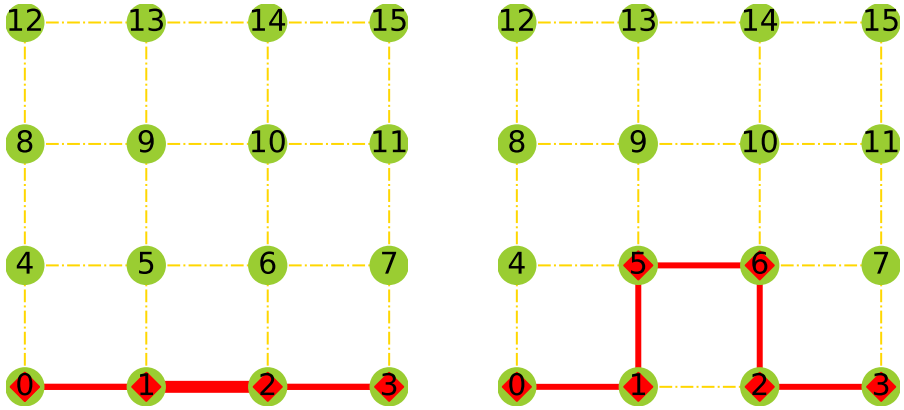
Selection simulates the idea of "Survival of the fittest". Typical selection methods include Roulette Wheel Selection, Rank Selection and Tournament Selection [95]. We adopt the popular tournament scheme to select parents. During each tournament, k individuals are randomly selected and the winner (the one with the highest fitness score) is picked as one parent.

Crossover takes place between two selected parents. A single point crossover similar to [50, 83] is utilized to generate offspring. Specifically, common vertices (except for the start and terminal vertices) in the two parents are identified and one common vertex is randomly picked as the crossover point. Segments are then exchanged around the common vertex. For instance, suppose two parents $[v_s, v_1, v_2, v_5, v_7, v_8, v_t]$ and $[v_s, v_3, v_4, v_5, v_6, v_9, v_t]$ are selected, and v_5 is the common vertex. After crossover, two offspring $[v_s, v_3, v_4, v_5, v_7, v_8, v_t]$ and $[v_s, v_1, v_2, v_5, v_6, v_9, v_t]$ are created. Furthermore, only the offspring that do not exceed the budget limit can survive. If there are no common vertices, no offspring are generated.

3.2.2.4 Mutation

A mutation mechanism is designed to promote the diversity of the population. From an optimization perspective, it enables the algorithm to escape from a local optimum. Various mutation operators (insertion, exchange, displacement, inversion) are developed for TSP problems when the graph is complete. When a graph is not complete, most of these operators are not feasible. Therefore, we propose a local extension mutation operator. Specifically, for a chromosome (a path) $\mathcal{P}$, we randomly select two intermediate adjacent vertices v_i, v_j as the mutation locations. The simplest case

is when v_i and v_j have a common adjacent vertex v_k. In this case, v_k can be directly inserted between v_i and v_j. When v_i and v_j do not share any common adjacent vertex, we create a connection through their own adjacent vertices as shown in Fig. 3.2.



(a) before mutation (b) after mutation

Figure 3.2: An example of mutation. (a) shows a path [0,1,2,3] and vertices 1 and 2 are the selected mutation positions. The two vertices do not have a common adjacent vertex. However, the connection can be built through 1's adjacent vertex 5 and 2's adjacent vertex 6 as shown in (b).

3.3 Performance Evaluation

In this section, we conduct experiments to compare the performance of different IPP algorithms. Specifically, we consider the collection of Wi-Fi RSS for indoor localization. The approach can be readily extended to include other types of spatial data.

Next, we introduce the implementation details. Then we describe the evaluation methodology and present the experimental results.

3.3.1 Implementation

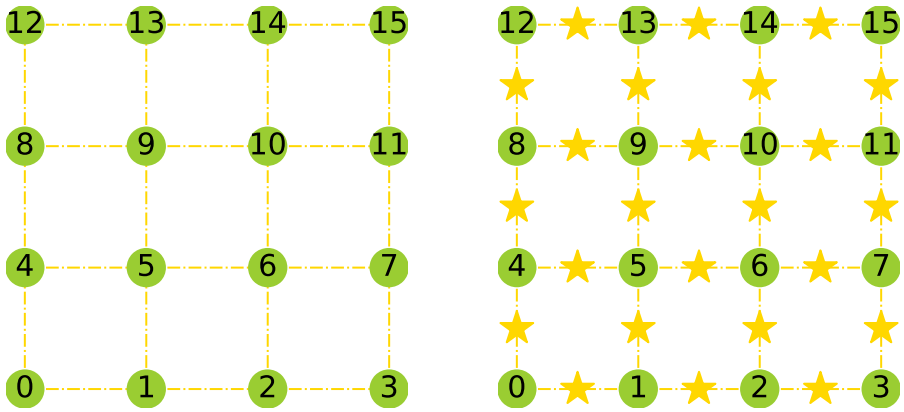
All algorithms are implemented in Python 2.7 and are executed on a MacBook (Intel Core i7, 16GB RAM[3]). We use GPy [33] to train and optimize GP models. Graphs are created and represented by NetworkX [39].

In addition to the Greedy algorithm and GA, we have extended the RG algorithm in [15] and RO in [4] for IPP. The details are as follows.

Recursive Greedy Algorithm We implemented QP-RG as outlined in [15]. The main modification is that when evaluating $f_{\mathcal{D}}(\mathcal{P})$, we take samples along the edges instead of only considering the vertices. The recursion depth parameter I is set to two. We have tried to increase this parameter to three, but find that the algorithm failed to terminate in hours since the complexity is exponential with respect to I. Similar execution time can be found in [8] even for a graph of 16 vertices.

Edge based Random Orienteering (ERO) We extended RO [4] to handle edge-based rewards. The edge nodes are added or deleted randomly. Fig. 3.3 gives an example of such a transformation. Furthermore, a Steiner TSP solver is used to plan a path among the selected edge nodes, since the resulting graph is incomplete. Our Steiner TSP solver is implemented based on the Concorde TSP solver [21] by calculating the shortest path between every pair of vertices [63].

[3]We will also use brute force approach to search for the optimal path, which is run on Compute Canada due to the extremely long run time.



(a) the original graph (b) represent edges as nodes

Figure 3.3: Illustration of graph transformation for ERO. The original graph is a 4 by 4 grid graph. The gold stars represent the edge nodes.

3.3.2 Evaluation Methodology

3.3.2.1 Experimental Design

To evaluate different path planning strategies fairly, it is important to subject them to comparable data. However, due to the time varying nature of Wi-Fi signals, even collecting RSS along the same trajectory multiple times would result in different data. Another consideration is that, during actual data collection using a robot, its speed may vary due to the presence of people in target areas. To mitigate these two issues, we propose an efficient method to evaluate utility and localization errors.

Fig. 3.4 illustrates the three main stages in the experimental design, namely, Exhaustive Fingerprint Collection, Path Planning with Algorithms and Localization Performance Test. Specifically,

i) Exhaustive Fingerprint Collection The purpose of this step is to exhaustively collect raw fingerprints in each test area densely and use them to fit a GP regression

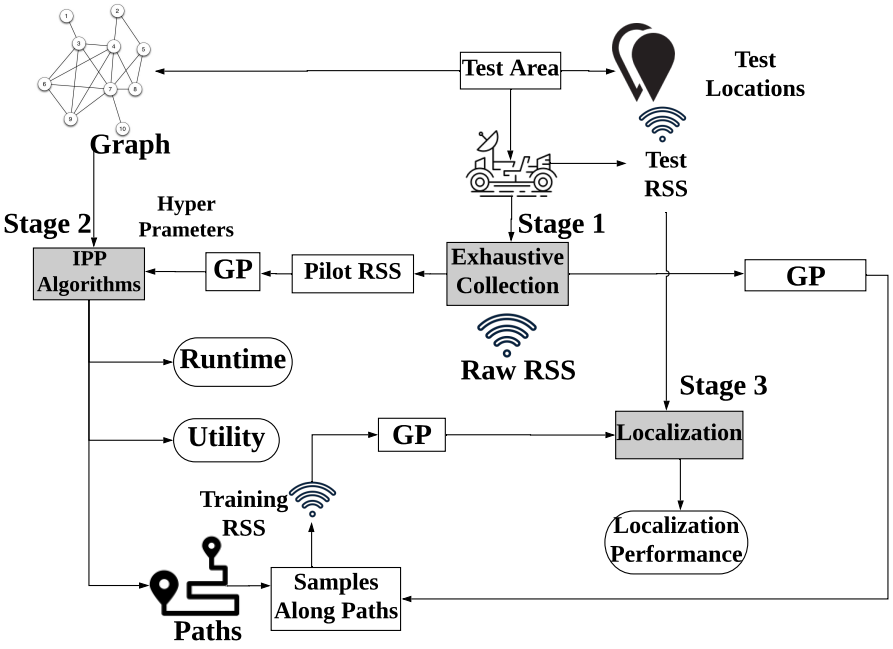


Figure 3.4: Experimental Design

model separately for each access point (AP) in that area. The set of GP models are denoted by $\mathcal{M}$. Later in the Localization Performance Test stage, training fingerprints are sampled from these models. With the densely sampled training data, it is expected that the predicted values can be close to the true values. Additionally, in this stage, another independent set of fingerprints is collected as the test RSS set.

ii) Path Planning For experimental purposes, we take a small subset of the collected fingerprints as pilot data to estimate the hyperparameters of the GPs. The hyperparameters are needed to determine the utility function. Test areas are then discretized into graphs and paths are generated by different algorithms. In the experiments, v_s is set to be identical to v_t, since we expect the robot to return to the start location after finishing fingerprint collection. The sample interval along the selected

paths is set to 0.5m. In Fig. 3.6, the red trajectory gives an example path with evenly sampled locations of the RSS. It is possible to sample more frequently, but doing so implies a lower moving speed.

iii) Localization Performance Test Once the paths are planned, fingerprints can be generated by the fitted GP models $\mathcal{M}$ at sampled locations as illustrated in the red trajectory in Fig. 3.6. These generated fingerprints are used as training data, and localization performance can then be evaluated using the test RSS collected in the first stage.

With such a design, during the localization performance test stage, since the fingerprints are generated by the same models, the only factor that may affect the collected fingerprints is the chosen path. Thus, the two afore-mentioned problems are eliminated.

3.3.2.2 Performance Metrics

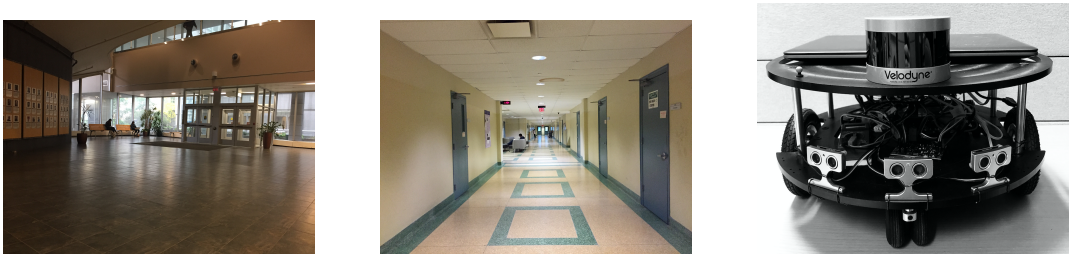
We consider utility, run time and the localization errors as performance metrics. Utility and run time are directly returned from the path planning algorithms.

To evaluate localization errors, we adopt the approach in [64]. Specifically, after paths are planned, fingerprints are sampled from the GP models $\mathcal{M}$ along the paths and utilized as training data. The test area is discretized into a set of uniformly distributed reference locations. For each AP, a GP regression model is fitted based on the training data and used to predict the fingerprint distributions at those reference locations. In the inference stage, given the fingerprints collected at a test location, the target location is predicted to be one of the reference locations that has the maximum probability to generate the fingerprints. Since the test fingerprints are collected by

the robot with known locations, localization error can be computed as the distance between the predicted and ground truth locations.

3.3.2.3 Fingerprint Collection

Two areas were selected for fingerprint collection and experiments in two buildings. The first is approximately 12m wide and 15m long. The second is a corridor and it is 63m long. Fig. 3.5 shows the two areas and the robot used for fingerprint collection. Fig. 3.6 and Fig. 3.7 show the graphs generated from the two areas, respectively. In evaluating localization errors, test locations are selected roughly uniformly across the test areas, as shown by the stars in the figures. Table 3.1 summarizes the settings of the data collection. We assume a small set of pilot data are known in advance as in [8, 36].



(a) Area One (b) Area Two (c) The Robot

Figure 3.5: Test areas and the robot for fingerprint collection.

A Wi-Fi interface card is installed on the robot to collect raw fingerprints. The robot is a moving platform equipped with a Velodyne Lidar allowing it to perform Simultaneous Localization And Mapping (SLAM), and it is manually driven to cover the available area. Simultaneously Wi-Fi RSS measurements are recorded and labeled with the locations where they were collected, as determined by the robot's SLAM algorithm. Location errors of the robot from the SLAM algorithm are around 15cm,

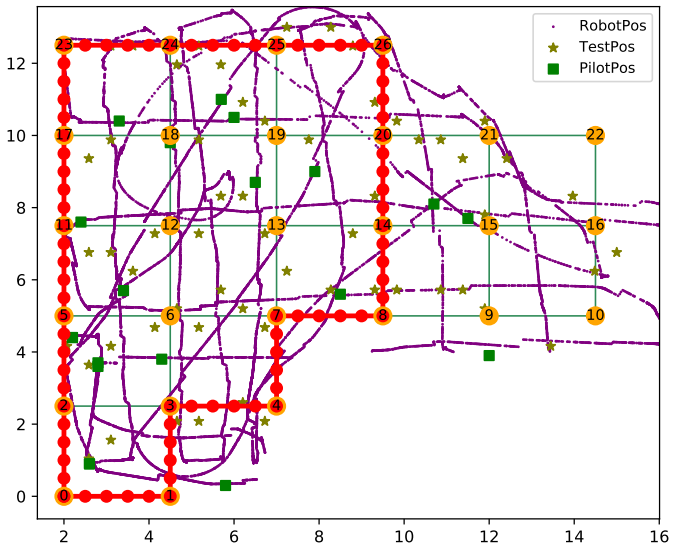


Figure 3.6: The graph generated from Area One. The X and Y axis represent the size of the area in meters. This area is discretized and represented as a grid graph. The purple lines show the robot's trajectories. The squares represent the pilot data locations, and the stars represent the test locations. Based on this grid graph, the red trajectory shows an example path with a budget of 40 meters, and each red dot represents a RSS sample location from the fitted GP (refer to Section 3.3.2.1).

depending on the number of distinct features present in the test areas.

Compared with collecting data by a human holding a smart-phone, the use of a robot has three advantages. Firstly, a human body can block the WLAN radio signal and cause a significant decay [23]. Secondly, the moving speed of the robot can be controlled more precisely. Lastly, location labeling errors are quite small with the help of SLAM. In contrast, when data are collected by a user holding a smart-phone, the location labels are either estimated manually or generated by leveraging a step counter [64], which are less accurate.

Since the utility (MI) of a path only depends on the hyperparameters estimated from the pilot data. During the path planning stage, given the hyperparameters,

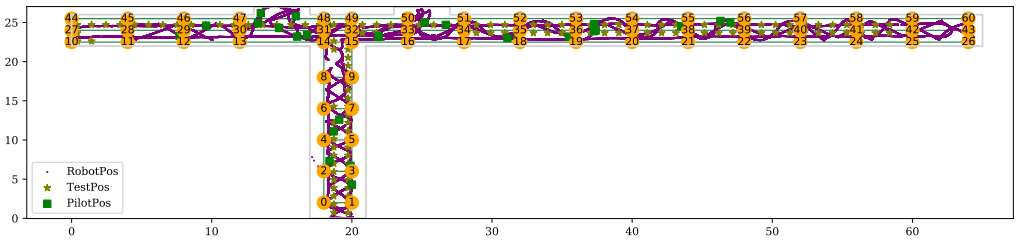


Figure 3.7: The graph generated from Area Two. Similar to Fig. 3.6, the squares represent the assumed pilot data locations, the stars represent the locations for localization error evaluation and the purple lines show the rover's trajectories.

Table 3.1: Information of collected data in the two areas

	Area One	**Area Two**
# of APs observed	11	18
# of RSS collected	29115	199590
# of Vertices in the graph	27	61
# of Pilot Locations	20	30
# of Test Locations	58	131
Localization Error (m, all RSS)	3.8	2.06

the run time and utility of IPP algorithms are independent of actual fingerprint measurements along the paths.

3.3.3 Results

3.3.3.1 Choice of GA parameters

GA has some parameters that need to be configured, including the population size *pop_size*, the tournament size *tn*, the percentage of offspring that needs mutation *mu_percent* and the number of generations to iterate *g_num*. Here, we focus on the population size and the number of generations since the run time of GA is mainly determined by these two parameters. We run GA with different *pop_size* on the graph derived from Area One for ten generations. Fig. 3.8 shows the best fitness

during each generation and the run time of different population sizes. As can be seen from the figure, a larger population size is more likely to achieve a higher utility, especially during the early generations. On the other hand, the run time increases as the population size becomes larger. A larger population pool implies that genetic operators such as cross over and mutation need to consider more individuals, and thus lead to a higher computation cost. In the experiments, we consider population sizes 100 and 200, and found that they experimentally strike a good trade-off between utility and run time. The parameters of GA for all experiments are listed in Table 3.2.

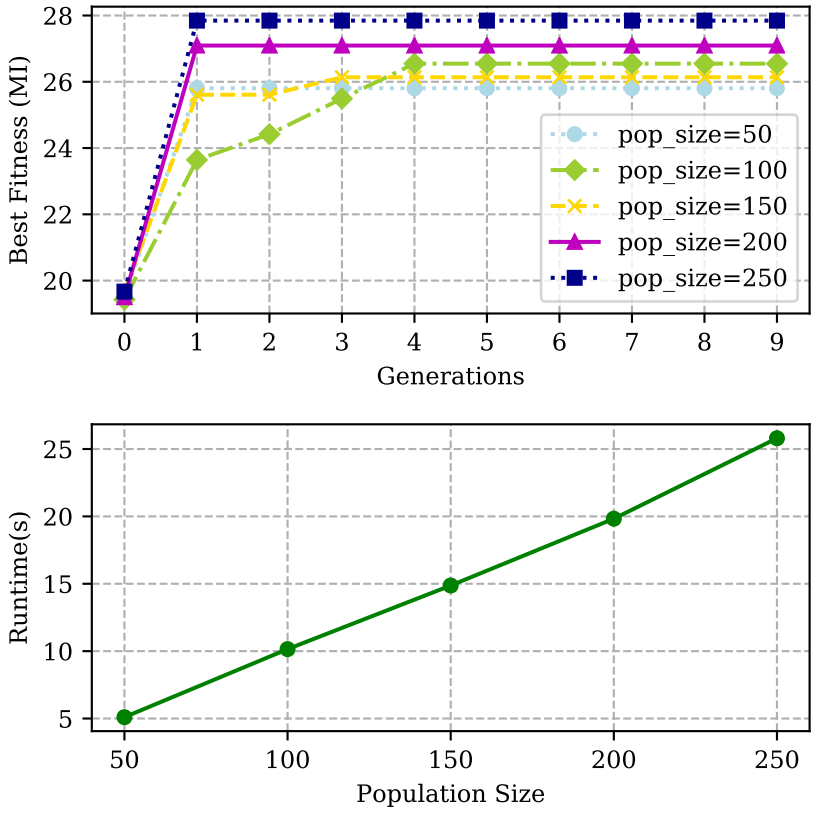


Figure 3.8: The utility of the GA with different population size and the corresponding run time. In this experiment, tournament size is set to ten and 90% offspring are mutated.

Table 3.2: GA parameter setting

Parameter	**Value**
population size	100 and 200
tournament size	10
offspring mutation percentage	90%
total generations	5

3.3.3.2 Relation between MI and Localization Errors

The premise of IPP is that the data collected along paths with higher utility will lead to models with lower localization errors. To validate this assumption, we conduct experiments to investigate the relation between utility and localization errors. Specifically, we randomly sample a number of paths with different fitness scores from the GA's population in different generations, and then sample fingerprints along the paths from $\mathcal{M}$. Localization models are trained and localization errors are evaluated on the test locations.

Fig. 3.9 illustrates the utility and the corresponding localization error at the two areas under different budgets. It can be seen that the localization performance in Area One tends to be better than that in Area Two. In both areas we observe that when the utility increases, the localization error tends to decrease, although it is not always true. This is because localization errors also depend on actual RSS measurements and the number of available access points, which are not modeled in the path utility. Intuitively, for two locations to be distinguishable, the distributions of fingerprints should have distinctive means and little overlap. Consider an extreme example when the distributions of Wi-Fi RSS are the same everywhere. In this case, location errors are unbounded despite the fact that more samples always increase the MI of paths. However, in practical scenarios, as illustrated in Fig. 3.9, MI is an effective metric for

planning paths to collect informative fingerprints.

In addition, it can be seen in Fig. 3.9 that at higher utilities (with more measurements or longer paths), the improvements in localization errors become saturated. This is expected due to the high variance of fingerprints and can be analyzed through Cramer-Rao bounds [14].

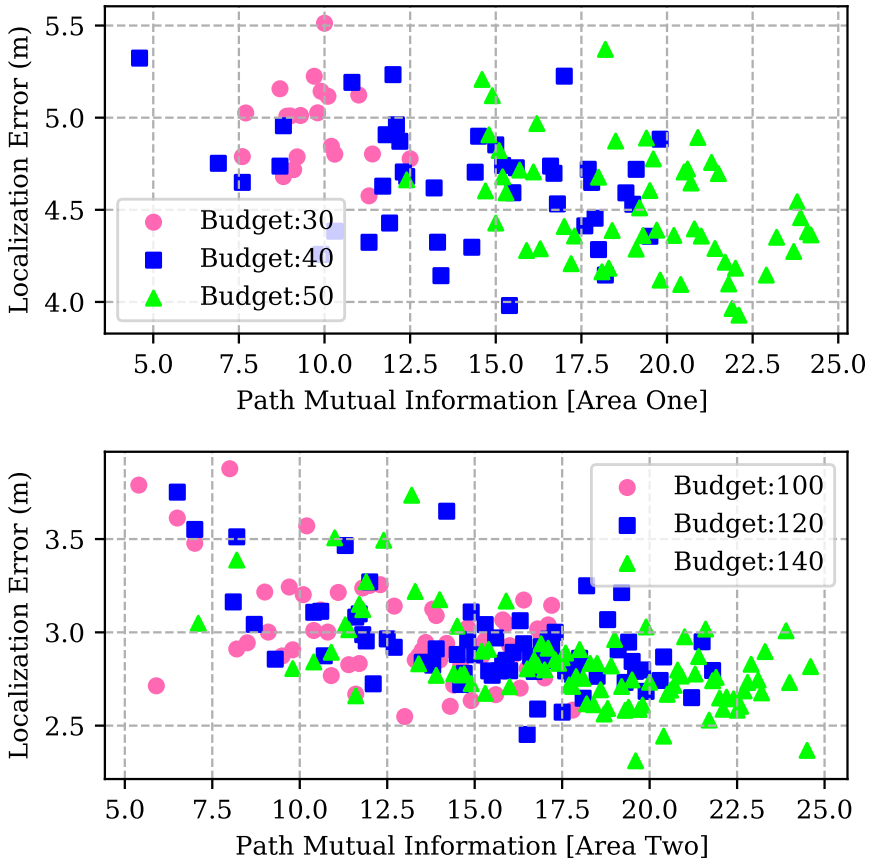
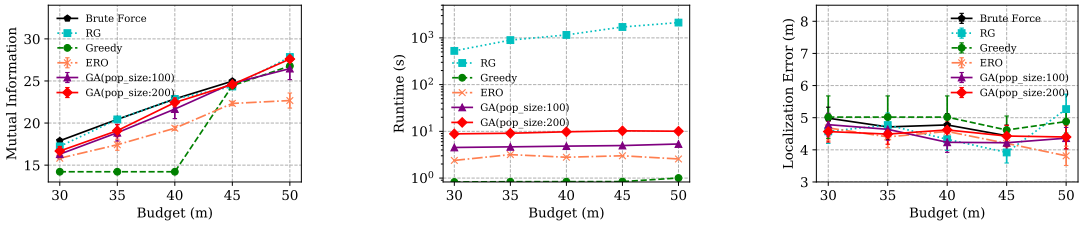


Figure 3.9: Relation between utility and localization error in the two areas. In Area One, the Pearson correlation coefficients for budget 30, 40 and 50 are -0.12, -0.28 and -0.43, respectively. In Area Two, the Pearson correlation coefficients for budget 100, 120 and 140 are -0.51, -0.56 and -0.56, respectively.

3.3.3.3 Performance

We only managed to run QP-RG in Area One, while in Area Two the algorithm failed to return after two hours due to a larger graph size and budget. As a result, we do not run RG on large graphs. For ERO and GA, since they are randomized algorithms and each run may give different results, we run five rounds under each budget and take the average. Fig. 3.10 and Fig. 3.11 show the results in Area One and Area Two, respectively.

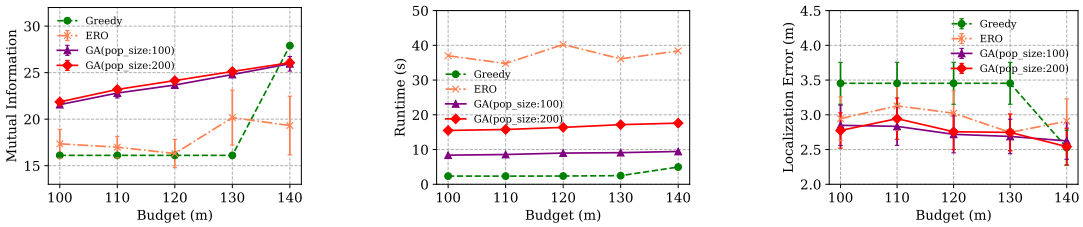
In order to find the path with the optimal utility, we also implemented a brute force approach by enumerating all the paths constrained by the start vertex, terminal vertex and budget. The brute force approach is executed for at most 72 hours for each task. In Area One, the brute force approach successfully finds the optimal path within 72 hours when the budget is set to 30, 35, 40 and 45.



(a) Utility (b) Runtime (c) Localization Error

Figure 3.10: Comparison of different algorithms in Area One under different budget constraints. The brute force approach failed to give the result in 72 hours when the budget is 50.

Utility Fig. 3.10(a) and Fig. 3.11(a) give the utility achieved by different path planning algorithms in the two areas. We first notice that Greedy sometimes can achieve a high utility, and sometimes cannot. For example, in Area One, the utility obtained by Greedy stays the same when the budget increases from 30 to 40, and



(a) Utility (b) Runtime (c) Localization Error

Figure 3.11: Comparison of different algorithms in Area Two under different budget constraints.

similar patterns can be found in Area Two. In contrast, the utility attained by other algorithms grow monotonically with budgets in Area One. In Area Two, the utility of ERO tends to fluctuate. Since ERO is extended from RO [4], it is not stable, particularly when the budget is insufficient. Most candidate solutions are likely to exceed the budget constraints when nodes are randomly added or deleted. GA shows a promising performance in both two areas, and in Area One its utility is quite close to RG and the brute force approach.

Runtime The run time of RG increases from 527.3 seconds to 2124.7 seconds when the budget increases from 30 to 50 in Area One. It is quite sensitive to the budget since the approach exhaustively searches the intermediate vertices and budget splits. In Area One, Greedy is quite fast finishing in less than 2 seconds, and ERO has a faster run time than GA. However, since both Greedy and ERO invoke a Steiner TSP solver, when the size of the graph increases, their run time increases significantly. As can be seen from Fig.3.11, the run time of ERO increases to 40 seconds in Area Two since the corresponding graph has 61 vertices (27 vertices in Area One). The run time of GA is mainly determined by the population size and the number of generations as shown in Fig. 3.8. The run time of the brute force approach is listed in Table 3.3.

When the budget rises, the run time increases dramatically, and thus for the budget of 50 it failed to return the result within three days.

Table 3.3: Run time of brute force for Area One

Budget	**Runtime (s)**
30	111.1
35	1365.7
40	37708.5
45	133456.5

Localization Errors From Fig. 3.9, we observe that the localization error generally decreases when the budget increases. In most cases we see GA has smaller localization errors because it achieves higher utility compared to other schemes. As previously discussed in Section 3.3.3.2, there is generally a negative correlation between utility and location errors, though there are exceptions due to variability of WiFi signals in space.

3.3.4 Discussion

IPP is an NP-hard problem, and there are no benchmark instances with known optimal solutions. It is challenging to find the optimal solution, particularly on a large graph with a sufficient budget, since the potential solution space is extremely large. We show that GA is flexible and well suited for solving IPP, since GA can be configured to adapt to different problem scales. Increasing population size and the number of generations promise a solution with more utility, but lead to a longer time. The run time of Greedy and ERO increases significantly when the graph size increases due to the TSP solvers.

Another advantage of GA is that it can be easily parallelized. A primary time consuming operation of IPP is to evaluate $f_{\mathcal{D}}(\mathcal{P})$. In evaluating $f_{\mathcal{D}}(\mathcal{P})$, calculation of the conditional entropy of the vertex locations based on the sample locations, involves a matrix inversion operation. GA maintains a population and for each individual $f_{\mathcal{D}}(\mathcal{P})$ needs to be evaluated. This task can be parallelized with multi-threading. In our experiments, we did not use parallelization to improve the speed.

3.4 Conclusion

In this chapter, we formulated the IPP problem to plan paths for spatial data collection. We proposed a Greedy algorithm and a Genetic Algorithm to plan paths in a graph. Experiments on Wi-Fi fingerprint collection were conducted to compare the performance of these algorithms. Localization results showed that paths with a higher utility are more likely to achieve lower localization errors.

Chapter 4

Learning based Path Planning for Flexible Budgets

In the previous chapter, we presented two IPP algorithms to maximize data utility when the budget of a robot is known and fixed. In this chapter, we focus on cases when budgets are flexible and not fixed. For instance, when robots are not fully charged, their battery performance degrade, new robots introduced, or there are task deadlines. These cases will lead to different budgets. One possible but inefficient solution is to repetitively apply the same algorithms whenever a budget change occurs.

A more efficient and desirable solution is to avoid searching from scratch given a changed budget. Since the topology of the target area usually does not change, paths of similar budgets may have common characteristics. This implies that a learning based path planning solution can be possible. Based on this insight, we define the informative path planning problem as a sequential decision making process and develop a reinforcement learning IPP framework, which is able to predict informative paths efficiently when budget changes.

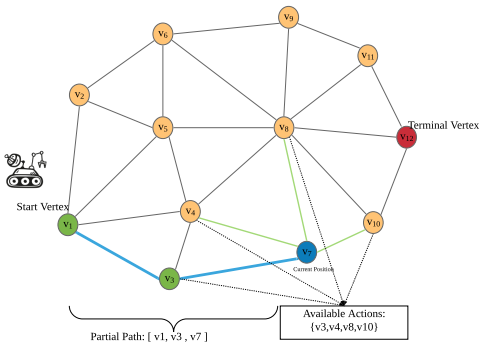


Figure 4.1: Sequential Decision Process for IPP

4.1 Sequential Decision Making Problems

It is straightforward to model IPP as a sequential decision problem. Specifically, suppose an agent is exploring informative paths in G from v_s to v_t, with a budget B. As shown in Fig. 4.1, we denote the vertices that have been traversed by the agent up to the k-th step as the partial path $\bar{\mathcal{P}}_k$. Initially, $\bar{\mathcal{P}}_1 = [v_s]$ since the agent is at the start location. In subsequent steps, the agent decides which vertices to travel to, and the candidates are the adjacent vertices of the last vertex in $\bar{\mathcal{P}}_k$, i.e., the current location. Once the the next vertex is selected, the agent moves there and obtains an immediate step reward signal. This decision process repeats until the budget is exhausted or the agent successfully reaches v_t. Each round of path exploration is known as an *episode*.

Formally, the decision process can be described by a Markov Decision Process (MDP) [102] $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$, where

- $\mathcal{S}$ is a finite set of states, which includes information such as the partial path $\bar{\mathcal{P}}$ and budget status.
- $\mathcal{A}$ is a finite set of actions which is equivalent to $\mathcal{V}$. However, for each step,

only a limited subset of actions (adjacent vertices) are available.

- $\mathcal{T}$ is the state transition function[1] defined as $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. The main effect of state transition here is that the partial path extends by one vertex.
- $\mathcal{R}$ is the reward function defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathbb{R}$ is a real value representing the reward perceived. The reward signal encourages the agent to explore more informative paths, which will be discussed later.

To solve the MDP with RL, we define a decision making policy as $\pi(s) : \mathcal{S} \to \mathcal{A}$. At each time step t, the agent takes an action $a_t = \pi(s_t) \in \mathcal{A}$ and perceives a reward r_t. The objective is to find a policy π such that the total future reward

$$R_t = r_t + \gamma r_{t+1} + ... + \gamma^{T-t} r_T \tag{4.1.1}$$

is maximized, where $\gamma \in [0, 1]$ is a discount factor about the priority of step reward and T is the last action time.

4.2 IPP Solution with RL

In this section, we present a reinforcement learning based IPP solution. The problem is still formulated on a graph as in Section 3.1 from Chapter 3. However, we focus on cases where budgets are unknown in advance. In other words, given an IPP formulation of a five-tuple $\langle G, v_s, v_t, f(\mathcal{P}), B \rangle$, B could vary. We propose a reinforcement learning framework to learn a model which could predict informative paths given any budgets. As a result, compared with naive solutions which apply the path planning

[1] In this work we consider deterministic transitions.

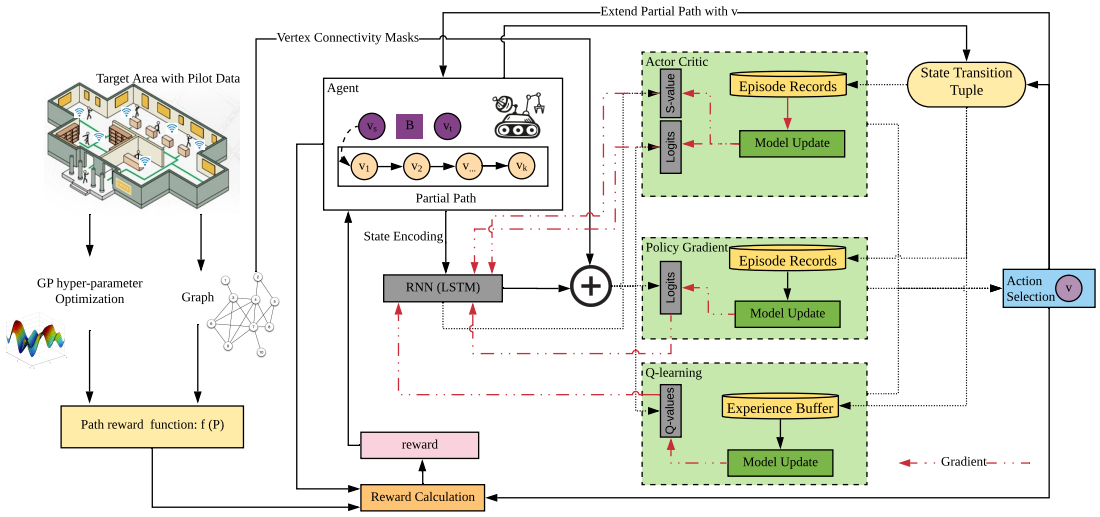


Figure 4.2: Solution overview with Reinforcement Learning.

algorithms from scratch, the RL based path planning solutions can improve efficiency significantly.

4.2.1 Solution Overview

Fig. 6.1 shows the overall architecture of the solution. One unique characteristic of IPP is that the reward of taking an action (visiting a vertex) depends on all the previous actions. For instance, re-visiting a vertex that has been visited before is supposed to gain less reward than visiting a new vertex. Thus, we incorporate the partial path into state encoding and utilize a recurrent neural network (RNN) in the architecture. To filter out vertices that are not adjacent to the current position v_k at each step, a connectivity mask vector $\mathbf{m}_1$ is added before the final output. Specifically,

$\mathbf{m}_1$ has a length of $|\mathcal{V}|$ and is defined as

$$\mathbf{m}_1[i] = \begin{cases} 0 & v_i \in \text{adj}(v_k) \\ -\infty & \text{else} \end{cases}, \tag{4.2.1}$$

where $\text{adj}(v_k) = \{v \in \mathcal{V} : (v_k, v) \in \mathcal{E}\}$ represents the adjacent vertices of v_k. Due to $\mathbf{m}_1$, the Q-values (for Q-learning) or probabilities (in the policy network) of the non-adjacent vertices are negligible by the policy.

Depending on how to interpret the output from the network, different kinds of RL methods can be applied, such as Q-learning, policy gradient and actor-critic. The respective models can be trained accordingly using state transition tuples. To ensure the agent can reach v_t within the budget, a novel action selection mechanism is designed, which may differ depending on the RL method adopted. The reward of the selected action can be calculated using $f(\mathcal{P})$. Next, we give a detailed description of the key components.

4.2.2 State Encoding

Fig. 4.3 illustrates the state encoding scheme based on the trajectory of the agent. A state $\langle \bar{\mathcal{P}}_k = [v_1, v_2, ..., v_k], v_s, v_t, B \rangle$ is encoded as a $5 \times k$ matrix that can be fed into a RNN. For each vertex $v_i \in \bar{\mathcal{P}}_k$, the corresponding RNN cell is $(x_i, y_i, x_t, y_t, \mathrm{b}_k)^T$, where (x_i, y_i) represents the agent's location, (x_t, y_t) represents the expected terminal location, and

$$\mathrm{b}_k = B - \sum_{i=2}^{k} c(v_{i-1}, v_i) \tag{4.2.2}$$

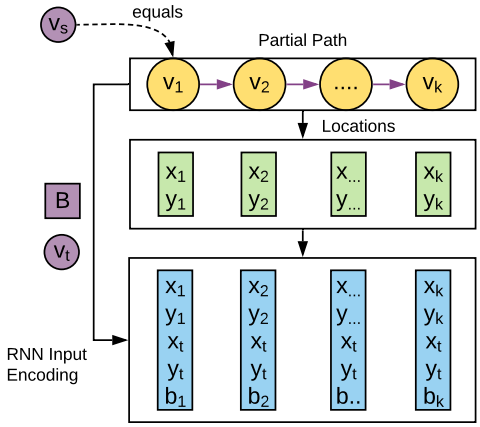


Figure 4.3: Input encoding for the states.

is the remaining budget. Since v_s equals to v_1 in $\bar{\mathcal{P}}$, it is not encoded separately like v_t.

4.2.3 Action Selection

One naive action selection strategy is to select the next action vertex from the adjacent vertices without any constraint, until the budget is exhausted. However, it cannot guarantee that the agent will reach v_t. In RL, one may signal a penalty reward when the agent fails to reach v_t, and expect the agent to gain the knowledge of valid paths through reward signals. However, this simple solution will lead to a large number of invalid paths that do not terminate at v_t and thus wastage computation resources. Next, we present our *constrained action selection* strategy that effectively reduces the search space.

First, given $\bar{\mathcal{P}}_k = [v_1, v_2, ..., v_k]$, we define *available* actions as

$$\mathcal{A}(\bar{\mathcal{P}}_k) = \text{adj}(v_k) = \{v \in \mathcal{V} : (v_k, v) \in \mathcal{E}\}. \tag{4.2.3}$$

Among $\mathcal{A}(\bar{\mathcal{P}}_k)$, we further determine *valid* actions that have chances to reach v_t within B as

$$\mathcal{A}'(\bar{\mathcal{P}}_k) = \{v \in \mathcal{A}(\bar{\mathcal{P}}_k) : c(v_k, v) + \text{LCP}(v, v_t) \leq \text{b}_k\}, \tag{4.2.4}$$

where $\text{LCP}(v, v_t)$ denotes the cost of the Least Cost Path from v to v_t and can be solved using the Dijkstra algorithm. It can be easily seen that if the agent selects an action from $\mathcal{A}'(\bar{\mathcal{P}})$ for every single step, it will reach v_t eventually within the budget constraint.

The action selection module needs to be customized for the specific RL method adopted. We consider two types, namely off-policy learning and on-policy learning. For off-policy learning such as Q-learning, actions are usually selected with the ϵ-greedy policy. The output from the RNN can be interpreted as Q-values, denoted by vector $\mathbf{Q}_{\mathcal{V}}$. Actions can then be selected as

$$a(\bar{\mathcal{P}}_k) = \begin{cases} \text{random } v \in \mathcal{A}'(\bar{\mathcal{P}}_k) & \text{with probability } \epsilon \\ \arg\max_{v \in \mathcal{A}'(\bar{\mathcal{P}}_k)} \mathbf{Q}_{\mathcal{V}}[v] & \text{with probability } 1-\epsilon \end{cases}. \tag{4.2.5}$$

For on-policy learning such as policy gradient, the output of the neural network can be generally interpreted as the probability distribution among actions, and actions are sampled from this distribution. To incorporate the extra constraint information into the distribution directly, we interpret the output from the RNN as the **logits** vector, which is the output tensor without applying the softmax operator, and define another mask vector as

$$\mathbf{m}_2[i] = \begin{cases} 0 & v_i \in \mathcal{A}'(\bar{\mathcal{P}}_k) \\ -\infty & \text{else} \end{cases}. \tag{4.2.6}$$

As such, the final distribution of actions can be computed as

$$\pi(a \mid \bar{\mathcal{P}}_k) = \text{softmax}(\mathbf{logits} + \mathbf{m}_2) \tag{4.2.7}$$

and actions can be sampled from this distribution. Invalid vertices would not be selected due to their zero probability resulted from negative infinity.

4.2.4 Environment and Reward Mechanism

An environment for RL is designed based on the graph. For each action, the agent receives an immediate reward signal and extends the partial path to the next vertex. The reward of taking an action $a \in \mathcal{A}'(\bar{\mathcal{P}}_k)$ is calculated as

$$r(\bar{\mathcal{P}}_k, a) = f([\bar{\mathcal{P}}_k, a]) - f(\bar{\mathcal{P}}_k). \tag{4.2.8}$$

Therefore, the reward in each single step from a path $\mathcal{P}$ adds up to the total reward of the path since

$$r(\mathcal{P}) = f(\mathcal{P}) = \sum_{k=1}^{|\mathcal{P}|-1} r(\bar{\mathcal{P}}_k, a). \tag{4.2.9}$$

This is exactly the optimization goal, i.e., to maximize the informativeness of the path. Finally, if the action equals to v_t, the environment transits to the terminal state; otherwise, the agent can repeat the step to extend the partial path.

4.2.5 Reinforcement Learning Methods

The proposed architecture supports a variety of RL methods such as Q-learning [75], policy gradient [97] and actor-critic [35], with minor adaptations. Readers interested

in RL can consult [96] for more information. For completeness, we provide the basic steps of training typical RL models for IPP.

4.2.5.1 Q-learning

For Q-learning, the RNN is interpreted as representing a function $\mathcal{Q}_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, with $\mathcal{Q}_\theta(s, a)$ being the total discounted future reward by taking action a from state s. The policy given $\mathcal{Q}_\theta$ can then be induced as $\pi(s) = \arg\max_a \mathcal{Q}_\theta(s, a)$.

At each time step t, let the state transition tuple be $\langle s_t, a_t, r_t, s_{t+1} \rangle$, namely, upon taking action a_t from state s_t, the agent gets a reward r_t and transits to s_{t+1}. With the transition tuples generated from each episode, the network can be optimized in an iterative way by minimizing the temporal difference using a loss function defined as

$$\ell(\theta) = \Big(\mathcal{Q}_\theta(s_t, a_t) - (r_t + \gamma \max_a \mathcal{Q}_\theta(s_{t+1}, a)) \Big)^2. \tag{4.2.10}$$

To stabilise the training process, advanced techniques have been proposed in recent years like DDQN [101], Prioritized Experience Replay [89], which can be adopted directly in the solution.

4.2.5.2 Policy Gradient

Q-learning models the Q-values to guide decision making indirectly, while policy gradient directly models the action probability. Let the probability distribution of actions given the states be $\pi_\theta(a \mid s)$, and we denote one episode of length T as $\tau = (s_1, a_1, r_1, ..., s_T, a_T, r_T)$. The expected total reward is

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[r(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t=1}^{T} r_t], \tag{4.2.11}$$

where $\pi_\theta(\tau) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t \mid s_t)$ is the probability of τ given π_θ. For episode τ, the gradient of $J(\theta)$ is then given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[(\sum_{t=1}^{T}\nabla_\theta \log\pi_\theta(a_t \mid s_t))(\sum_{t=1}^{T} r_t)]. \tag{4.2.12}$$

The reinforce algorithm [96] is a policy gradient approach that uses (4.2.12) to update the model. Usually, policy gradient methods update the model when a whole episode is finished. Note that in implementation, the output from the RNN is not the probability, but the logits, since the final probability of actions needs to incorporate another mask vector as discussed in Section 4.2.3.

4.2.5.3 Actor-Critic

One disadvantage of policy gradient method is that policy estimators suffer from a high variance [113], and one has to wait until the end of an episode to update the model. Actor-critic methods address this problem by approximating the gradient with

$$\nabla_\theta J(\theta) \approx \sum_{t=1}^{T}\nabla_\theta \log\pi_\theta(a_t \mid s_t)A(s_t, a_t), \tag{4.2.13}$$

where $A(s_t, a_t)$ is the critic used to assess the utility of the action. One typical option is the advantage actor-critic (A2C) [76],

$$A(s_t, a_t) = r_t + V_\phi(s_{t+1}) - V_\phi(s_t), \tag{4.2.14}$$

where V_ϕ is another neural network to estimate the state value and can be trained by minimizing the temporal difference of $(r_t + V_\phi(s_{t+1}) - V_\phi(s_t))^2$. The original network is called the actor network. The state value (S-value) network only depends on the

state, and does not need the vertex connectivity mask as shown in Fig. 6.1. Both networks share the RNN component for better efficiency.

4.2.6 Model Training and Path Inference

4.2.6.1 Model Training

Although the exact budget B is unknown in advance, we assume it falls in a range depending on the target system. The basic idea is to train a model using different budgets such that it could make predictions for unseen budgets. To differentiate from the case when B is fixed, we denote a set of possible budgets sampled from the range as $\mathcal{B}$, which will be used as *training budgets*.

The model training procedure is described by Algorithm 5. In each episode, a specific budget B is randomly sampled from $\mathcal{B}$ and used for training. Meanwhile, for every E episodes, a snapshot of the model is captured and stored in a snapshot set $\mathcal{M}$, which is the final output of the training procedure. Note that Algorithm 5 only outlines the general steps involved and updates the model after each episode. For Q-learning or actor-critic training, models can be updated step-wise. Other details such as training a value-network for actor-critic are omitted.

4.2.6.2 Path Inference

In the model training stage, a collection of snapshots of the model are captured. Since the model is trained based on the transition tuples from different budgets, different snapshots can have different inference performance with regard to a specific budget. In addition, it is known that RL models can be unstable and may not converge to the optimal policy when neural networks are used to approximate the value functions or

Algorithm 3: Model Training Procedure

Input : $\langle G, v_s, v_t, f(\mathcal{P}), \mathcal{B} \rangle$, snapshot period E
Output: model snapshot set $\mathcal{M}$

choose and instantiate a RL model M
initialize $\mathcal{M} = \{\}$
for episode $e = 1, 2, ...$ **do**
    randomly sample $B \in \mathcal{B}$
    initialize $\bar{\mathcal{P}} = [v_s]$
    episode records = $\{\}$
    for step $t = 1, 2, ..., T$ **do**
        selection action a_t with (4.2.5) or (4.2.7)
        execute a_t and calculate reward r_t with (4.2.8)
        add $\langle \bar{\mathcal{P}}, a_t, r_t, [\bar{\mathcal{P}}, a_t] \rangle$ to episode records
        $\bar{\mathcal{P}} = [\bar{\mathcal{P}}, a_t]$
    end
    encode the states with episode records, v_t, B
    /*refer to section 4.2.2*/
    calculate the loss of the specific RL model M
    update M with gradient descent
    if e mod $E = 0$ **then**
        take a snapshot of M and add it into $\mathcal{M}$
end
return $\mathcal{M}$

policies. Thus, we store multiple snapshots of the model during training.

In the path inference stage, given the specific budget B, all the snapshots $\mathcal{M}$ are loaded and $|\mathcal{M}|$ paths are generated greedily using these snapshots. The path with the maximum reward is the final output path. The path planning procedure is outlined in Algorithm 6. Since the models are trained with different budgets, at inference time, the same models can be utilized given different budgets.

Algorithm 4: Path Inference Procedure

Input : v_s, v_t, B, model snapshot set $\mathcal{M}$
Output: best path $\mathcal{P}^*$

initialize a path set $\mathcal{P}_{\mathcal{M}} = \{\}$
for $M \in \mathcal{M}$ **do**
  create an environment with v_s, v_t, B
  initialize $\bar{\mathcal{P}} = [v_s]$
  for step $t = 1, 2, ..., T$ **do**
    select a_t by the max Q-value (or probability)
    execute action a_t and get reward r_t
    if episode terminates **then**
      add $\bar{\mathcal{P}}$ to $\mathcal{P}_{\mathcal{M}}$
    else
      $\bar{\mathcal{P}} = [\bar{\mathcal{P}}, a_t]$
  end
end
return $\mathcal{P}^* = \arg\max_{\mathcal{P} \in \mathcal{P}_{\mathcal{M}}} f(\mathcal{P})$

4.3 Experimental Evaluation

In this section, we first evaluate the impacts of constrained action selection in learning efficiency and inspect the convergence of different algorithms. We then investigate the impact of training budgets and different RL methods. Finally, we compare the performance of the proposed algorithms with other IPP algorithms in terms of utility

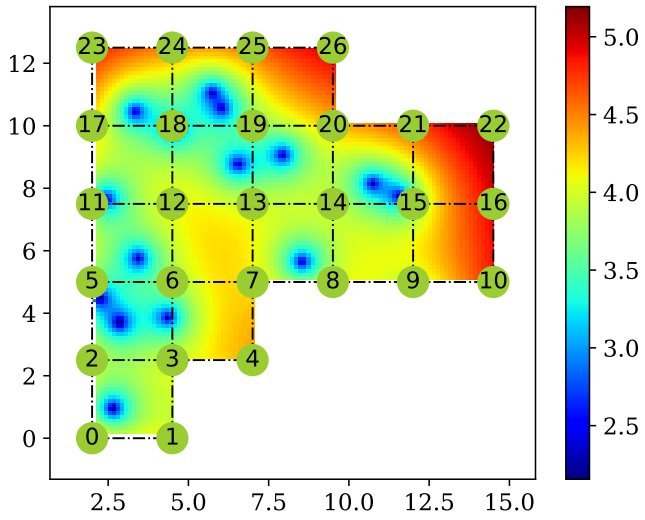


Figure 4.4: Uncertainty heat-map of Area One. The size of the whole area is approximately 12m * 13m. The X and Y axes show the dimensions in meters, and the color represents the uncertainty (entropy) of the predicted signals by fitting a GP with the pilot data. The grid graph has 26 vertices and are indexed with integers.

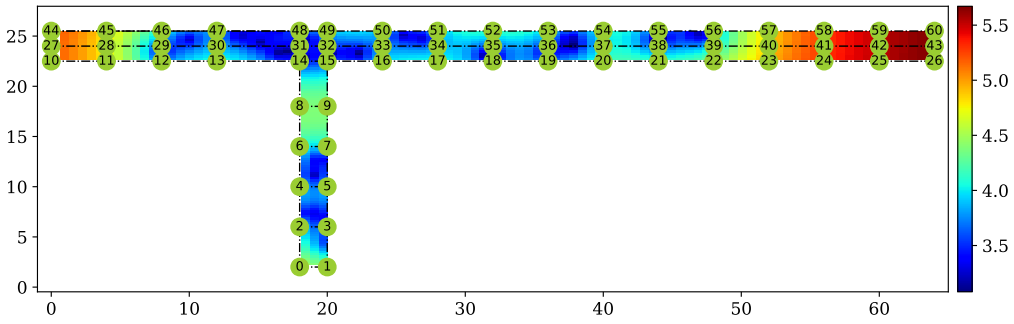


Figure 4.5: Uncertainty heat-map of Area Two. This area is a "T" shape corridor, with 25m in height and 64m in length. The graph has 61 vertices as shown by the green circles.

and path planning efficiency.

4.3.1 Graph Setting and Implementation

We use the same two areas as in the previous chapter for evaluation purposes. The uncertainty heat-map of Wi-Fi signals in the two areas are illustrated in Fig.4.4 and Fig.4.5, respectively.

The RL environment is implemented in Python, which tracks all the necessary information for state encoding and reward calculation, such as partial paths, remaining budgets and graph structures. To accelerate computation, the shortest paths between all pairs of vertices are calculated and stored in advance, since the agent needs access to the shortest paths at each step to filter out invalid actions. The APIs of the environment are similar to those in the OpenAI Gym[2], a reinforcement learning toolkit with a variety of environments.

The neural network is implemented in PyTorch, where each RNN cell is an LSTM unit. We adopted a one-direction one-layer RNN structure, with a hidden size of 128. After the last RNN cell, a linear layer is used to produce the final output. During training, the learning rate is set to 0.0001, and gradient clipping is used to avoid gradient explosion [116]. Reward discount factor γ is set to 0.9, and a snapshot of the model is taken every 1000 episodes.

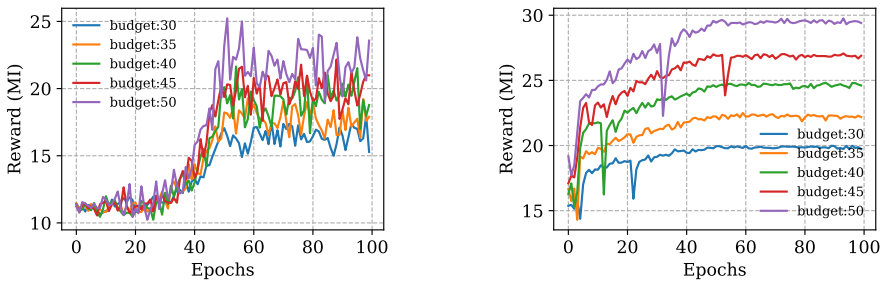
4.3.2 Comparison with Unconstrained Action Selection

In addition to the proposed constrained action selection, an alternative simple action selection scheme is implemented that considers all adjacent vertices as potential next way-points as mentioned in Section 4.2.3.

We trained a double Q-learning model [101] with prioritized experience replay [89]

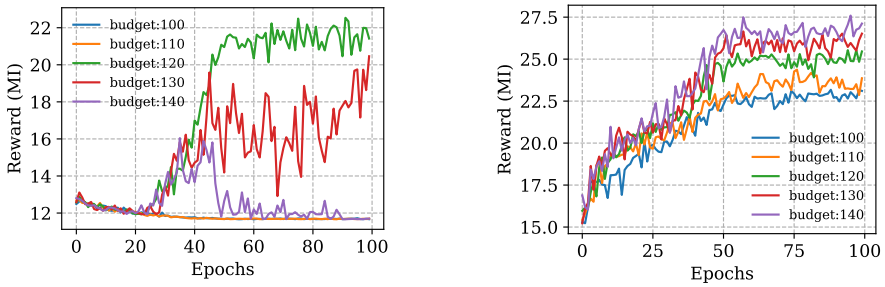
[2]https://gym.openai.com

using the two action selection schemes. In this experiment, we fix the training budget at a specific value to compare the convergence property. Fig. 4.6 and Fig. 4.7 show the average episode reward over the learning process in Area One and Two, respectively.



(a) unconstrained action selection (b) constrained action selection

Figure 4.6: Average reward per episode with Q-learning in the graph from Area One. The start and terminal vertices are set to 0, so the path forms a tour. Experiments are run for different budgets (maximum distance) with ϵ-greedy policy with $\epsilon = 0.9$ initially and decay to $\epsilon = 0.1$ at the 50th epoch. Each epoch means learning for 50 episodes, and the Y axis shows the average reward. (a) shows the unconstrained action selection scheme and (b) shows the constrained action selection with shortest path.



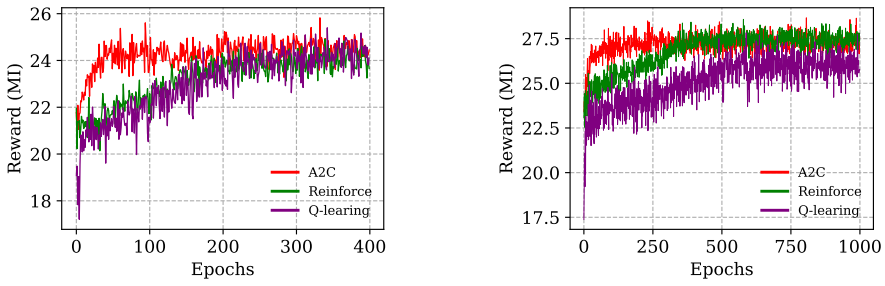
(a) unconstrained action selection (b) constrained action selection

Figure 4.7: Average reward per episode with Q-learning in the graph from Area Two. The parameter settings are similar with Fig.4.6.

Similar to [75], each epoch is defined as 50 episodes of learning, and 100 epochs are run for each budget setting. It can be seen clearly that the constrained action

selection strategy achieves higher rewards (MI) and higher efficiency. During the initial episodes of the unconstrained action selection scheme, the rewards are low since most generated paths are invalid, i.e, failing to terminate at v_t. Thus, the agent will receive a penalty reward signal at the last step. The difference is more significant in Area Two since the graph size is larger than that of Area One. In a larger graph, blind searches have a smaller probability to construct a valid path. As can be seen from Fig.4.7, under some budget setting (e.g., 100, 110, 140) the unconstrained action selection strategy fails to improve in terms of average rewards. In comparison, the constrained action selection strategy shows promising results, with the average reward improving gradually until convergence under different budgets.

4.3.3 Convergence using Different RL methods



(a) Area One (b) Area Two

Figure 4.8: Average reward per episode during training with different RL methods.

Next we inspect the convergence property of different RL methods. Specifically, we evaluate Q-learning, the reinforce algorithm and the advantage actor-critic algorithm. In this experiment, the budget B is not known in advance. The models are trained using a budget set $\mathcal{B}$ as described in Algorithm 5. For Area One, $\mathcal{B}$ is set to be the sequence of [30.5, 31.5, ..., 50.5] and for Area Two it is set to be the sequence of [100.5,

101.5, ..., 140.5]. The goal is to demonstrate that the model is capable of making predictions using the trained model for a specific budget during inference. Since the number of vertices in the first graph is smaller than that of the second graph, the number of epochs trained for the two graphs are 400 and 1000, respectively.

In addition, in later experiments, we will consider two types of paths, namely

- *tour*, the agent is required to return to its start location after data collection, i.e., $v_s = v_t$,
- *non-tour*, the terminal location is different from the start location, i.e., $v_s \neq v_t$.

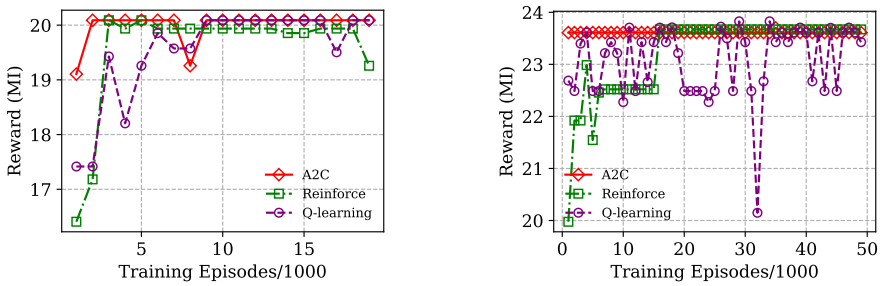
For illustration purposes, for each episode, v_s in Area One is set to vertex 0, and v_t is chosen from vertices [0, 26] in instances of tour and non-tour cases. Thus, in the path planning stage, the model can be used to infer paths given the corresponding v_s and v_t. Similarly, v_s in Area Two is set to vertex 0 and v_t is chosen from vertices [0, 60].

Fig. 4.8 shows the average reward per episode for different RL methods. In Q-learning, episodes are generated using the ϵ-greed policy (4.2.5) with ϵ start at 0.9 and decay to 0.1 gradually; in the other two methods, episodes are generated from the policy network directly (4.2.7). In both areas, it can be seen that A2C has a faster convergence speed. In Area One, the reinforce algorithm shows a similar trend to Q-learning, and A2C performs slightly better after convergence. In Area Two, the reinforce algorithm has a competitive performance compared with A2C after convergence, and both methods outperform Q-learning.

4.3.4 Path Inference Performance

Next we show the performance when the models are utilized for path inference with Algorithm 6. We first show the impact of the number of snapshot and then the effect of training budget selection.

4.3.4.1 Impact of Snapshots

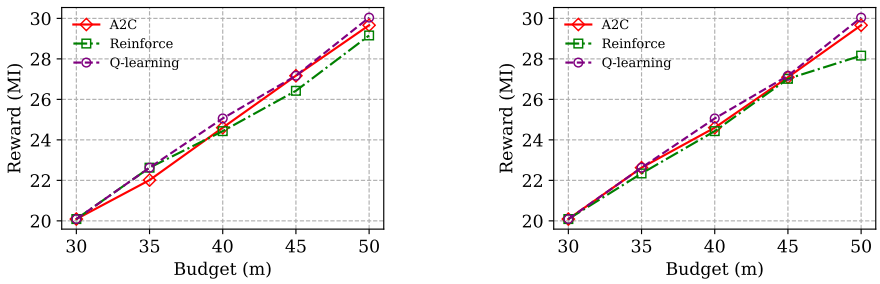


(a) Area One (b) Area Two

Figure 4.9: Rewards of paths inferred using the snapshots. The snapshots are captured every 1000 episode. For Area One, the path specification is given as $v_s = 0, v_t = 0, B = 30$, and for Area Two, it is $v_s = 0, v_t = 0, B = 100$.

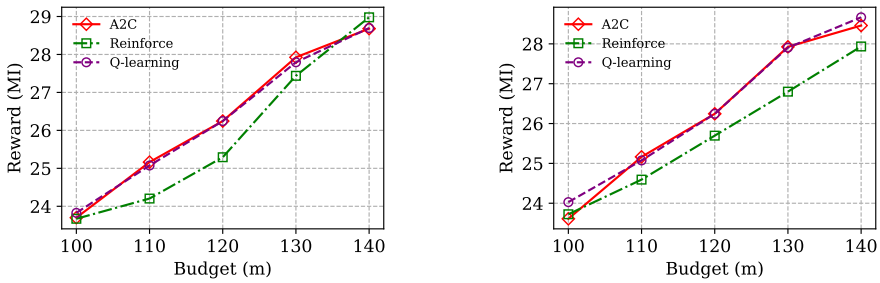
Snapshots captured during training are used to infer paths in the inference stage. Fig. 4.9 shows the final rewards for two budgets in the two areas with different RL methods. For Q-learning and the reinforce algorithm, it can be seen that different snapshots have different inference performances. Since the probability of a path is $\pi_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t)$, even a minor change on the neural network parameters may affect the inferred path significantly. In contrast, we observed that snapshots from A2C are more stable and consistent than those from Q-learning or the reinforce algorithm.

4.3.4.2 Impact of Training Budgets



(a) budget set one (b) budget set two

Figure 4.10: Rewards of paths generated through the models trained with different $\mathcal{B}$ in Area One. The X-axis represent the specific B during inference, and v_s, v_t are set to 0 for a tour case. $\mathcal{B}$ in (a) is [30.5, 31.5, ..., 50.5], and $\mathcal{B}$ in (b) is [30, 31, ..., 50].



(a) budget set one (b) budget set two

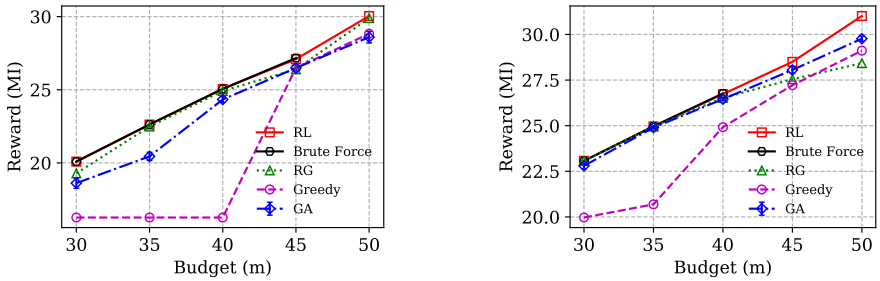
Figure 4.11: Rewards of paths generated through the models trained with different $\mathcal{B}$ in Area Two. The X-axis represent the specific B during inference, and v_s, v_t are set to 0 for a tour case. $\mathcal{B}$ in (a) is [100.5, 101.5, ..., 140.5], and $\mathcal{B}$ in (b) is [100, 101, ..., 140].

Recall that in Section 4.3.3 we trained models with a set of budgets $\mathcal{B}$ that are uniformly sampled from a range since the exact budget B is not known. Intuitively, the selection of $\mathcal{B}$ could affect the inference performance given a specific budget B.

We select another set of training budgets to train different models. These models are used to infer paths and the results are compared with those from the first set of

training budgets. Fig. 4.10 and Fig. 4.11 show the rewards obtained in Area One and Two, respectively. It can be seen that Q-learning and A2C have a similar performance and are less sensitive to the training budgets, while the reinforce algorithm is more sensitive, particularly in Area Two. This could be attributed to that the reinforce algorithm has a high variance in gradients.

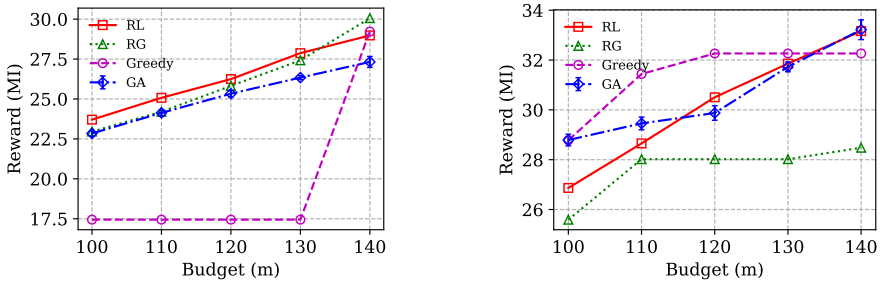
4.3.5 Comparison with Other IPP solutions



(a) tour (b) non-tour

Figure 4.12: Path reward comparison using different algorithms for Area One. The start vertex v_s is set to 0. For (a) v_t is set to 0 for the tour case, while for (b) v_t is set to 26 as a non-tour case.

In this section, we compare the rewards achieved through reinforcement learning with those from other IPP algorithms. As shown from the previous experiments, the three reinforcement learning methods have their own characteristics, and there is no single best method for all budgets during inference. Thus, we *combine* the three RL methods and select the best path inferred among the three methods for a problem instance. Specifically, as can be seen from Fig. 4.9, many snapshots have the same result and thus only a subset of those snapshots are needed. We take snapshots from each RL method every 5000 episode and merge them together. Thus, for Area



(a) tour (b) non-tour

Figure 4.13: Path reward comparison with different algorithms for Area Two. The start vertex v_s is set to 0. For (a) v_t is set to 0 for the tour case, while for (b) v_t is set to 60 as a non-tour case.

One, the total number of snapshots is $3 \times 4 = 12$ since there are 20000 episodes trained. Similarly, in Area Two, the total number of snapshots is 30. Given a path specification, the final path is the one with the maximum reward among the paths generated using these snapshots.

We compare the RL based path planning solution with a Brute Force tree search method, the Recursive Greedy algorithm, the Greedy algorithm and the GA from the previous chapter. For GA, the population size is set to 100. The number of generations for Area One and Two, are 50 and 100, respectively. Due to randomness, we run five rounds of experiments independently and take the average for each budget setting.

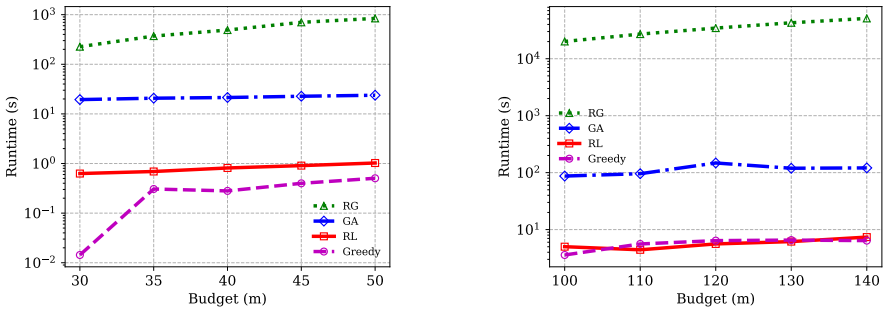
For each area, we consider both the tour case ($v_s = v_t$) and a non-tour case ($v_s \neq v_t$). For each different path specification (v_s, v_t and B), all the other solutions are executed from scratch. In contrast, in the proposed RL based solution, only the path inference procedure (Algorithm 6) is executed using the snapshots captured during training (Section 4.2.6.1).

It can be seen from Fig. 4.12 that RL achieves the best performance compared

with all the other algorithms. When the budget is under 40 meters, the brute force approach managed to return the optimal path with the maximum reward. Thus, the paths found by RL is also optimal, since they coincide with those from the brute force search (note it is overlapped in the figure). The rewards obtained by GA and RG increase monotonically with budgets, while the rewards from the greedy algorithm sometimes remain unchanged even when the budgets increase.

The graph from Area Two contains 61 vertices, with budgets larger than that in Area One, which leads to exponential increase in search space. Fig. 4.13 shows the results from RL, RG, GA and the greedy approach. In the tour case, RL outperforms the other algorithms in four out of the five budget settings. However, in the non-tour case in Fig.4.13 b, the greedy approach achieves better performance than other solutions when budget is less than 130. Meanwhile, the RG shows a limited performance under this scenario.

Overall, RL achieved the best performance in 15 cases out of 20 test cases. For path planning, besides the optimality, another important aspect is computation efficiency, which is demonstrated next.



(a) Area One (b) Area Two

Figure 4.14: Approximate run time of different algorithms for the graph of Area One and Two on iMac (4GHz, Intel Core i7).

Next we compare the computational efficiency of different path planning methods. RG suffers from a high computation complexity with $O((2nB)^I \cdot T_f)$ [15], where n is the number of vertices and T_f is the maximum time to evaluate the reward function on a given set of vertices, and I is the recursion depth. In our experiments, I is set to two in both cases. A larger recursion depth will increase the run time dramatically. The Greedy algorithm relies on the TSP solver to generate paths, and the complexity can be expressed as $O(Bn * t(n))$, where $t(n)$ is the complexity of the adopted TSP solver. GA is an evolutionary algorithm, and the complexity is dominated by the defined number of generations and population size.

For RL, we mainly focus on the inference time, since training can be done offline before path planning. The inference complexity using the snapshots of models is $O(T * |\mathcal{M}|)$, where $|\mathcal{M}|$ is the number of snapshots used and T is the number of steps in an episode.

The path planning time on an iMac desktop computer (4GHz Intel Core i7, 16 GB RAM, *without* GPU) is shown in Fig. 5.9. In both areas, RG takes much more time than other solutions. GA takes approximately 20 and 100 seconds in the two areas, respectively. Both the greedy algorithm and the RL solution are quite efficient (within seconds). In Area One, the greedy algorithm is faster, while in Area Two the run time is similar. This is because the greedy algorithm involves a TSP solver with a worst case exponential complexity while the complexity of RL is linear with respect to T or $|\mathcal{M}|$.

4.4 Discussion

We observe that in the smaller graph in Section 4.3, RL works well for both the tour and non-tour cases. While in the larger graph for the non-tour case, the performance degrades under some budget settings. In this section, we discuss three possible reasons when the RL based IPP fails to achieve the best performance.

The first reason is that the RNN is difficult to train with gradient descent when the sequence is longer, and may not converge to the global optimal policy. Thus, the path generated according to the policy network can be suboptimal.

Secondly, since the models are trained using a set of estimated budgets, the resulting model is expected to have different inference performance for a specific budget. It would be challenging to train a model that works best under all different conditions.

Thirdly, the underlying true path reward function is actually a discontinuous function [91] with respect to the budget. Depending on the graph and edge lengths, the budget needs to increase to a certain level so that the optimal path can change. Fig. 4.15 illustrates a simple example. An RNN can be used to represent continuous functions. When using them to approximate a step-wise function, the results may differ from the ground truth. This may lead to inaccurate Q-values.

4.5 Conclusion

In this chapter, we presented a RL framework for IPP. To address the unique challenges posed by path constraints, a novel action selection module is designed with the assistance of the shortest paths. Compared with the unconstrained action selection strategy, it has a better efficiency and optimality. Under the framework, we

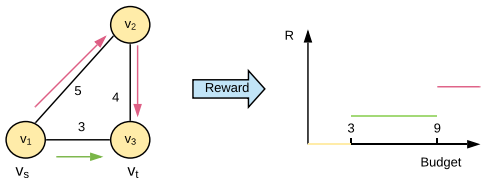


Figure 4.15: The actual Q-value is discontinuous with respect to budget. In this simple example, the graph makes a triangle. When $v_s = v_1$ and $v_t = v_3$, the minimal budget required from v_s to v_t is is 3. The best path ($v_1 \to v_3$) and corresponding reward will not change until budget increases to 9 ($v_1 \to v_2 \to v_3$).

implemented Q-learning, actor-critic and the reinforce algorithm, and compared with other IPP algorithms. The RL based solution achieves better path utility in most cases with less running time.

Chapter 5

Multi-robot Cooperative Path Planning

In the previous chapter, we presented a reinforcement learning framework that is able to learn the characteristics of informative paths. Given any budget in a pre-defined range, the RL model can generate paths efficiently. In recent years, with the technology advancement and availability of low-cost mobile robotic platforms like unmanned aerial vehicle, it is attractive to use multiple robots for data collection. In this chapter, we discuss solutions to extend the RL based path planning framework to generate informative paths for multiple cooperative robots.

5.1 Related Work

One important research topic in RL is multi-agent reinforcement learning (MARL). In MARL, the formulations differ for different task settings [11], e.g., whether agents can communicate with team members, whether agents can perceive the complete global

state, whether agents are decentralized, and whether the agents are cooperative or competitive. MARL problems are generally more challenging than single-agent RL and remain as an active research area in machine learning and robotics communities.

The most relevant MARL scenario to our work is the case of multi-agent fully cooperative learning, where all the agents perceive the same global state and determine their own actions. These actions form a joint-action, and after being executed, a team reward signal can be received by all agents. The task goal is to optimize the long-term team reward cooperatively. This type of multi-agent system (MAS) is formulated as a multi-agent Markov Decision Process (MMDP) [9]. In game theory, it is also called a fully cooperative stochastic game (or also called Markov game) [86].

One key challenge of MARL is that the size of joint action space increases exponentially with the number of agents. To reduce the search space and make learning and decision making scalable with respect to the number of agents, one solution is to use independent learners [107], where each agent learns based on individual reward signals.

With independent learners (or players), from a game-theoretic point of view, the concepts of equilibrium are utilized to define solutions [73]. The most commonly used solution concept is Nash equilibrium [43]. In a Nash equilibrium, each agent's policy is the best response to the other agents' policies. Thus, no agent can improve its gain by unilaterally deviating from its equilibrium policy. A markov game can have more than one Nash equilibrium. The ultimate goal of fully cooperative MARL is to search the optimal Nash equilibrium such that the corresponding team gain is maximized.

There are two categories of independent learning schemes in literature. The first category is to reshape the reward perceived by each agent, which is also known as the

multi-agent credit assignment [77]. In this case, each agent learns its own policy using the assigned reward. It is non-trivial to find an optimal credit assignment strategy. Intuitively, if the reward an agent receives is not commensurable to its efforts (e.g., costs), the agent may be discouraged from cooperation, or may become a freeloader. One well known credit assignment mechanism is difference-rewards, where each agent is rewarded for the marginal team utility after she joins the team. As such, each agent is awarded for her own contribution and the effect of other agents (i.e., noise) is removed [2]. Some works [77] have shown empirically that difference-rewards based credit assignment outperforms that based on equal reward split. One thing to notice about difference-rewards is that it changes the game from a fully-cooperative game to a general-sum game [44] since each agent has different reward assigned, and a potential risk is that it may not be able to find the optimal solution for the team.

The second category of approach is to let all the agents directly learn with the same team reward. There has been little theoretical analysis of the properties independent reinforcement learning in multi-agent settings. In [9], the authors conjecture that straightforward Q-learning will lead to Nash equilibrium strategies, although the equilibrium may not be optimal. However, the conjecture has not been formally proved so far.

In [60] the authors present a distributed Q-learning algorithm for deterministic MMDPs. They first propose to learn the optimistic Q-values, which is the maximal possible Q-value of an action when the other team players select the best response actions. However, even if each agent picks the best action with respect to the Q-values, the formed joint action may not be the optimal due to mis-coordination issue. To address this problem, they propose to memorize the best action with another table,

which can be seem as a coordination strategy based on social rules [9]. Although it was proved theoretically that the proposed method can find optimal policies in deterministic environments, the method can be only applied to tabular Q-learning since it needs to remember exact the best action seen so far given a specific state. For large state spaces where functional approximation (e.g., deep neural network) is required for state representation, it is non-trivial to extend the idea.

In [105], the authors propose Optimal Adaptive Learning, which was proved to converge to the optimal Nash equilibrium with probability one in any stochastic game. However, the computation complexity of the solution is too high since it needs to construct and solve a virtual game for each single state. Thus, it is not applicable in most real world stochastic games.

A few solutions to MARL consider heuristic strategies to encourage exploration. Specifically, Frequency Maximum Q-value (FMQ) is proposed in [49]. In FMQ, when an agent is exploring its action space, besides its predicted Q-values, it counts how frequently an action produces its maximum corresponding reward, which is similar to the optimistic projection in the distributed Q-learning in [60]. In [72], the authors proposed Hysteretic Q-learning, where the Q-functions are updated through two different learning rates. When the temporal difference is positive, a larger learning rate is adopted; a smaller learning rate is adopted when the temporal difference is negative. The distributed Q-learning in [60] can be seem as a special case of Hysteretic Q-learning when the smaller learning rate is 0. Another optimistic learning approach is the lenient learning introduced in [79]. The basic idea is to use a temperature to control leniency. Initially, the learners are lenient to forgive or ignore sub-optimal actions by teammates that lead to low rewards. Gradually, the degree of leniency is

decayed and agents transit from optimistic to average reward learners.

Table 5.1: Summary of MARL solutions for MMDP Tasks

Algorithm	Basic Idea	Theoretic Convergence
Independent Q-learning [19]	Directly apply conventional Q-learning.	Nash equilibrium (conjecture)
Distributed Q-Learning [60]	Learn optimistic projection, combined with rule based coordination.	Optimal policy
COIN [110]	Decompose the team reward for each independent learner.	No
OAL [105]	Create and solve virtual game for each step.	Optimal policy
FMQ heuristics [49]	Modify the exploration strategy.	No
Hysteretic Q-learning [72]	Use two different learning rates.	No
LDQN [79]	Use a temperature to control leniency.	No

Table. 5.1 summarizes representative solutions for MMDP tasks. Most of them do not have theoretic convergence guarantees. Although the approach in [105] and [60] are provably converge, their applications are limited due to unrealistic problem setup or simplified assumptions. In this thesis, to solve the MIPP problem, we present two solutions. One is based on the credit assignment approach. In the other solution, we leverage the unique spatial-temporal characteristics of MIPP and convert it to a single agent reinforcement learning.

5.2 Problem Formulation

Like the single-robot path planning problem, we formulate the MIPP problem with a five-tuple $\langle G, \mathcal{C}, \mathbf{v}_s, \mathbf{b}, f_{\mathcal{D}}(\mathcal{P})\rangle$. Specifically,

- $G = (\mathcal{V}, \mathcal{E})$ is a graph representing the layout and reachability of the target area.

- $\mathcal{C} \subseteq \mathcal{V}$ denotes the set of charging locations or depots. We assume that each charging location or depot has sufficient capacity to serve all robots in the system. Depots with limited capacity will be considered in our future work.

- $\mathbf{v}_s$ is a vector that denotes the start locations of the team of robots on G. The start location of robot i satisfies $v_s^i \in \mathcal{C}$, i.e, it is initially located at one of the depots. Let $N = |\mathbf{v}_s|$ be the number of robots.

- $\mathbf{b}$ is a vector of dimension N denoting the available travel budgets for the robots.

- $f_{\mathcal{D}}(\mathcal{P})$ is a function to evaluate the utility of a path set $\mathcal{P} = \cup_{i=1}^{n} \mathcal{P}_i$ traversed by the team of robots. Here, the subscript $\mathcal{D}$ represents historically collected sensing data, such as pilot data used to estimate the hyper-parameters of the GP. Like the single robot case, the utility of paths is defined as

$$f_{\mathcal{D}}(\mathcal{P}) = \mathrm{MI}(\mathbf{y}_{\mathcal{V}}; \mathbf{y}_{\mathcal{P}} \cup \mathbf{y}_{\mathcal{D}}) = H(\mathbf{y}_{\mathcal{V}}) - H(\mathbf{y}_{\mathcal{V}} \mid \mathbf{y}_{\mathcal{P}} \cup \mathbf{y}_{\mathcal{D}}) \tag{5.2.1}$$

The optimization goal of the MIPP is thus:

$$\begin{aligned} &\text{maximize } f_{\mathcal{D}}(\mathcal{P}), \\ &\text{s.t. } C(\mathcal{P}_i) \leq b_i \text{ and } \mathcal{P}_i[-1] \in \mathcal{C}, \\ &\forall i \in \{1, ..., N\}, \end{aligned} \tag{5.2.2}$$

where $C(\mathcal{P}_i)$ is the cost of path $\mathcal{P}_i$, and $\mathcal{P}_i[-1]$ is the last vertex of the path. For each

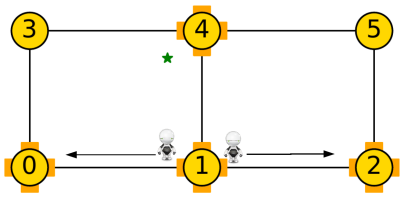


Figure 5.1: A simple 1-step MIPP example. The green '*' sign represents the location that has been previously sensed, the orange bold '+' shape denotes charging stations. Suppose each edge has unit length (1), if $\mathbf{v}_s = [1, 1]$ (vertex index) and $\mathbf{b} = [1, 1]$ (budgets), then the optimal paths for the two robots are $\mathcal{P}_1 = [1, 0]$ and $\mathcal{P}_2 = [1, 2]$, or $\mathcal{P}_1 = [1, 2]$ and $\mathcal{P}_2 = [1, 0]$ due to the existence of previously sensed locations. On the other hand, if $\mathbf{v}_s = [0, 2]$, the only solution is $\mathcal{P} = [[0, 1], [2, 1]]$ due to the budget and charging station constraints.

robot, it needs to return to one of the depots within its budget constraint. The initial locations and budgets of the robots may be the same or different. Clearly, MIPP is NP-hard since it is equivalent to IPP with identical start and terminal locations when $N = 1$ and $|\mathcal{C}| = 1$.

Fig. 5.1 shows a toy example of MIPP with 2 robots. In this example, each robot has a budget of one unit. The costs of all edges are one. In this case, MIPP can be viewed as a 2-agent matrix game. By varying the start locations or the locations of the pilot data (indicated by the green star), the game may have one or multiple equilibria, which can be found manually. Even for such a simple scenario, designing an MARL strategy that converges to the optimal equilibrium is non-trivial for a repeated version of the game. As the number of the robots and budgets increase, for larger graphs, the search space becomes prohibitively vast for any manual solution to be feasible.

5.3 RL Strategies to MIPP

Due to the NP-hardness of MIPP, one needs to devise heuristic solutions that can be computed in polynomial time. Conventional heuristic approaches such as GA require re-execution of their procedures whenever any parameter in $\langle G, \mathcal{C}, \mathbf{v}_s, \mathbf{b}, f_{\mathcal{D}}(\mathcal{P})\rangle$ changes since the corresponding problem instance is different. The advantage of an RL-based solution, on the other hand, lies in its ability to generalize to problem instances with different parameter settings with a single round of training.

Next, we first formalize MIPP from the prospective of Multi-agent Markov Decision Processes (MMDPs), and then discuss the proposed state encoding and reward function. Lastly, we present three different RL policies.

5.3.1 MMDP for MIPP

A multi-agent Markov Decision Process [9] is defined using a five-tuple $\langle S, \alpha, \{\mathcal{A}_i\}_{i\in\alpha}, Pr, R\rangle$, where

- S and α are finite sets of states and agents, and the states can be perceived by all agents,
- $\mathcal{A}_i$ is the set of actions available to agent i,
- $Pr : S \times \mathcal{A}_1 \times \cdots \mathcal{A}_N \to [0, 1]$ is the state transition probability,
- $R : S \times \mathcal{A}_1 \times \cdots \mathcal{A}_N \to \mathbb{R}$ is a reward function.

At any stage, each agent selects an action and executes it. The actions from all agents form a joint action. Accordingly, the environment makes a state transition and provides a reward signal. An MMDP is also known as a stochastic game with

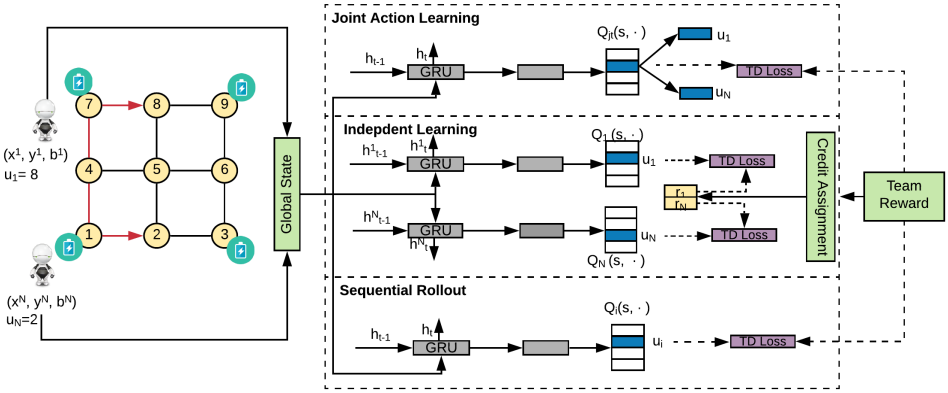


Figure 5.2: State encoding and three learning schemes

identical interests (or a fully cooperative game) in game theory [86], where each player tries to maximize the same total pay-off.

In MIPP, each agent represents a data collection robot. Initially, agent i is at its start location v_s^i with an initial budget b_i. The available actions are the neighboring vertices that agents can move to given remaining budgets. Execution of the joint action results in a single team reward. However, unlike the definition in [9], in MIPP, the state transitions are deterministic, i.e, given the current state and a joint action, the next state is unique. The goal of the agents is to plan paths that maximizes cumulative discounted team rewards. All agents can observe the same global state. The state transition and the reward function are not known *a priori*, and need to be learned through RL.

5.3.2 States and Action Selection

We denote the state transition tuple at step t as $\langle s_t, \mathbf{u}_t, R_t, s_{t+1} \rangle$, where $\mathbf{u}_t = [u_t^1, ..., u_t^N]$ is the joint action, and R_t is the team reward. The state of robot i is encoded as

(x_t^i, y_t^i, b_t^i), which consists of its coordinates in the target area and its remaining budget. Thus, the state of all agents at time t is given by $s_t = \{(x_t^i, y_t^i, b_t^i)_{i=1}^N\}$. The remaining budget b_t^i is updated as

$$b_t^i = b_{t-1}^i - c(v_{t-1}^i, v_t^i), \tag{5.3.1}$$

where $c(v_{t-1}^i, v_t^i)$ is the cost incurred by traversing the previous edge (v_{t-1}^i, v_t^i), and $b_0^i = b_i$ is the initial budget. Here, $v_t^i = (x_t^i, y_t^i)$ is the nodal position of agent i as the result of action u_t^i. Since historical actions (past paths) affect the current reward, we adopt a Recurrent Neural Network based on GRU [18] to encode the historical information as hidden states.

The available actions at t to agent i are the neighboring vertices of its current location. Since agents have different budgets, some may terminate earlier than others. To handle this situation, we include a dummy action to the action space, which is available when an agent arrives at a charging station and has insufficient budget to visit other locations. When all the agents arrive at charging stations, one training episode terminates. Furthermore, since an agent must reach one of the charging stations, we further prune the available actions of an agent as follows. Let the current position of agent i be v_t^i. With respect charging station $c_j \in \mathcal{C}$, its valid actions are given by

$$\mathcal{A}_t^i(v_t^i, c_j) = \{v \in \mathrm{Nbr}(v_t^i) : c(v_t^i, v) + \mathrm{LCP}(v, c_j) \leq b_t^i\}, \tag{5.3.2}$$

where Nbr represents adjacent vertices, and LCP denotes the cost of the Least Cost Path from v to v_t that can be computed using the Dijkstra algorithm. Thus, the valid

action set for agent i at v_t^i is,

$$\mathcal{A}_t^i(v_t^i) = \bigcup_{j=1}^{|\mathcal{C}|} \mathcal{A}_t^i(v_t^i, c_j). \tag{5.3.3}$$

In the implementation, we define a mask vector based on (5.3.3) and add it to the Q-values to remove invalid actions, so that agents are guaranteed to reach some charging stations within budget constraints.

5.3.3 Team Reward

The team reward R_t is calculated as the reduction in the uncertainty of $\mathbf{y}_\mathcal{V}$ due to the incremental data collected by all robots at step t. Recall that uncertainty only depends on sensing locations under the GP assumption, which can be determined from the sampling frequency and the moving speed of robots. Formally,

$$R_t = H(\mathbf{y}_\mathcal{V} \mid \mathbf{y}_{\mathcal{P}_{t-1}} \cup \mathbf{y}_\mathcal{D}) - H(\mathbf{y}_\mathcal{V} \mid \mathbf{y}_{\mathcal{P}_t} \cup \mathbf{y}_\mathcal{D}), \tag{5.3.4}$$

where $\mathbf{y}_{\mathcal{P}_{t-1}}$ and $\mathbf{y}_{\mathcal{P}_t}$ are the sensor data collected by the robot team through the previous and current steps, respectively. Clearly, $\sum_t R_t = f_\mathcal{D}(\mathcal{P})$ is exactly the optimization goal. As a result, the team of robots are incentivized to explore informative paths through the reward signal.

5.3.4 Learning Schemes

To make the learned policy generalizable to different start locations and initial budgets for all agents, during the training stage, we treat $\mathbf{v}_s$ and $\mathbf{b}$ as variables. For each

training episode, we randomly sample the start location of each robot from the set $\mathcal{C}$. Similarly, the budget of each robot is sampled uniformly from $[B_{\min}, B_{\max}]$. Next, we present three types of training schemes based on Q-learning.

5.3.4.1 Joint Action Learning

Joint Action Learning (JAL) treats the MMDP as a single agent MDP, and encodes a joint action as a single action. For instance, in a two-agent system, a pair of action (u_1, u_2) can be encoded as a joint action index $u_1 \cdot (1 + |\mathcal{V}|) + u_2$ (note the existence of the dummy action). Conversely, any joint action index can be mapped to a pair of actions. The reward signal can be thus seen as the reward to the encoded single action. By doing so, existing single-agent Q-learning algorithm like DQN [75] can be applied. We call this approach DQN-JAL. The main disadvantage of JAL is that it can only handle small problem sizes with limited number of agents and a small action set.

5.3.4.2 Independent Q-learning

One way to reduce the action space of the MMDP is by assigning credits to agents based on their respective actions and the team reward in each step, and training the agents separately. Formally, a credit assignment strategy is defined as a function $\eta : S \times A \times \mathbb{R} \to \mathbb{R}^N$ that maps $\{s_t, \mathbf{u}_t, R_t\}$ to $[r_t^1, ..., r_t^N]$. We consider two different forms of $\eta(s_t, \mathbf{u}_t, R_t)$ for MIPP. The first approach is to split reward R_t equally among robots regardless of their actions with the exception of robots who have terminated and will receive zero awards. The second approach is based on difference rewards [77, 110], which consider the marginal contribution of each agent. In particular, r_t^i can be

calculated as

$$r_t^i = H(\mathbf{y}_{\mathcal{V}} \mid \mathbf{y}_{\mathcal{P}_t^{-i} \cup \mathcal{D}}) - H(\mathbf{y}_{\mathcal{V}} \mid \mathbf{y}_{\cup_{j=1}^{N} \mathcal{P}_t^j \cup \mathcal{D}}), \tag{5.3.5}$$

where $\mathcal{P}_t^{-i}$ denotes the set of paths from teammates, excluding the path from agent i. Since $\sum_{i=1}^{N} r_t^i \neq R_t$, we further normalize r_t^i with $\frac{R_t}{\sum_{i=1}^{N} r_t^i}$ to obtain the final reward.

Based on the outcome of credit assignment, the training procedure of independent learning is outlined by Algorithm 5. We adopt Q-learning as the RL algorithm for good sample efficiency. The optimization goal is to minimize the temporal-difference (TD) error of the Q-function network,

$$\mathcal{L}_{\text{TD}}(s_t, u_t^i, r_t^i, s_{t+1}; \boldsymbol{\theta}_i) = (Q_i(s_t, u_t^i; \boldsymbol{\theta}_i) - y_{\text{target}}^i)^2, \tag{5.3.6}$$

where $y_{\text{target}}^i = r_t^i + \gamma \max_{u_{t+1}^i} Q_i(s_{t+1}, u_{t+1}^i; \boldsymbol{\theta}_{\boldsymbol{i}}^-)$. For every C episodes, we save the model parameters $\boldsymbol{\theta}$ (the set of parameters for all agents). We call this approach IQL-EqualSplit or IQL-DiffRewards depending on the credit assignment approach used.

5.3.4.3 Sequential Rollout

In MIPP, intuitively, agents can maximize team rewards by spatially spreading out in the target area. Their actions can be performed in an asynchronous manner. For example, given two paths, collecting data using two robots concurrently achieves the same utility as running one robot after the other sequentially along the paths since utility only depends on the locations of data collection (though the completion time is different). However, unlike sequential allocation [93] that treats robots independently, when planning the path of the first robot, one should be cognizant of the initial

Algorithm 5: IQL based on Credit Assignment

Input : $\langle G, \mathcal{C}, f_{\mathcal{D}}(\mathcal{P}), B_{\min}, B_{\max} \rangle$, model save cycle C
Output: model parameter set Θ

initialize replay memory $\mathcal{M}$
initialize $\boldsymbol{\theta}$ randomly and set $\boldsymbol{\theta}^{-} = \boldsymbol{\theta}$
for episode $e = 1, 2, ...$ **do**
  set $\mathbf{v}_s$ and $\mathbf{b}$ by sampling from $\mathcal{C}$ and $[B_{\min}, B_{\max}]$
  get initial global state s_0
  for step $t = 1, 2, ..., T$ **do**
    for agent $i = 1, 2, ..., N$ **do**
      with probability ϵ select a random action u_t^i from $\mathcal{A}_t^i(v_t^i)$
      otherwise select $u_t^i = \arg\max_{u_t^i} Q_i(s, u_t^i)$
    end
    take action $\mathbf{u}_t$ and calculate R_t
    store transition $(s_t, \mathbf{u}_t, R_t, s_{t+1})$ to $\mathcal{M}$
    sample a batch of $(s_j, \mathbf{u}_j, R_j, s_{j+1})$ from $\mathcal{M}$
    calculate the credit assignment $[r_j^1, ..., r_j^N]$
    update $\boldsymbol{\theta}_i$ by minimizing the loss $\mathcal{L}_{\text{TD}}(s_j, u_j^i, r_j^i, s_{j+1}; \boldsymbol{\theta}_i)$ for each agent
  end
  set $\boldsymbol{\theta}^{-} = \boldsymbol{\theta}$ with some period K
  if $e \bmod C = 0$ **then**
    save parameter $\boldsymbol{\theta}$ to Θ
end
return Θ

locations and budgets of the remaining robots. Motivated by this key insight, we propose a new cooperative mechanism called *sequential rollout* by converting the MMDP to a single-agent decision process. We fix the order of the robots. At each time step t, the agent makes decision for one robot, and all the other robots take the dummy action, i.e, they stay at the same location. When the current robot arrives at a charging station and is out of budget, the agent switches to the next robot on the list.

In sequential rollout, the full state includes the status of all robots to predict the total discounted future reward. However, the one-step reward only comes from the active robot and its action. The main training procedure is similar to Alg. 5 except that there is only a single agent, and no credit assignment is needed. This approach does not suffer from the problem of exponential growth of the joint action space with respect to the number of robots. One potential disadvantage is that the time step horizon is increased from T steps to NT steps. In other words, it trades spatial with time. This may make the prediction of the total future reward less accurate compared with a shorter horizon. We call this approach SEQ-ROLLOUT-DQN with DQN being the learning algorithm.

5.3.5 Path Planning

Once the model has been trained, it can be utilized for path planning. Given any $\mathbf{v}_s$ and $\mathbf{b}$, a greedy policy with respect the Q-values can be used to select actions. Note that except for JAL-DQN, the next action of a robot only takes $O(|\mathcal{V}|)$ time.

To mitigate local optimality, we execute the path planning algorithm multiple times with different model parameters output by Algorithm 6, and select the path set

with the maximum reward. The path planning procedure is outlined in Algorithm 6 for independent learning. For sequential rollout, paths are generated one by one sequentially.

Algorithm 6: Path Planning Procedure (with IQL)

Input : $\Theta, \mathbf{v}_s, \mathbf{b}$
Output: best path $\mathcal{P}^*$
initialize a candidate path set $\mathcal{P}_\Theta = \{\}$
for $\boldsymbol{\theta} \in \Theta$ **do**
  get initial global state s_0
  for step $t = 1, 2, ..., T$ **do**
    for agent $i = 1, 2, ..., N$ **do**
      select $u_t^i = \arg\max_{u_t^i} Q_i(s, u_t^i)$ based on $\boldsymbol{\theta}$
    end
    take action $\mathbf{u}_t$
  end
  add the generated path to $\mathcal{P}$
end
return $\mathcal{P}^* = \arg\max_{\mathcal{P} \in \mathcal{P}_\Theta} f_{\mathcal{D}}(\mathcal{P})$

5.4 Performance Evaluation

In this section, we compare the performance of different strategies for MIPP. For evaluation, we consider the task of path planning as introduced in Chapter 4, but with multiple robots. The same two areas are evaluated.

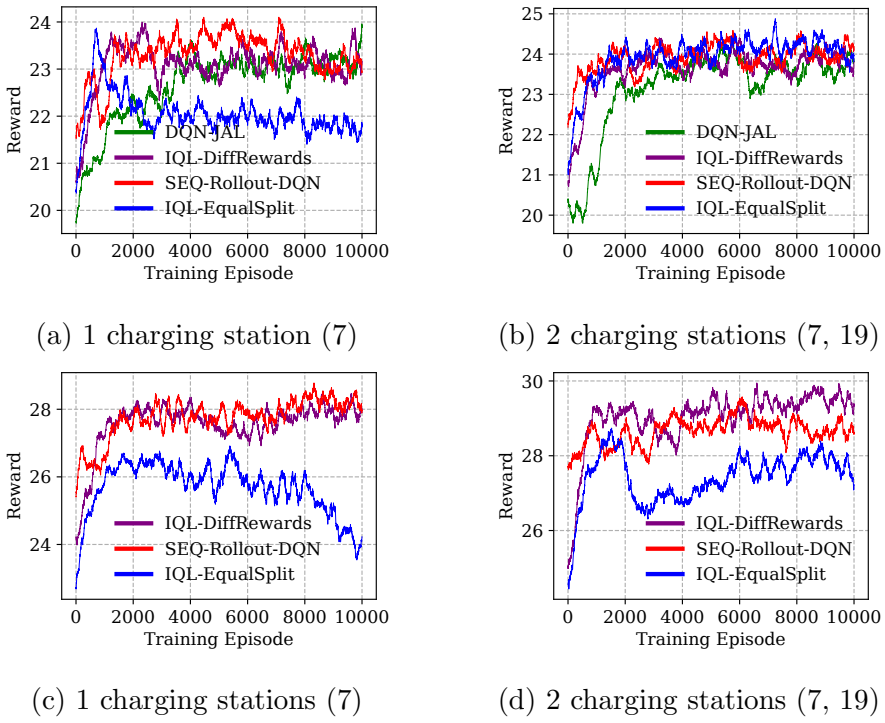
5.4.1 Implementation and Environment Setup

During training, the neural networks in DQN are implemented using PyTorch [1]. RMSProp [7] is utilized as the optimizer for parameter estimation with a learning

[1] https://pytorch.org

rate of 0.0001 for all the learning schemes. The minibatch size is set to 32 for each training iteration. The size of experience buffer is 5000. The discounter factor γ for long-term returns is set to 0.99. Different settings are considered, including different numbers of robots, charging stations and budgets. In each setting, we train 10000 episodes on the graph of Area One, and 20000 episodes on the graph of Area Two.

5.4.2 Training and Convergence



(a) 1 charging station (7)

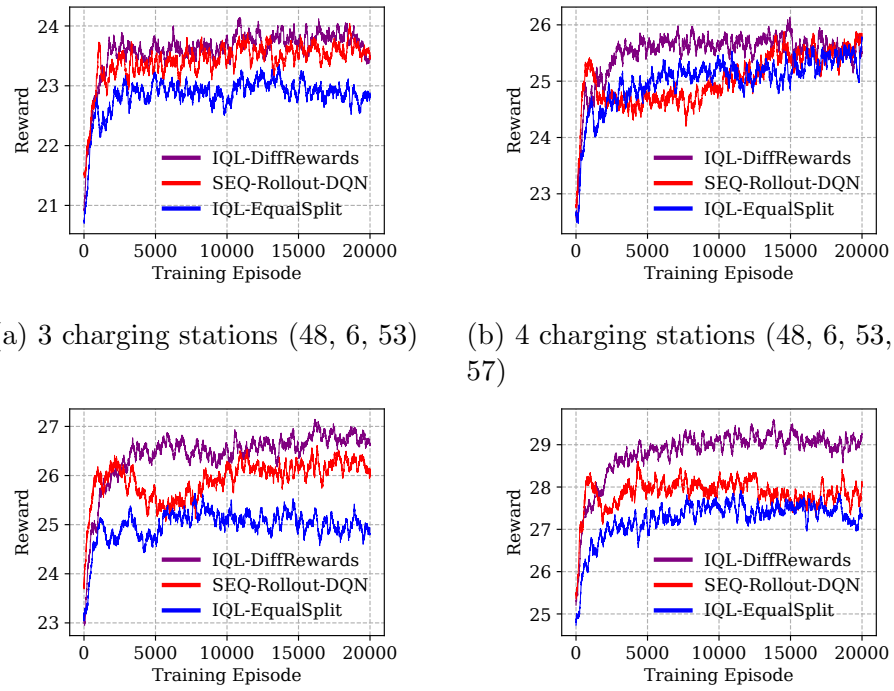
(b) 2 charging stations (7, 19)

(c) 1 charging stations (7)

(d) 2 charging stations (7, 19)

Figure 5.3: Average reward per episode during training for Area One, with different setting of charging stations and number of robots. (a) and (b): **2** robots, (c) and (d): **3** robots.

In the first area, policies are trained for the cases of 2 or 3 robots. In each case, 1 or 2 charging stations are included. In the second area, due its elongated shape, more robots (3 – 4), and charging stations (3 – 4) are deployed in the field. Robots



(a) 3 charging stations (48, 6, 53)

(b) 4 charging stations (48, 6, 53, 57)

(c) 3 charging stations (48, 6, 53)

(d) 4 charging stations (48, 6, 53, 57)

Figure 5.4: Average reward per episode during training for Area Two. (a) and (b): **3** robots, (c) and (d): **4** robots.

in the first and the second areas have budget ranging from 15 to 30 units and 30 to 50 units, respectively. The cost of a single edge in the first and the second areas are 2.5 units and 2 units, respectively. During training, budgets are sampled from these ranges for all robots as shown in Alg. 5.

Fig. 5.3 shows the average reward during training per episode for robots in Area One. When there are only 2 robots, we also evaluated DQN-JAL, which has a joint action space of size $28 \times 28 = 784$. Due to its poor scalability, as the graph size grows (in Area Two) or the number of robots increases, it is no longer computationaly feasible to run DQN-JAL. From the figure, we see that in the first area,

IQL-DiffRewards and SEQ-Rollout-DQN have a comparable performance, and are both better than IQL-EqualSplit. In addition, DQN-JAL converges slowly and reaches similar rewards as IQL-DiffRewards and SEQ-Rollout-DQN after 10,000 episodes of training.

Fig. 5.4 shows the rewards during the training process with 3 and 4 robots in Area Two. Similar to the first Area, it can be observed that IQL-EqualSplit has inferior performance overall, and IQL-DiffRewards achieves the best result among all three schemes. Moreover, the performance of SEQ-Rollout-DQN degrades as more robots are available. As discussion previously in Section 5.3, sequential rollout has a longer time horizon since the total number of steps increases linearly with the total budget of the robots. In contrast, with independent learning, all robots are trained concurrently.

Finally, the search space grows when the number of robots or the number of charging stations increases. For instance, when there is only one charging station, the paths of all the robots form a tour. Adding an extra charging station means that robots have more available paths to explore since they do not have to return to where they started. In both two areas, rewards at convergence time increase with more robots or charging stations as expected.

5.4.3 Path Planning Performance

To evaluate the informativeness of the predicted paths of the proposed schemes, we also implement a non-learning sequential allocation method inspired by [94], which plans paths one after another independently using a single-robot IPP algorithm. We

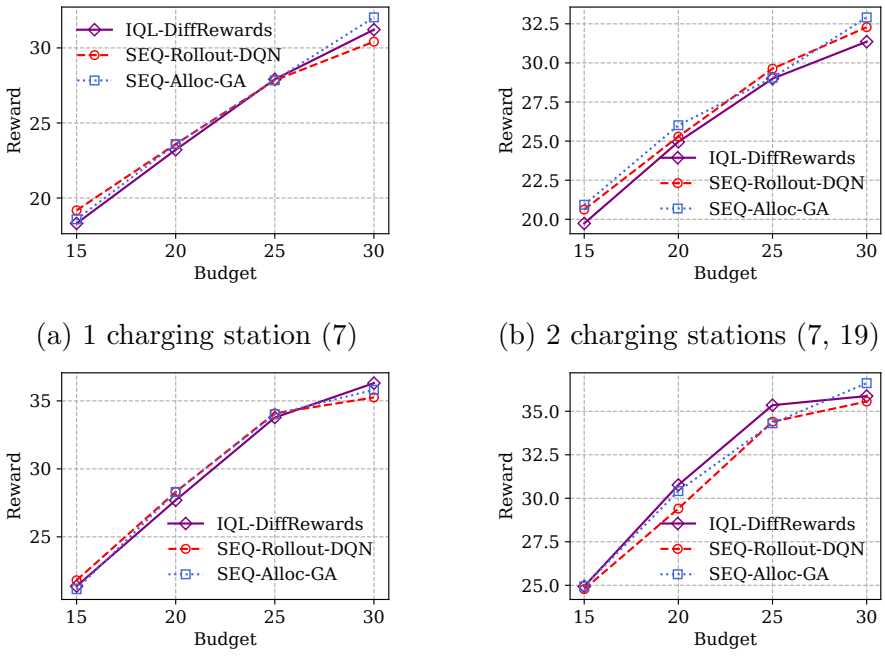
adopt GA as the single-robot IPP algorithm. In GA, paths are encoded as chromosomes, cross over and mutation operators are designed to increase the diversity of the population. The best path among the final evolution generation is chosen as the output. Due to the existence of multiple charging stations, the GA needs to be adapted to handle situations that robots can terminate at *any* of the charging stations. We refer to this approach as SEQ-Alloc-GA in subsequent comparisons. In the experiments, the population size is set to 100 and 50 generations are evolved. Note that SEQ-Alloc-GA needs to be re-executed each time the budgets of robots or their initial positions change.

For RL based solutions, we select policies trained by IQL-DiffRewards and SEQ-Rollout-DQN in path planning, since they have fast and better convergence as seen in the previous section. Next, we present the results of two scenarios, namely, homogeneous budgets and heterogeneous budgets among the robots. In the first scenario, all robots have the same budget, while in the second case, at least one robot has a different budget from the rest.

5.4.3.1 Homogeneous Budgets

Fig. 5.5 shows the rewards of different path planning schemes in Area One. In the experiments, we vary the budgets, the number of robots and charging stations. It can be seen that the RL-based path planning methods achieve similar rewards as SEQ-Alloc-GA. As the budgets increase or the number of robots increases, high reward can be achieved by all schemes. Introduction of more charging stations also improve the rewards as more feasible paths are available to all robots.

Similar observations can be made for Area Two as shown in Fig. 5.6 for 3 and 4



(a) 1 charging station (7)

(b) 2 charging stations (7, 19)

(c) 1 charging station (7)

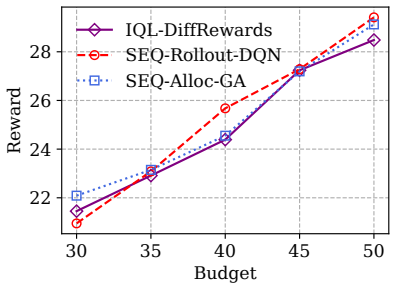
(d) 2 charging stations (7, 19)

Figure 5.5: Path planning performance in Area One for homogeneous robots. The X-axis shows the budget for each robot for a single experiment. (a) and (b): **2** robot team, (c) and (d): **3** robot team. In (b), the initial locations $\mathbf{v}_s = [7, 19]$, in (d), $\mathbf{v}_s = [7, 7, 19]$.

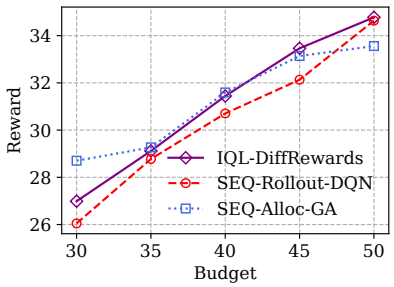
robots. All methods have competitive performance.

5.4.3.2 Heterogeneous Budgets

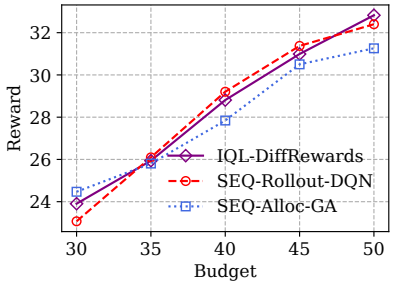
Fig. 5.7 and Fig. 5.8 show the results of planning robots with heterogeneous budgets in Area One and Area Two, respectively. In this scenario, SEQ-Alloc-GA has inferior performance than the RL-based solutions in most cases. This is because SEQ-Alloc-GA is a greedy solution. It can only make use of the current robot's budget information, and plan the robots independently. When the robots have different budgets, the order which robot is planned first matters. To understand this, consider



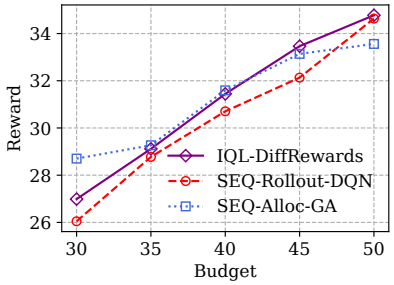
(a) 3 charging stations (48, 6, 53)



(b) 4 charging stations (48, 6, 53, 57)



(c) 3 charging stations (48, 6, 53)

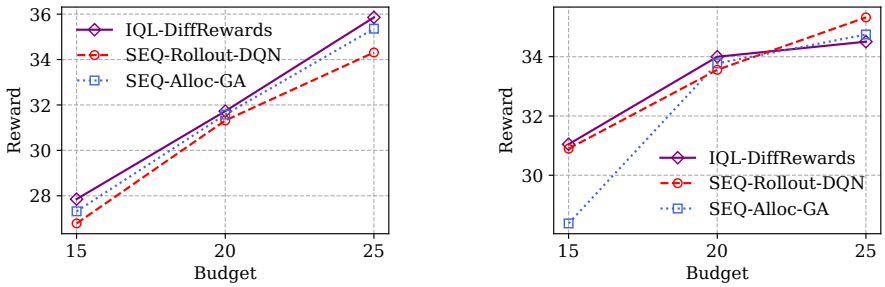


(d) 4 charging stations (48, 6, 53, 57)

Figure 5.6: Path planning performance in Area Two for homogeneous robots. (a) and (b): **3** robot team, (c) and (d): **4** robot team. In both (a) and (b), the initial locations $\mathbf{v}_s = [48, 6, 53]$, in (c), $\mathbf{v}_s = [48, 6, 53, 48]$, in (d), $\mathbf{v}_s = [48, 6, 53, 57]$.

the case of two robots, one with a large budget and one with a small budget. To maximize the total reward, it is desirable to plan the robot with the smaller budget first as the other robot has more flexibility to maneuver to regions with higher rewards given the decision of the first robot. The RL-based schemes, by design, take into account of the (future) decisions of all robots in updating Q-functions and thus are agnostic to the orders.

Lastly, it can be observed from Fig. 5.7 and Fig. 5.8 that with the exception of SEQ-Alloc-GA, all schemes achieve higher rewards with more average budgets and more charging stations in both areas. IQL-DiffRewards has the best performance



(a) 1 charging station (7) nodes (b) 2 charging stations (7, 19)

Figure 5.7: Path planning performance in Area One for **3** heterogeneous robots. One robot has a fixed budget $b = 30$, and X-axis shows budgets for the other two robots. In (b), the initial locations $\mathbf{v}_s = [7, 19, 7]$.

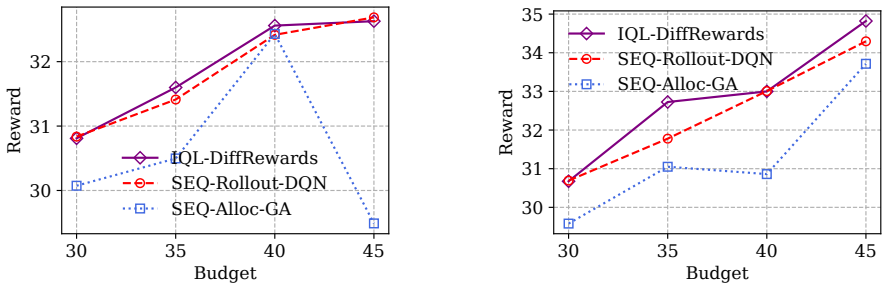
in most cases.

5.4.4 Computation Efficiency

In addition to the total rewards, an important evaluation metrics is the computation time. Without runtime limitation, one can even adopt brute force search to find paths with the highest rewards. For GA, one can infinitely increase the population size or the number of evolution generations to approach global optimal.

For RL-based MIPP schemes, we focus on the computation time of path planning (or inference), since the policies are trained beforehand without the knowledge of specific budgets or initial locations of robots. The computation complexity of IQL-DiffRewards and SEQ-Rollout-DQN is both $O(N|\mathcal{V}|B_{\max}|\Theta|)$, where $B_{\max}$ is the maximum budget of the robots and $|\Theta|$ is the size of the model parameter sets. The complexity of JAL-DQN is $O(|\mathcal{V}|^N B_{\max}|\Theta|)$.

Fig. 5.9 shows the path planning run time of different approaches on a desktop PC with Intel Core i7 and 16GB memory for the two target areas. It can be seen that both



(a) 3 charging stations (48, 6, 53)

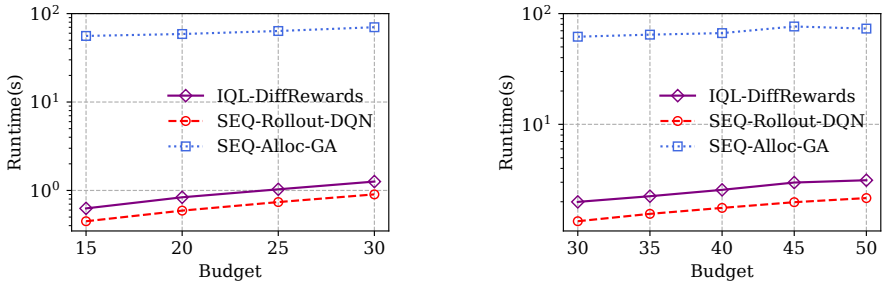
(b) 4 charging stations (48, 6, 53, 57)

Figure 5.8: Path planning performance in Area Two for **4** heterogeneous robots. Two robots have a fixed budget of 50, and X-axis shows the budgets of the remaining two robots. In (a), the initial locations $\mathbf{v}_s = [48, 6, 53, 48]$ and (b), the initial locations $\mathbf{v}_s = [48, 6, 53, 57]$.

RL-based path planning solutions are efficient, and can finish within seconds even in the worst case. Since in each time step, IQL-DiffRewards needs to compute the action of each agent, it is slightly slower than SEQ-Rollout-DQN. We error on the optimistic side to run GA for only 50 generations though it is far from reaching the optimal solution. Even so, SEQ-Alloc-GA takes much longer time than the RL-based solutions.

5.5 Conclusion

In this chapter, we formulated cooperative spatial sensing among multiple robots as a MMDP problem and developed efficient RL-based solutions. We devised two solutions, namely, individual learning based on credit assignment and sequential rollout based reinforcement learning. With both schemes, the learned policy is utilized to plan paths and achieve higher or similar rewards compared with baseline solutions in most cases, but with less run time.



(a) Area One, 2 robots (b) Area Two, 3 robots

Figure 5.9: Approximate run time of different solutions for the two areas on a desktop (16GB memory, Intel Core i7).

Chapter 6

Data Collection through Mobile Crowdsourcing

In the previous chapters, we investigated the path planning problems for mobile robotic sensing so as to maximize data utility given limited budgets. Besides robotic sensing, another popular data acquisition approach emerged in recent years is mobile crowdsourcing (MCS). Extensive research has been done for MCS-based data collection [13]. However, it remains challenging to deploy such approaches in practice.

As an example, MCS-based Wi-Fi fingerprint site survey was first proposed more than a decade ago [80], but until now no successful deployment is known due to a number of challenges. First, an MCS campaign requires contributions from participants, but individuals may be reluctant to participate and contribute data for various reasons (e.g., privacy concerns , extra efforts required, additional costs or energy expenditure). Consequently, a campaign organizer needs to motivate the crowd to participate and gather data. The second challenge is how to impart the necessary knowledge to the crowd so they can conduct site surveys successfully. Although site

survey may seem easy and trivial to experts, laymen may find it complicated to figure out where and how to collect data without prior knowledge. Lastly, it is difficult to guarantee the quality of crowdsourced data, which may be inaccurate due to inexperienced users or low-quality sensors. To an organizer, low-quality data leads to erroneous data analysis. To participants, sensing and delivering inaccurate data are wasteful in terms of battery and time.

Due to these challenges, conducting effective MCS-based site surveys for location fingerprints remains an open problem. Existing works only address a subset of the challenges. For instance, [80] investigated when to prompt users to provide inputs and how to detect erroneous crowdsourced data. [62] developed mechanisms to bind fingerprints to locations when no maps are available. [111] focuses on how to use crowdsourced data, and only involves a few volunteers to simulate an MCS scenario and get cowdsourced data in its evaluation. None of these works considered practical issues such as participant recruitment and incentive mechanisms. We believe these challenges are primarily social-psychological, since it involves human nature and behavior characteristics. Thus, the main motivation of this study is to investigate human responses to MCS-based site survey through a real-world campaign.

In this chapter, we present a user behavior study through a real world MCS campaign on Wi-Fi fingerprints. We tackle all aforementioned challenges, from platform design to participant recruitment and analysis of localization performance. Our main contribution can be summarized as follows.

- *An indoor fingerprint MCS platform.* We design and implement an Indoor Fingerprint Crowdsourcing System (IFCS), which harnesses the crowd for fingerprint site survey.

- *A novel metric to assess participant contributions.* To promote better coverage of target areas, we propose a metric based on the informativeness of participants' collected data.

- *A large-scale real-world data collection campaign* [1]. From September 2019 to January 2020, a data collection campaign had been conducted in the McMaster University campus for five months. In total, 97 participants signed up for the campaign and 66 of them uploaded fingerprint data. Over 1400 fingerprints had been collected from 24 buildings and 58 floors at the cost of $200.

- *Participant recruitment and behavior analysis.* To the best of our knowledge, this work is the first to investigate participant recruitment and behavior analysis for an MCS-based site survey.

- *Data quality control and inspection method.* Data collected by the crowd may suffer from poor quality such as inaccurate labels due to intentional or unintentional mistakes. We present a few methods to validate crowdsourced data before they are used for localization model training. Assessed using test data collected from three different devices, the resulting localization model achieves an average localization error of 9.6 meters, which is comparable to the result in [67].

Although the goal of the campaign is to collect Wi-Fi fingerprint data, the designed platform and proposed solutions are applicable to other similar types of spatial data.

[1] The data collection campaign and the follow-up survey have been approved by the research ethics board of McMaster University.

6.1 System and Campaign Design

In this section, we present the system and campaign design. First, we discuss the main design considerations. Then, we present a system overview followed by details of main components.

6.1.1 Design Considerations

Although several studies have proposed to use MCS for fingerprint site survey, it is challenging to put the idea into practice. To successfully operate an MCS campaign for site survey in the real-world, we need to carefully design campaign strategies along several aspects:

- *user recruitment*: MCS relies on participants and they may be reluctant to carry out a data collection task. As a result, it can be challenging to find a sufficient number of participants.
- *teaching phase*: How to impart necessary knowledge to participants is another non-trivial concern, since they may not know how to make contributions but a long-learning session is likely to discourage people from participating.
- *participant contribution*: A fair contribution assessment method is needed. It is desirable that the measure is independent of a participant's device type, and should encourage him to explore more un-surveyed areas.

6.1.2 System Overview

The implementation of IFCS adopts a client-server architecture. Fig. 6.1 illustrates the main modules on the client and server sides. The client side is an Android App

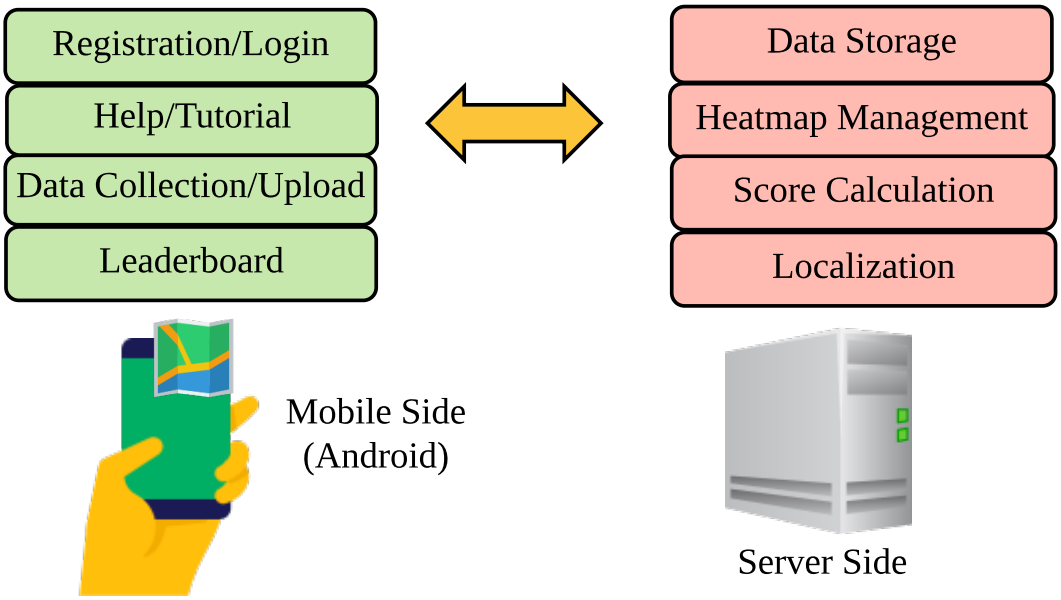


Figure 6.1: Main modules in the crowdsourcing platform

(SDK version 26) that we published in Google Play Store[2]. The main component of the *user interface* (UI) is a map, which is customized with the Mapbox Studio[3]. Specifically, location features such as rooms and stairs are extracted from vectorized indoor floor plans and then stacked on a base map layer of building contours and pathways from OpenStreetMap[4].

On the client side, detailed help information and tutorials are provided inside the App. Specifically, two options for data collection are provided to the user. The first option is *point-based collection*, in which a participant travels to selected locations and collects respective fingerprints at these locations. In the second option, called *path-based collection*, a participant specifies a path by marking the start and terminal

[2]https://play.google.com
[3]https://www.mapbox.com/
[4]https://www.openstreetmap.org/

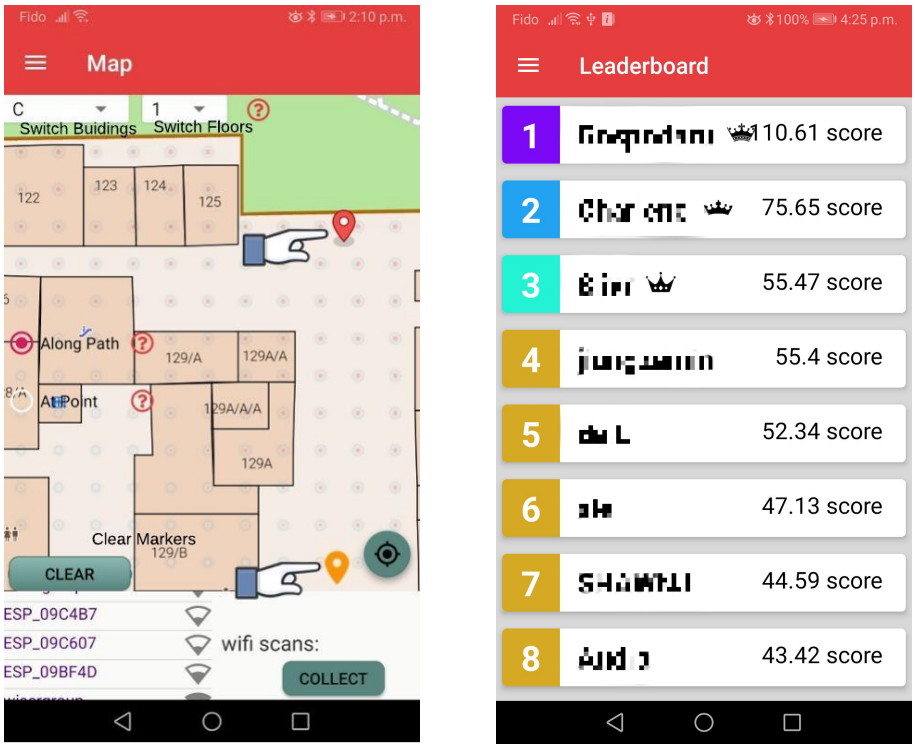
positions on the map inside the App[5], and then walks along the path. During the walk, Wi-Fi scanning is performed in background. Location labels of Wi-Fi scans are inferred during the post-processing stage by leveraging the path information [64].

When a participant indicates the completion of the data collection process (e.g., finishing scanning at one location in the point-based method or arrival at a terminal location in the path-based approach), all the data (e.g., Wi-Fi signals, IMU sensors, location labels, device information, and timestamps) are packed into a single package using Protocol Buffers[6] and uploaded to the server. Once data are delivered, the participant receives an immediate feedback containing the *scores* she obtained for this submission. There may be cases when data fail to be uploaded due to Wi-Fi disconnection. To reduce user intervention, these data will be uploaded automatically in the background when connection is established again. Participants can also check their total scores and their rankings in the "Leaderboard" page as shown in Fig.6.2 b. As previously explained, the leaderboard provides a gamified incentive for contributing data, as the students would be in a game competing with one another.

The server side was developed with Django[7]. Received data are organized by buildings and floors. For each Building/Floor (referred as B/F in later descriptions), the server tracks the locations that have been surveyed. A heatmap is generated to indicate which areas have not been surveyed and thus promise more scores. The score module evaluates users' contribution, which can be used to redeem monetary compensation.

[5]The location labels are recorded in the GPS format.
[6]https://developers.google.com/protocol-buffers
[7]https://www.djangoproject.com

6.1.3 User Interface



(a) MAP (b) Leaderboard

Figure 6.2: Main user interfaces of IFCS.

After registration, when a participant logs in for the first time, the user interface shows a few tutorial pages. As soon as participants get familiar with the procedure, they can start collecting data. To make the user interface easy to understand, annotation is also provided for each UI element with a question mark. Fig.6.2 a shows the main user interface for data contribution. Cellular network or GPS is used as a coarse-grained localization method to determine the participant's current building automatically. Manual building selection can be used when the automatic selection fails. Additionally, participants can choose the floor levels they are on. Indoor floor

detection itself is a challenging and non-trivial task [25, 38, 45]. Thus, we let participants decide the floor levels. At the final data submission stage, a dialog will appear to confirm the floor selection (Fig. 6.3 b).

With the above design, we expect that the learning curve is shortened so that participants (especially those without any prior knowledge of site survey) can figure out the data collection procedure by themselves. Unlike many MCS systems proposed in literature, IFCS does not perform task assignment [31]. Participants do not have to accept pre-defined tasks and finish them. Instead, they are free to choose anywhere and anytime to collect data and accumulate scores.

6.1.4 Contribution Assessment

To assess a participant's contribution, we propose a scoring method based on the informativeness of the paths or locations of data collection. Specifically, we model the spatial distribution of Wi-Fi signals by a Gaussian Process (GP) [58]. For a given B/F, we first divide the space into grids. Each grid point, denoted by its physical 2D coordinate $\mathbf{x} = (x_1, x_2)$, serves as a reference point. Let the set of grid points be $\mathcal{V} = \{\mathbf{x}^{(i)}\}_{i=1}^{p}$, where p is the number of grid points. Equivalently, we represent the grid points using a $p \times 2$ matrix $X_{\mathcal{V}}$. Denote the corresponding Wi-Fi signal strengths at these points by random variables $\mathbf{y}_{\mathcal{V}}$.

Now suppose that a participant collects data at a set of locations $X_{\mathcal{S}}$. For point-based collections, $X_{\mathcal{S}}$ only contains one location, i.e, the location where she stands. For path-based collections, we set $X_{\mathcal{S}}$ to be the locations where step events[8] occur along the paths. Typical step frequency is from 1Hz to 2Hz for healthy adults.

[8] A step event is defined as the beginning of the swing phase in a stride cycle for either foot.

Compared to using locations where Wi-Fi scans are actually obtained during the walk, which is highly device and environment dependent, doing so is more indicative of participants' efforts. We have implemented a peak detection based step counter and combine that with map information to infer locations of step events. Let the "observations" at $X_{\mathcal{S}}$ be $\mathbf{y}_{\mathcal{S}}$. For each data submission, we calculate the score by the decrement in the entropy of $y_{\mathcal{V}}$ as the result of $y_{\mathcal{S}}$. Formally,

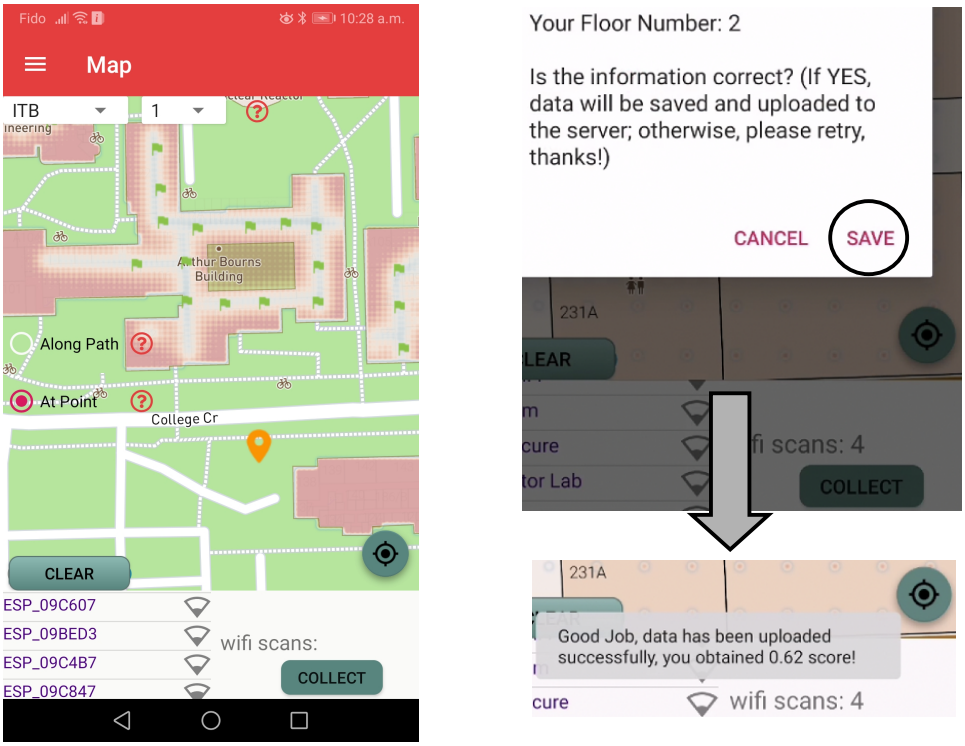
$$F(\mathbf{y}_{\mathcal{S}}) = \beta[H(\mathbf{y}_{\mathcal{V}}) - H(\mathbf{y}_{\mathcal{V}}|\mathbf{y}_{\mathcal{S}})] = \frac{\beta}{2}(ln|\Sigma_{\mathcal{V}}| - ln|\Sigma'_{\mathcal{V}}|), \tag{6.1.1}$$

where β is a scaling factor that can be used to balance the scores for path-based and point-based options.

The score determined by (6.1.1) only depends on the scaling factor, the kernel parameters, past and current survey locations. Actual RSS values or device characteristics do not affect the assessment of contributions. A participant can accumulate more scores if she collects data in under-explored areas. Staying at the same area or places that have been extensively covered by others is discouraged. Compared with simple metrics such as the number of scans or the time spent by users, (6.1.1) is by design not only a fairer metric to participants but also beneficial to improve the coverage of the campaign. The latter aspect is supported to some extent by the user study we conduct in Section 6.3.

6.1.5 Heatmap

To illustrate areas that have been surveyed and to incentivize participants to collect data from under-explored areas, a heatmap layer is stacked on floor maps. Specifically, for each grid point $v \in \mathcal{V}$, the predictive variance of RSS values at the location can be



(a) Heatmap (b) Submission and feedback

Figure 6.3: Heatmap and data submission interface

calculated by (3.1.5) with $X_{\mathcal{V}}$ replaced by $\mathbf{x}_v$. The heatmap layer is then generated using these variances. Areas with a deep red color indicate that they have not been surveyed before. The effect is shown in Fig. 6.3 a.

6.2 The Site Survey Campaign

This section describes the campaign. First, we explain the deployment of the proposed MCS system, and then we discuss in detail the incentives we adopted to stimulate students' contribution and recruitment strategies.

6.2.1 Campaign Scenario

The campaign was launched in the McMaster University Campus. Before the final release, internal tests and a small-scale user study were conducted to fine-tune parameters and design choices. Table 6.1 summarizes the parameters used in the scoring function. Under such settings, the estimated time to reach a score of 30 (the threshold

Table 6.1: Parameter Settings

Parameter	Description	Value
grid interval	distance between grid points	2.0 (meter)
σ_f^2	maximum variance of the GP	50.0
α	length scale of the GP	20.0
σ_n^2	noise variance of the GP	10.0
β_1	scaling factor of scores (point-based)	1.0
β_2	scaling factor of scores (path-based)	3.0

to redeem reward) is roughly between 15 and 30 minutes including the registration process. The resulting area covered approximately equals to the corridors of one floor in a mid-sized building on our campus if the path-based option is adopted. Other factors that can affect the time cost or score accumulation include the collection method and whether the participant is familiar with the area. Participants may spend some time to localize themselves on the map if they are not familiar with the buildings they are at. We observed in internal testings that path-based data collections generally accumulate scores faster than point-based collections.

6.2.2 Incentive Mechanisms and Recruitment Strategies

Extensive research has been conducted on the incentive mechanisms for MCS systems [46, 117]. During the campaign, we utilize monetary rewards and a reputation

system based on the leader board in the App, which can also be seen as an entertainment incentive. Any participant who reaches a score no less than 30 can redeem a $10 gift card. Next, we introduce two recruitment strategies.

6.2.2.1 Active Recruitment with Teaching

Starting from September 2019, we actively recruited volunteers in the university by setting up booths during the welcome week and through personal connections. For those who expressed interests and requested further information, we demonstrated in person the data collection procedure using a smartphone and provided tips such as where to collect data and the difference between the path-based and point-based options. This group of participants did not need to go through the tutorial or figure out the process by themselves. From September 2019 to the first week of December 2019, in total, we have reached out to 42 participants in the university, and 17 of them have signed up.

Active recruitment with teaching is advantageous in that there is almost no learning cost to participants. Additionally, face-to-face interaction allowed better understandings of their interests and willingness. However, for such a strategy, it was time-consuming to demonstrate and teach each potential participant details of the campaign.

6.2.2.2 Passive Recruitment by Posters

Another recruitment strategy is through posters, which were posted to public bulletin boards inside buildings across the campus. An example is shown in Fig. 6.4. A would-be participant simply scans the QR code on the poster, which directs them to the App

Figure 6.4: A poster attached to a bulletin board

in Google Play. Passive recruitment has the potential to reach out to more people. However, the drawback is obvious. Participants need to spend time learning the data collection process on their own with the help of the in-app tutorial. Although we tried our best to lower the learning curve, it may still require some efforts to understand the purpose of the campaign and the proper procedure. This can be discouraging, especially for those without sufficient motivation and interest who may quit after registration.

6.2.2.3 Recruitment Results

From September 2019 to the end of January 2020, 97 participants signed up in total (including the 17 participants who agreed to participate and had been taught by us in person).

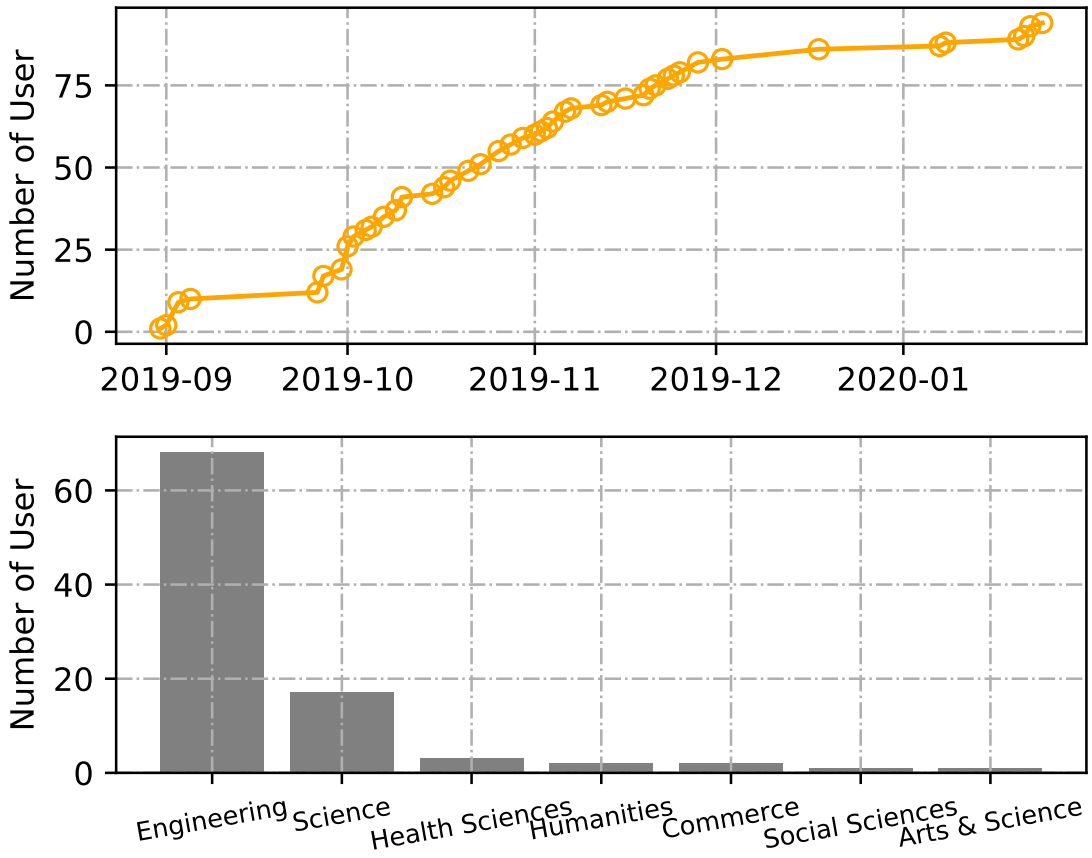


Figure 6.5: Participant number and department distribution.

Fig. 6.5 shows the total number of registrations over time. Most participants registered in October and November, 2019. In December, there were fewer registrations since it is time for final exams and participants do not have much extra time. Starting from January, 2020, new registrations were obtained, though the number was smaller. Part of the reason is that in the beginning of the new semester, posters were removed by the university, and thus information about the campaign is difficult to find. Meanwhile, as shown in Fig. 6.5 , majority of participants are from the Faculty of Engineering or the Faculty of Science.

6.3 Data Statistics and Analysis

In this section, we analyze the crowdsourced data from the campaign by the end of Jan. 2020. We first present the characteristics of user scores and fingerprints. Then we discuss data collection behaviors and feedback from participants.

6.3.1 Scores Attained

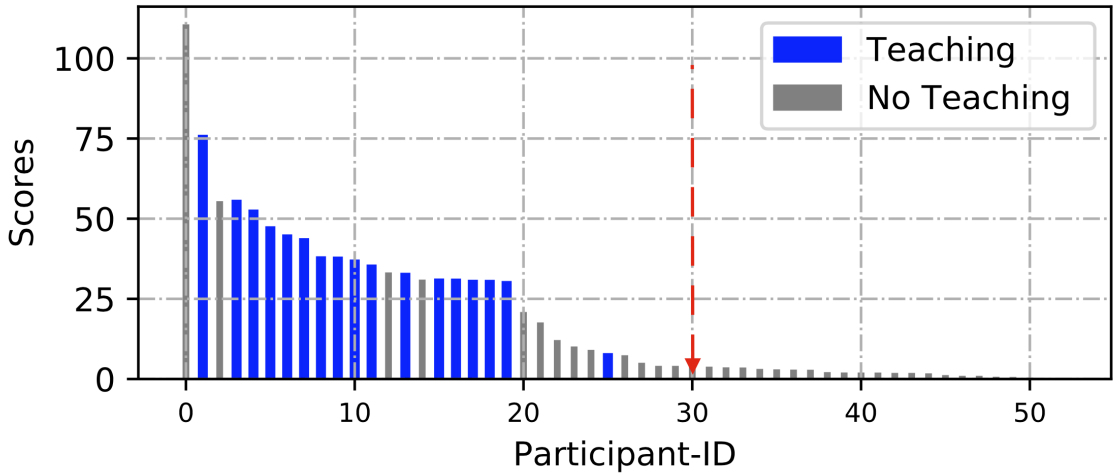


Figure 6.6: Scores attained by participants

Fig. 6.6 shows the score attained by participants by Jan. 26, 2020. Recall that the monetary reward can be redeemed with scores exceeding 30. In the figure, an obvious cut-off point can be seen at the score of 30. In total, 20 participants reached the threshold. Among them, around half stopped making further contributions after reaching the threshold. However, the other half continued to collect more data even though they were already eligible for the reward. Particularly, the top participant has been making persistent and significant contributions. On the other hand, around 30 participants had scores below threshold. 31 registered participants failed to submit

any data.

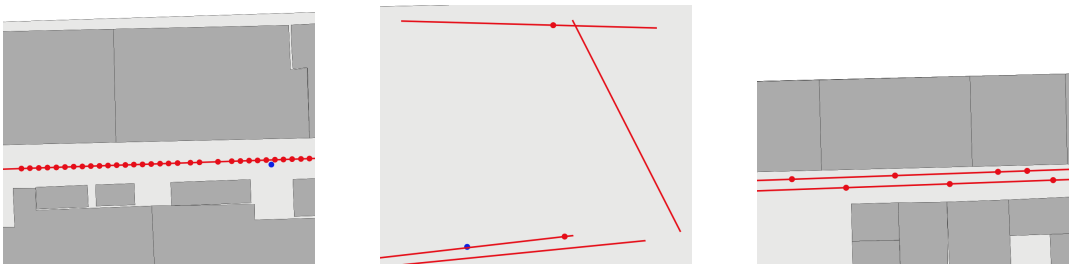
Overall, participants who have been taught achieve higher scores. Among the 20 participants that reached the threshold score, 16 of them were from active recruitment with teaching. Interestingly, the first place and the third place were participants recruited through posters, although most participants in this group had low scores.

6.3.2 Fingerprint Characteristics

With a slight abuse of notation, we use a vector $\mathbf{y} = (y_1, y_2, ..., y_M)$ to represent a Wi-Fi fingerprint, where M is the total number of APs and y_i is the RSS readings of the ith AP. The corresponding location label is $\mathbf{x}$. For point-based collections, if there are multiple scans, the RSS readings are averaged. Therefore, each point-based collection only corresponds to one single fingerprint. For path-based collections, since participants are moving, we define that each Wi-Fi scan corresponds to a fingerprint. Theoretically, the arrival time of signals within a single scan is different[9]. However, we observed that the difference is quite small (less than half a second) and can be neglected. Thus, it is reasonable to construct a fingerprint from the respective RSS readings, and its location tag is inferred using the average arrival timestamp. On the other hand, we notice that the time interval between two consecutive scan varies a lot among different devices.

Fig. 6.7 shows the traces from three devices from three participants. From all fingerprints obtained, we find that Samsung smartphones with an Android API Level under 28 are good options for site survey. They have a higher scan frequency resulting in a denser distribution of fingerprints. On the other hand, in Android 9 (Android

[9]https://developer.android.com/guide/topics/connectivity/wifi-scan



(a) Samsung, SM-A520W, Android API Level 26

(b) OnePlus, A6000, Android API Level 28

(c) Huawei Honor, JSN-L23, Android API Level 27

Figure 6.7: Locations of received Wi-Fi scans during walking from three devices. Red lines represent paths, and red dots represent locations of scans (inferred from the timestamps and step counts). Blue dots represent point-based collection.

API Level 28), due to the throttling of Wi-Fi scans, only four scans are permitted every two minutes for each foreground app. This leads to a sparse distribution of fingerprints along paths. As shown in Fig. 6.7b, there are even paths without any scan.

Fig. 6.8 depicts the accumulation of fingerprints over time and distribution among participants. The general trend is similar to that of the registration as shown in Fig. 6.5. By the end of January 2020, a total of 1400 fingerprints have been uploaded. Specifically, actively recruited participants contributed 820 fingerprints, and the remaining 584 fingerprints were collected by others. Note that the distribution is different from that of user scores, since the number of fingerprints depends on both the device used and the collection method. There are two participants with a limited number of fingerprints (less than 10), although they attained larger scores. They both have phones with Android 9 (HUAWEI EML-L09; HONOR BKL-AL00). Although the number of scans is constrained by the mobile operating system, the scores are calculated based on the informativeness of the paths as discussed in Section 6.1.4. On

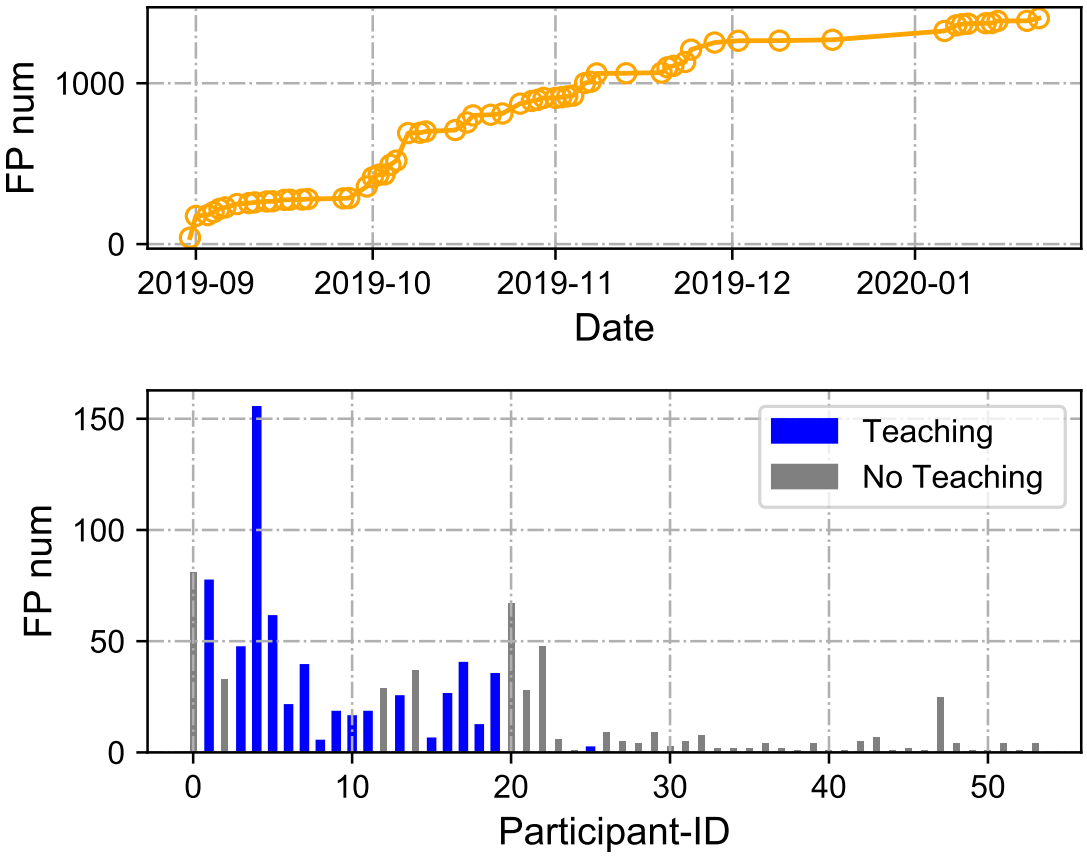
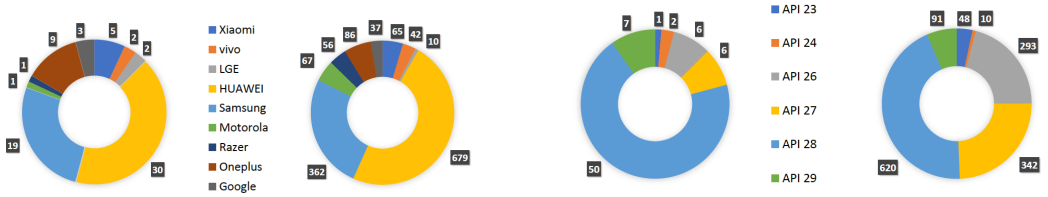


Figure 6.8: The number of fingerprints overtime and distribution among participants.

the other hand, participants (e.g., User 48) with more fingerprints are not guaranteed with higher scores.

Fig. 6.9 illustrates the percentage of devices and collected fingerprints categorized by vendors and API Levels. We observe that Huawei is the dominate vendor followed by Samsung. On the other hand, around 3/4 of the devices run Android 9 (API Level 28). In contrast, the number of fingerprints from Android 9 devices is less than half of the total fingerprints, which is again due to the throttling of Android 9. Fig. 6.10



(a) Number of different models and the respective fingerprints

(b) Android API Level distribution and the respective fingerprints

Figure 6.9: Device (Model and Android API) distribution and the respective number of fingerprints

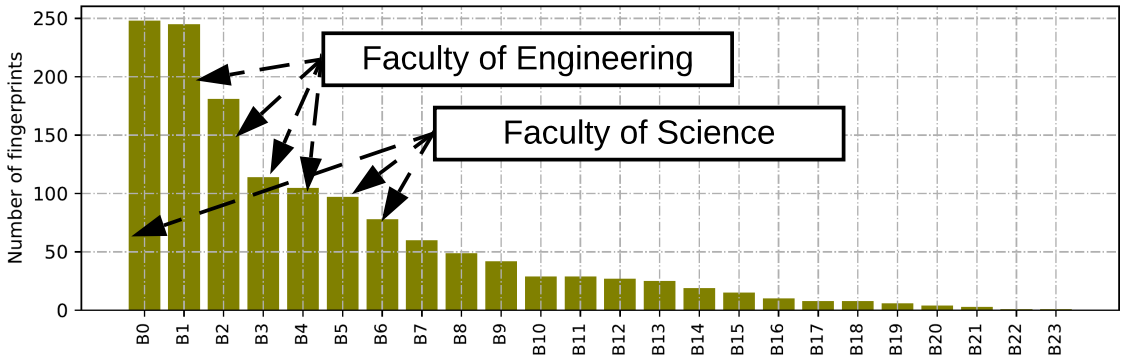


Figure 6.10: Fingerprint distribution by building

shows the fingerprint distribution by buildings. Most of the fingerprints are collected at the 7 buildings from the Faculty of Engineering or Science.

Finally, all fingerprint statistics are listed in Table 6.2.

6.3.3 Data Collection Behaviors

To investigate the reasons why many participants achieved low scores and made limited contributions, we conduct a detailed analysis on participants' data collection behavior based on upload records.

Table 6.2: Summary of Fingerprint Characteristics

Description	Value
Total number of scans	4856
Total number of fingerprints	1404
Total scan time	13.16 (hour)
Total number of buildings/floors	24/58
Total number of point-based collection rounds	228
Total number of path-based collection rounds	187 (8.6 KM)

6.3.3.1 Group based on Score Range

Firstly, we divide the participants into three groups according their scores. Participants in the first group achieve *Low* scores greater than 0 but less than 10 (those with no submission record have been excluded). Analyzing the behavior of this group helps us understand why they did not succeed in the campaign. In the study, this group of participants constitute more than half of all participants. If more in this group could be better engaged, more data would've been collected.

The second group is the *Borderline* group, with a score between 30 to 35. It is reasonable to hypothesize that this group of participants only care about monetary rewards. They did not continue or only do so shortly after meeting the threshold for rewards. The last group is the *High* score group with scores greater than 45. We have reasons to believe that they have other motivations besides monetary rewards.

Behavior characteristics of the three groups are listed in Table 6.3. Path-based data collections are more efficient since each data submission usually contains more informative information and thus yields higher scores. Another way to earn higher scores is to explore different buildings or floors especially ones that are under-covered.

Table 6.3: Participant Behavior Summary Grouped by Score Range.

	Low	Borderline	High
# of participants	30	9	6
# of paths / # of points	**16/70 (0.23)**	**58/34 (1.70)**	**72/47 (1.53)**
AVG # of submissions	2.77	10.22	19.83
AVG score/submission	**0.80**	**2.80**	**2.95**
AVG # of B/F explored	**1.32**	**3.77**	**7.16**
AVG # of active days	1.13	1.77	**3.8**
AVG # of active hours	1.3	2.44	**5.8**
# $T_{r2fs} < 1h$	28 (93.3%)	7 (77.7%)	6 (100%)
# $1h <= T_{r2fs} < 1d$	0	1 (11.1%)	0
# $1d <= T_{r2fs} < 7d$	1 (3.3%)	0	0
# $T_{r2fs} >= 7d$	1 (3.3%)	1 (11.1%)	0
# $T_{aspan} < 1h$	26 (86.6%)	3 (33.3%)	2 (33%)
# $1h <= T_{aspan} < 1d$	2 (6.6%)	2 (22.2%)	0
# $1d <= T_{aspan} < 7d$	0	2 (22.2%)	1(16%)
# $T_{aspan} >= 7d$	2(6.6%)	2(22.2%)	3(50%)

Both in the Borderline group and the High group, the path-based option is the dominate approach utilized. Each submission (regardless of the collection options) gains around 3 scores. The High (Borderline) group members explored 7.16 (3.77) buildings or floors on average . In contrast, participants in the Low group mainly use the point-based option, majority of whom stayed in one floor of one building. This leads to a limited average score (0.80) per submission. One possible reason is that this group of participants are not aware of the different options of data collection and the scoring mechanism because, for instance, they did not take the time to go through the in-app tutorial. Slow score accumulation may discourage such participants from further contributing to the campaign.

We further analyzed the temporal characteristics of data collection. Specifically, we define active days (or hours) as the number of days (or hours) that have data

Table 6.4: User Behavior Grouped by Teaching or without Teaching.

	without Teaching	with Teaching
# of participants	37	17
# ofpaths / # of points	**68/180 (0.37)**	**119/48 (2.47)**
AVG # of submissions	6.7	9.82
AVG score/submission	**1.27**	**3.54**
AVG # of B/F explored	**2.5**	**3.4**
AVG # of active days	1.75	1.64
AVG # of active days hours	2.5	2.29
# $T_{r2fs} < 1h$	34 (91.8%)	15 (88.2%)
# $1h <= T_{r2fs} < 1d$	1 (2.7%)	1 (5.8%)
# $1d <= T_{r2fs} < 7d$	1 (2.7%)	0
# $T_{r2fs} >= 7d$	1 (2.7%)	1 (5.8%)
# $T_{aspan} < 1h$	26 (70.2%)	11 (64.7%)
# $1h <= T_{aspan} < 1d$	3 (8.1%)	2 (11.7%)
# $1d <= T_{aspan} < 7d$	1 (2.7%)	2 (11.7%)
# $T_{aspan} >= 7d$	7(18.9%)	2(11.7%)

collection activity. We denote the time from registration to the first submission as T_{r2fs}, and the time time-span from the first submission to the last submission as T_{aspan} (Table 6.3). 86.6% of participants in the Low group spent less than one hour in the campaign, although 89.4% of them tried collection immediately after registration. In contrast, the High group are more engaged spending on average 5.8 active hours and 3.8 active days.

6.3.3.2 Groups with Teaching and without Teaching

Another perspective is to group the participants based on whether they were taught in person or not, and the results are shown in Table 6.4. Participants in the group with teaching mostly adopted the path-based option and explored more buildings/floors with an average score of 3.54 for each submission. In contrast, only a small fraction

of participants in the group without teaching used the path-based option with a per-submission score at 1.27 on average. This indicates that the group without teaching may not understand how to earn scores fast.

In the group without teaching, 91.8% of the participants made the first submission within an hour after registration, and the percentage with an activity time span less than one hour is 70.2%. The corresponding percentages in the teaching group are 88.2% and 64.7%, respectively. This indicates that the group without teaching were more likely to give up in the first hour.

Overall, the group of participants with teaching perform better in the campaign. It is known that face-to-face interactions and teaching promote commitments [26]. Except for one participant in this group, all other participants reached the threshold of monetary reward. However, the first and third places are participants without teaching. The number of covered buildings/floors and active hours for the two participants are (12,11) and (8,10), respectively.

6.3.4 Feedback from Participants

To further investigate the reasons behind participant behaviors, particularly among those passively recruited, we conducted a survey at the end of the campaign. Specifically, we sent an online questionnaire to all participants in this group and conducted a face-to-face interview with the first and third placed participants.

6.3.4.1 Interview

The 1st and 3rd placed participants are both recruited by posters. They expressed that financial reward was an important motivating factor for them to sign up for the

campaign. The 1st place participant also mentioned curiosity and fun. In terms of UI design of the App, both of them agreed that it was easy to understand and they knew where to collect data for more scores. Another commonality is that neither of them spent much time to collect data purposefully. Instead, they collected data when they got nothing to do and needed to kill time, for example, when waiting for someone or during class breaks.

After reaching the threshold for monetary reward, they both continued to make further contributions. The 3rd placed participant hoped to get more financial reward, while the 1st placed participant mentioned that he felt that contributing to the campaign is a good use of his spare time which would otherwise be wasted. Monetary reward was not the only motivation to him. Another difference is that the 3rd place participant only adopted the point-based option and hardly utilized the path-based option, while the 1st place participant utilized both options, although it took him some time to figure out the path-based option.

6.3.4.2 Questionnaire

In total, 15 passively recruited participants responded to the online questionnaire. The purpose of the questionnaire is three-fold, i) to evaluate the relevant knowledge learned by themselves, ii) when and where data collection takes place, iii) the reasons why participants failed to reach the reward threshold.

Firstly, we asked whether participants were comfortable to grant location and storage permissions to the App. 2/3 of them responded positively. Overall, the majority of participants claimed that they understood the purpose of the campaign (73%) and the tutorial was easy to follow (86%), although most of them (73%) mentioned that it

took some time to figure out the difference between the point-based and path-based options. However, only 60% of the participants claimed they knew where to get data to achieve high scores.

Most participants (66%) indicated that they perform data collection at locations coinciding with their daily activities (e.g., classes or meals). 26% of the participants chose locations based on the heatmap in the App. 11 participants agreed that $10 for 30 scores was reasonable, while 1 participant thought $10 was too little and 3 participants complained 30 scores were too time-consuming to attain.

Lastly, there are various reasons why participants fail to reach the reward threshold as shown in Fig. 6.11. The primary reason is that too few scores were obtained per submission. The secondary reason is that some participants were too busy. According to the feedback, increasing the per-submission score (or decreasing the threshold of monetary reward) may better motivate participation. However, doing so will pose a heavier financial burden to the MCS organizers. How to design incentive mechanisms for indoor data collection with better participant engagement and lower costs remains to be an important research topic.

6.4 Localization Experiments

In this section, we investigate localization performance utilizing the crowdsourced fingerprints. We first inspect data quality, and then present the localization method and performance analysis.

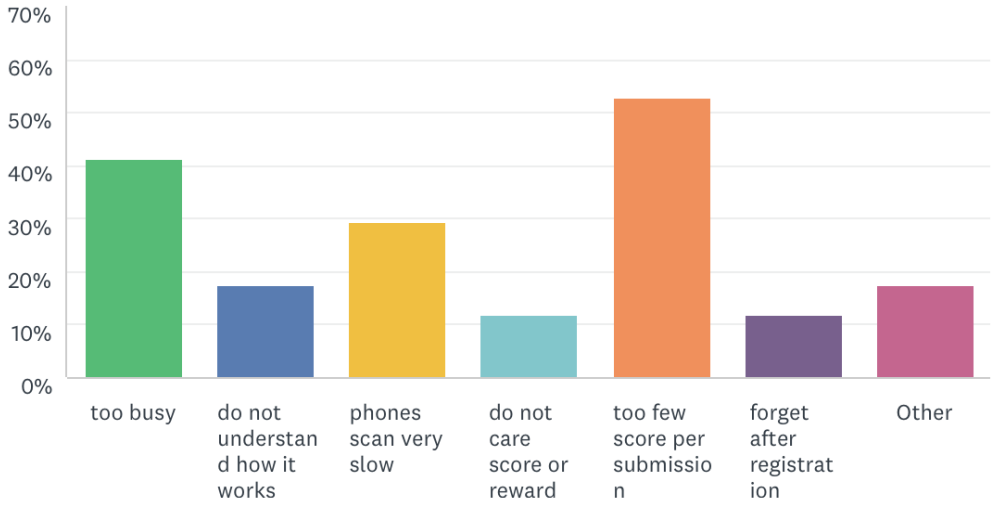


Figure 6.11: Reasons for failing to get the reward

6.4.1 Data Quality

When conducting MCS campaigns, it is crucial to ensure a certain level of data quality (or data validity) [32, 88]. The crowd can contribute inaccurate data or even provide falsified data on purpose. For instance, in [115], location frauds were detected during a crowdsourcing game. Incentive mechanisms themselves can be a double-edged sword as participants may cheat to get rewards fast. We call such kind of data errors *intentional* errors. Otherwise, errors that are *unintentional* if participants make mistakes accidentally.

We consider the following four types of data errors.

- *Location labels outside buildings*: location labels marked on the map are expected to be inside specific buildings. This type of errors can be unintentional or intentional, and can be detected using building perimeters (e.g., the start and terminal locations must be inside the perimeters).

- *Location frauds*: malicious participants may pretend that they are on campus though they are not. To detect such behaviors, we check if each data item contains campus specific AP SSIDs. Furthermore, other location services (GPS, Cellular network) are utilized to estimate the coarse locations of the participants using the App.

- *Abnormal walking patterns*: we advise participants to walk in a steady pace in the path-based collection. IMU sensors in smartphones are utilized to identify participants with abnormal walking patterns. Specifically, we implement a step counter, and calculate the average step length from the distance traveled and the step counted. According to [42], men walk at approximately 1.15 m/s and have mean step lengths of 0.66 m. Women walk at approximately 1.08 m/s and have mean step lengths of 0.57 m. Since the step length and walking speed change with gender, height and walking patterns, we loosly restrict step lengths between 0.3 m to 0.8 m, with a walking speed between 0.5 m/s to 1.6 m/s. Estimated values beyond this range are considered outliers.

- *Motion during point-based collections*: In the point-based option, participants are expected to remain at the same location. Similarly, we use data from IMU sensors to detect if they are moving. If three step events are detected during the collection period, the participant is classified as non-stationary.

We post-process uploaded data and detect these types of errors to ensure data validity. Table 6.5 shows the validation results. Among the total number of submissions, 16 submissions (3.8%) were wrongly labeled. No location fraud cases were detected. The number of abnormal path-based submissions and the number of point-based submissions with movements are 39 and 38, respectively. Furthermore, the

Table 6.5: Data Validity Inspection Result

error type	# submission	w/o teaching: teaching
labels outside buildings	16 (3.8%)	9:7
location fraud	0	-
abnormal walking	39 (20.8 %)	25:14
movements	38 (16.6%)	37:1

Table 6.6: Training and Test Data Information

B/F	training data (# FP, # AP)	# test locations
1	(78, 125)	6
2	(34, 106)	10
3	(40, 110)	10
4	(47, 235)	9
5	(55, 263)	10
6	(144, 144)	8
7	(75, 115)	7

results also show that the group of participants with teaching made fewer mistakes.

6.4.2 Test Data Collection

Seven floors from five buildings were evaluated. Test data were collected using the point-based option by one of our developers. No professional measurement devices were employed. To be close to the ground truth, we chose test locations near landmarks such as elevators, stairs, starts (or turns) of corridors, room boundaries, etc. Such locations can be accurately labeled on the map in general. Three mobile phones were utilized for testing, namely "Mate 10", "MI4 LTE" and "Honor 8X". The training data size and the number of test locations in each floor is summarized in Table 6.6. At each test location, only one Wi-Fi scan is captured by each test device and used in testing.

6.4.3 Localization Method

We adopt the Gaussian Process (GP) regression based localization algorithm in [64]. Specifically, RSS readings from each AP are processed separately, and are fitted using GP regression models. Following the notations in Section 6.1, let the RSS measurements at locations $X_{\mathcal{S}}$ be $\mathbf{y}_{\mathcal{S}}$. Hyperparameters of the GP of an AP are optimized by maximizing the log-likelihood

$$log\ p(\mathbf{y}_{\mathcal{S}}|X_{\mathcal{S}}) = -\frac{1}{2}(\mathbf{y}_{\mathcal{S}} - m(X_{\mathcal{S}}))^T(K(X_{\mathcal{S}}, X_{\mathcal{S}}) + \sigma_n^2 I)^{-1} (\mathbf{y}_{\mathcal{S}} - m(X_{\mathcal{S}})) - \frac{1}{2}log|K(X_{\mathcal{S}}, X_{\mathcal{S}}) + \sigma_n^2 I| - \frac{n}{2}log2\pi. \quad (6.4.1)$$

The posterior RSS distribution at each reference point $v \in \mathcal{V}$ is $\mathcal{N}(\mu'_v, \sigma'^2_v)$ and can be obtained from (3.1.4) and (3.1.5) (by replacing $\mathcal{V}$ with a single point v). Given a test fingerprint observation $\mathbf{o} = (o_1, o_2, ..., o_M)$, a straightforward solution is to search $\mathcal{V}$ and select the point that has the maximum probability to generate $\mathbf{o}$ as the predicted location.

However, due to likely mismatch [81] between the devices to be localized and ones used during site survey, localization performance can be degraded. To compensate for the RSS differences due to different antenna gains, we perform location estimation and device calibration simultaneously. Specifically, assume that the expected RSS distribution from an AP at $v \in \mathcal{V}$ follows a Gaussian distribution $\mathcal{N}(\mu^*_v, \sigma^{*2}_v)$, where μ^*_v, σ^{*2}_v are the unknown mean and variance. Under the assumption that the pair-wise antenna gains of two different devices can be modeled by a linear transformation [81], an RSS measurement y from a site survey device can be mapped to that of a test

device using

$$y^* = a \times y + b, \tag{6.4.2}$$

where y^* is the transformed RSS, a and b are the coefficient and bias to be estimated, respectively. Then we have

$$\mu_v^* = a \times \mu_v' + b, \tag{6.4.3}$$

$$\sigma_v^{*2} = a^2 \times \sigma_v'^2. \tag{6.4.4}$$

Given $\mathbf{o}$, the parameters a, b and location v can be estimated jointly as

$$v = \underset{v \in \mathcal{V}, a, b}{\operatorname{argmax}} \; log \prod_{j=1}^{M} p^*(o_j | v, a, b), \tag{6.4.5}$$

where $p^*(o_j)$ is the respective probability density of the jth AP.

To solve (6.4.5), for each $v \in \mathcal{V}$, we maximize $log \prod_{j=1}^{M} p(o_j | a, b)$ with respect to a, b. We constrain $a \in [0.8, 1.2]$ and $b \in [-20, 20]$ dBm since Wi-Fi devices need to pass compliance tests and thus their gains are expected not to deviate too much. The final predicted location is chosen to be the reference point with the maximum log-likelihood satisfying the constraints.

6.4.4 Localization Performance

The localization performance is summarized in Table 6.7. It can be seen that the localization performance varies with specific buildings/floors and test devices. The overall average localization error is around 9.6 m. The performance is comparable to the state-of-the-art results using a crowdsourced Wi-Fi fingerprint dataset in [67].

Table 6.7: Localization Error (Meter).

B/F	MI4LTE		Mate 10		Honor 8X	
	Average	80%ile	Average	80%ile	Average	80%ile
1	6.04	10.40	8.52	15.40	4.97	5.80
2	16.27	27.49	12.78	14.97	11.86	23.81
3	11.55	16.79	6.41	9.00	6.45	8.30
4	9.3	12.73	8.06	10.99	11.92	15.86
5	8.08	11.27	11.8	14.79	7.03	10.00
6	10.94	18.22	5.44	9.05	6.98	10.09
7	12.40	16.87	15.39	25.56	8.05	12.49
All	**10.96**	16.73	**9.6**	11.99	**8.32**	12.25

The dataset in [67] was collected by eight volunteers (21 devices) point-by-point in a controlled manner, and no data quality issues were reported. In contrast, our data collection is done in the "wild" with more device heterogeneity.

With crowdsourced Wi-Fi fingerprints, many factors could affect the localization performance. We summarize the main reasons as follows.

- *Imbalanced data distribution.* Unlike controlled site surveys where locations are planned ahead, survey locations from the crowd are unlikely to be uniform. Although by design we encourage participants to collect fingerprints in a distributed manner spatially using the scoring mechanism and the heatmap, many areas remain under-explored. Among areas that have been surveyed, the density of fingerprints also varies a lot.

- *Labeling accuracy.* Location labels in point-based collections are provided by participants by marking on the map directly. Errors may occur if participants make mistakes. For the path-based option, in addition to errors in the start and terminal locations of paths, accuracy of location labels is also affected by

variations in walking patterns.

- *Device heterogeneity.* The devices used for contributing data in the campaign and for localization belong to the participants and are heterogeneous. Consequently, they may have different antenna characteristics.

- *Other environmental factors.* The dimensions (relative to fingerprint densities) and structural complexity of the buildings, the number of available Wi-Fi APs can contribute to localization accuracy as well. Furthermore, Wi-Fi signals are noisy and time varying, and may differ with different phone orientations and presence of other pedestrians nearby, etc.

6.5 Limitations and Lessons Learned

During the data collection campaign, one major limitation comes from the throttling of Wi-Fi scans in Android 9. When the platform was under development, Android 9 was not yet the prevalent OS among Android devices. By May 2019, only 10% of the total Android devices was running Android 9. However, our statistics (Figure 6.9) show that the percentage was higher among participants on our campus. As a result, though Android 9 participants spent comparable amount of time as the participants using other Android versions, they contributed much fewer fingerprints. If throttling were not in place, the total number of fingerprints would have been much higher. This limitation has been rectified in Android 10, in which users are given the option to turn the throttling off.

Most participants signed up for the campaign because of the monetary reward, although many of them did not reach the threshold necessary for the reward at the

end. In particular, it happened for those passively recruited participants without teaching. From the analysis of user survey in Section 6.3.4, the reason is twofold. First, the reward was not high enough to stimulate a consistent contribution from some participants. Second, the learning curve to use the App deters some users from contributing. In this case, an interactive tutorial can be helpful.

Another limitation of the current study is that due to limited budgets, each participant is limited to a fixed reward (e.g., a $10 gift card). Further contributions beyond the threshold of 30 scores are not rewarded. This has a negative impact on participants' willingness for continuing involvements as evident from Fig. 6.6 that many participants stopped right around attaining the threshold scores.

6.6 Conclusion

In this chapter, we presented IFCS, an MCS participatory system for indoor location fingerprint collection. We proposed to evaluate participants' contributions through a scoring mechanism based on their informativeness. Unlike existing works, we focused on participant recruitment and behavior analysis. Localization models trained using the crowdsourced fingerprints can achieve a performance comparable to state-of-the-art results. The experience and problems revealed during the campaign are beneficial to the development of spatial data crowdsourcing.

Chapter 7

Concluding Remarks

7.1 Conclusion

In this thesis, in order to improve the efficiency of spatial data collection, we considered two mobile sensing based solutions, namely mobile robotic sensing and mobile crowdsourcing.

For mobile robotic sensing, we formulated the IPP problem to plan data collection paths. We proposed a Greedy algorithm and a Genetic Algorithm based on a graph. Experiments on Wi-Fi fingerprints collection showed that paths with a higher utility are more likely to achieve lower localization errors, which demonstrated that IPP is an effective solution for planning site survey paths.

In addition, when budgets change in the same target area, we presented a RL framework for informative path planning. To address the unique challenges posed by path constraints, a novel action selection mechanism is designed with the assistance of the shortest paths. Under the framework, we implemented Q-learning, actor-critic and the reinforce algorithm, and compared with other state-of-the-art non-learning

based algorithms. The RL based solutions achieve better path utility in most cases, and is much more efficient since they only need to run the inference process using the pre-trained model.

We also investigated the multi-robot cooperative path planning problem based on reinforcement learning. Specifically, we devised two solutions, namely, individual learning based on credit assignment and sequential rollout. With both schemes, the learned policy is utilized to plan paths and achieves higher or similar rewards compared with the baseline solutions. More importantly, the RL-based solutions can handle different parameter settings during inference without retraining.

Finally, to study user behavior of crowdsourcing based data collection, we designed IFCS and launched a real-world campaign. We proposed to evaluate participants' contributions through a scoring mechanism based on informativeness. Unlike existing works, we focused on participant recruitment and behavior analysis. Two recruitment strategies were adopted and evaluated. We demonstrated both the feasibility and the practicality to build an MCS-based indoor localization system through a real-world campaign.

7.2 Future Work

In this section, we outline several future research directions. Specifically, we discuss further work on mobile robotic sensing, and then highlight aspects that can be enhanced for crowdsourcing-based data acquisition.

First, for the RL based path planning, one promising direction is to investigate alternative state encoding schemes. Currently, states s in the RL formulations in Chapter 4 encapsulate all the historical information through an RNN. One potential

drawback is that the performance of the model may degrade when the number of time steps grows large. One possible solution to this problem is to encode states using information only in the current step and leverage graph neural networks in the representation.

Second, in our problem formulation, a path is represented as a sequence of vertices in a graph. This is applicable in applications (e.g., collecting air quality data in a city) with pre-existing graph structures as such road networks in a target area. In other applications (e.g., detecting biomass contained in lakes), the target areas are often open spaces, in which robots can travel in any direction. Although constructing a grid graph is still feasible, it is desirable to plan paths directly in the continuous 2D or 3D space. Thus, another research direction is to consider RL based path planning in continuous action space. A robot needs to learn which direction to move and how far to move in each decision step, as well as how to avoid obstacles.

Third, in our current work, actual measurements do not affect path utility. In practice, actual measurements can be crucial to specific applications. For instance, in indoor localization, the actual measurements (Wi-Fi RSS) are used to differentiate locations. Collecting data along informative paths based on MI alone does not guarantee constructing models with smaller localization errors. Thus, one may need to adjust or re-plan paths as an on-going process based on collected data, known as adaptive sampling [41]. In the future, we are interested in investigating how to collect data adaptively using robots so as to reduce localization errors.

Lastly, MCS based data collection requires further investigation on effective incentive mechanisms to recruit and retain more participants. According to our experiences, a better UI design such that participants can easily learn how to make a contribution can also improve the campaign outputs. Another challenge in MCS-based data acquisition is to evaluate the effectiveness of a proposed solution. Simulation results are generally not convincing since human behaviors cannot be accurately modeled. To make evaluation results statistically significant, multiple rounds of real world experiments need to be carefully designed with controlled groups. Further research on experimental designs for MCS are warranted.

Bibliography

[1] Acer, U. G., Broeck, M. v. d., Forlivesi, C., Heller, F., and Kawsar, F. (2019). Scaling crowdsourcing with mobile workforce: A case study with belgian postal service. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **3**(2).

[2] Agogino, A. and Turner, K. (2005). Multi-agent reward analysis for learning in noisy domains. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 81–88.

[3] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer networks*, **38**(4), 393–422.

[4] Arora, S. and Scherer, S. (2017). Randomized algorithm for informative path planning with budget constraints. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4997–5004. IEEE.

[5] Bahl, P. and Padmanabhan, V. N. (2000). RADAR: An in-building RF-based User Location and Tracking System. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784. Ieee.

[6] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

[7] Bengio, Y. and CA, M. (2015). Rmsprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390*.

[8] Binney, J., Krause, A., and Sukhatme, G. S. (2010). Informative Path Planning for An Autonomous Underwater Vehicle. In *2010 IEEE International Conference on Robotics and Automation*. IEEE.

[9] Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210.

[10] Brunato, M. and Battiti, R. (2005). Statistical learning theory for location fingerprinting in wireless LANs. *Computer Networks*, **47**(6), 825–845.

[11] Busoniu, L., Babuska, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **38**(2), 156–172.

[12] Capponi, A., Fiandrino, C., Kliazovich, D., Bouvry, P., and Giordano, S. (2017). A cost-effective distributed framework for data collection in cloud-based mobile crowd sensing architectures. *IEEE Transactions on Sustainable Computing*, **2**(1), 3–16.

[13] Capponi, A., Fiandrino, C., Kantarci, B., Foschini, L., Kliazovich, D., and Bouvry, P. (2019). A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities. *IEEE Communications Surveys & Tutorials*, **21**(3), 2419–2465.

[14] Catovic, A. and Sahinoglu, Z. (2004). The Cramer-Rao Bounds of Hybrid TOA/RSS and TDOA/RSS Location Estimation Schemes. *IEEE Communications Letters*, **8**(10), 626–628.

[15] Chekuri, C. and Pal, M. (2005). A Recursive Greedy Algorithm for Walks in Directed Graphs. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 245–253. IEEE.

[16] Chen, D. and Shin, K. G. (2019). Turnsmap: Enhancing driving safety at intersections with mobile crowdsensing and deep learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **3**(3).

[17] Chen, H., Li, F., Hei, X., and Wang, Y. (2018). Crowdx: Enhancing automatic construction of indoor floorplan with opportunistic encounters. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **2**(4).

[18] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[19] Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, **1998**(746-752), 2.

[20] Cook, G. and Zhang, F. (2020). *Mobile Robots: Navigation, Control and Sensing, Surface Robots and AUVs*. John Wiley & Sons.

[21] Cook, W. (since 2015). Concorde TSP solver. `http://www.math.uwaterloo.ca/tsp/concorde/`.

[22] Cooke, M., Wei, Y., Hao, Y., and Zheng, R. (2018). Ilos: a data collection tool and open datasets for fingerprint-based indoor localization. In *Proceedings of the First Workshop on Data Acquisition To Analysis*, pages 15–16.

[23] Della Rosa, F., Pelosi, M., and Nurmi, J. (2012). Human-induced Effects on RSS Ranging Measurements for Cooperative Positioning. *International Journal of Navigation and Observation*, **2012**.

[24] Fang, Y., Deng, Z., Xue, C., Jiao, J., Zeng, H., Zheng, R., and Lu, S. (2015). Application of an Improved K Nearest Neighbor Algorithm in WiFi Indoor Positioning. In *China Satellite Navigation Conference (CSNC) 2015 Proceedings: Volume III*, pages 517–524. Springer.

[25] Fazl-Ersi, E. and Tsotsos, J. K. (2009). Region classification for robust floor detection in indoor environments. In *International Conference Image Analysis and Recognition*, pages 717–726. Springer.

[26] Feng, W., Yan, Z., Zhang, H., Zeng, K., Xiao, Y., and Hou, Y. T. (2017). A survey on security, privacy, and trust in mobile crowdsourcing. *IEEE Internet of Things Journal*, **5**(4), 2971–2992.

[27] Ferris, B., Hähnel, D., and Fox, D. (2006). Gaussian processes for signal strength-based location estimation. In *Robotics: Science and Systems II, August 16-19, 2006. University of Pennsylvania, Philadelphia, Pennsylvania, USA*.

[28] Figuera, C., Rojo-Álvarez, J. L., Wilby, M., Mora-Jiménez, I., and Caamaño, A. J. (2012). Advanced support vector machines for 802.11 indoor location. *Signal Processing*, **92**(9), 2126–2136.

[29] Gentile, C., Alsindi, N., Raulefs, R., and Teolis, C. (2012). *Geolocation Techniques: Principles and Applications*. Springer Science & Business Media.

[30] Golden, B. L., Levy, L., and Vohra, R. (1987). The orienteering problem. *Naval Research Logistics (NRL)*, **34**(3), 307–318.

[31] Gong, W., Zhang, B., and Li, C. (2018). Task assignment in mobile crowdsensing: Present and future directions. *IEEE network*, **32**(4), 100–107.

[32] Gong, X. and Shroff, N. (2018). Incentivizing truthful data quality for quality-aware mobile data crowdsourcing. In *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 161–170.

[33] GPy (since 2012). GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

[34] Grefenstette, J., Gopal, R., Rosmaita, B., and Van Gucht, D. (1985). Genetic Algorithms for the Traveling Salesman Problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, pages 160–168.

[35] Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **42**(6), 1291–1307.

[36] Guestrin, C., Krause, A., and Singh, A. P. (2005). Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272. ACM.

[37] Gunawan, A., Lau, H. C., and Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, **255**(2), 315–332.

[38] Gupta, P., Bharadwaj, S., Ramakrishnan, S., and Balakrishnan, J. (2014). Robust floor determination for indoor positioning. In *2014 Twentieth National Conference on Communications (NCC)*, pages 1–6. IEEE.

[39] Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15.

[40] Heinonen, M., Mannerström, H., Rousu, J., Kaski, S., and Lähdesmäki, H. (2016). Non-stationary gaussian process regression with hamiltonian monte carlo. In *Artificial Intelligence and Statistics*, pages 732–740. PMLR.

[41] Hitz, G., Galceran, E., Garneau, M.-È., Pomerleau, F., and Siegwart, R. (2017). Adaptive continuous-space informative path planning for online environmental monitoring. *Journal of Field Robotics*, **34**(8), 1427–1449.

[42] Hollman, J. H., McDade, E. M., and Petersen, R. C. (2011). Normative spatiotemporal gait parameters in older adults. *Gait & posture*, **34**(1), 111–118.

[43] Holt, C. A. and Roth, A. E. (2004). The nash equilibrium: A perspective. *Proceedings of the National Academy of Sciences*, **101**(12), 3999–4002.

[44] Hu, J. and Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, **4**(Nov), 1039–1069.

[45] Ichikari, R., Ruiz, L. C. M., Kourogi, M., Kurata, T., Kitagawa, T., and Yoshii, S. (2015). Indoor floor-level detection by collectively decomposing factors of atmospheric pressure. In *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–11. IEEE.

[46] Jaimes, L. G., Vergara-Laurens, I. J., and Raij, A. (2015). A survey of incentive techniques for mobile crowd sensing. *IEEE Internet of Things Journal*, **2**(5), 370–380.

[47] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, **4**, 237–285.

[48] Kaemarungsi, K. and Krishnamurthy, P. (2004). Properties of Indoor Received Signal Strength for WLAN Location Fingerprinting. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 14–23. IEEE.

[49] Kapetanakis, S. and Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. *AAAI/IAAI*, **2002**, 326–331.

[50] Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K., and Zabielski, P. (2012). Genetic Algorithm Solving Orienteering Problem in Large Networks. In *KES*, pages 28–38.

[51] Keller, C. P. (1989). Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, **41**(2), 224–231.

[52] Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358.

[53] Kim, K., Zabihi, H., Kim, H., and Lee, U. (2017). Trailsense: A crowdsensing system for detecting risky mountain trail segments with walking pattern analysis. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **1**(3).

[54] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

[55] Krause, A. and Guestrin, C. (2007a). Near-optimal observation selection using submodular functions. In *AAAI*, volume 7, pages 1650–1654.

[56] Krause, A. and Guestrin, C. (2007b). Nonmyopic active learning of gaussian processes: an exploration-exploitation approach. In *Proceedings of the 24th international conference on Machine learning*, pages 449–456.

[57] Krause, A., Singh, A., and Guestrin, C. (2008). Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, **9**(Feb), 235–284.

[58] Kumar, S., Hegde, R. M., and Trigoni, N. (2016). Gaussian process regression for fingerprinting based localization. *Ad Hoc Networks*, **51**, 1–10.

[59] Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., and Dizdarevic, S. (1999). Genetic Algorithms for the Travelling Salesman Problem: A review of Pepresentations and Operators. *Artificial Intelligence Review*, **13**(2), 129–170.

[60] Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer.

[61] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, **521**(7553), 436–444.

[62] Ledlie, J., Park, J.-g., Curtis, D., Cavalcante, A., Camara, L., Costa, A., and Vieira, R. (2012). Molé: a scalable, user-generated WiFi positioning engine. *Journal of Location Based Services*, **6**(2), 55–80.

[63] Letchford, A. N., Nasiri, S. D., and Theis, D. O. (2013). Compact formulations of the steiner traveling salesman problem and related problems. *European Journal of Operational Research*, **228**(1), 83–92.

[64] Li, C., Xu, Q., Gong, Z., and Zheng, R. (2017). Turf: Fast data collection for fingerprint-based indoor localization. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8. IEEE.

[65] Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

[66] Liu, L., Liu, W., Zheng, Y., Ma, H., and Zhang, C. (2018). Third-eye: A mobilephone-enabled crowdsensing system for air quality monitoring. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **2**(1).

[67] Lohan, E. S., Torres-Sospedra, J., Leppäkoski, H., Richter, P., Peng, Z., and Huerta, J. (2017). Wi-fi crowdsourced fingerprinting dataset for indoor positioning. *Data*, **2**(4), 32.

[68] Luo, T., Kanhere, S. S., Huang, J., Das, S. K., and Wu, F. (2017). Sustainable incentives for mobile crowdsensing: Auctions, lotteries, and trust and reputation systems. *IEEE Communications Magazine*, **55**(3), 68–74.

[69] Ma, K.-C., Liu, L., and Sukhatme, G. S. (2017). Informative planning and online learning with sparse gaussian processes. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4292–4298. IEEE.

[70] MacDonald, R. A. and Smith, S. L. (2019). Active sensing for motion planning in uncertain environments via mutual information policies. *The International Journal of Robotics Research*, **38**(2-3), 146–161.

[71] Mardle, S., Pascoe, S., *et al.* (1999). An Overview of Genetic Algorithms for the Solution of Optimisation Problems. *Computers in Higher Education Economics Review*, **13**(1), 16–20.

[72] Matignon, L., Laurent, G. J., and Le Fort-Piat, N. (2007). Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE.

[73] Matignon, L., Laurent, G. J., and Le Fort-Piat, N. (2012). Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems.

[74] Mehmood, H., Tripathi, N. K., and Tipdecho, T. (2010). Indoor Positioning System Using Artificial Neural Network. *Journal of Computer science*, **6**(10), 1219.

[75] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[76] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[77] Nguyen, D. T., Kumar, A., and Lau, H. C. (2018). Credit assignment for collective multiagent rl with global rewards. In *Advances in Neural Information Processing Systems*, pages 8102–8113.

[78] Ouyang, R., Low, K. H., Chen, J., and Jaillet, P. (2014). Multi-robot active sensing of non-stationary gaussian process-based environmental phenomena.

[79] Palmer, G., Tuyls, K., Bloembergen, D., and Savani, R. (2017). Lenient multi-agent deep reinforcement learning. *arXiv preprint arXiv:1707.04402*.

[80] Park, J.-g., Charrow, B., Curtis, D., Battat, J., Minkov, E., Hicks, J., Teller, S., and Ledlie, J. (2010). Growing an Organic Indoor Location System. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 271–284. ACM.

[81] Park, J.-g., Curtis, D., Teller, S., and Ledlie, J. (2011). Implications of device diversity for organic localization. In *2011 Proceedings IEEE INFOCOM*, pages 3182–3190. IEEE.

[82] Peng, Z., Richter, P., Leppäkoski, H., and Lohan, E. S. (2017). Analysis of

crowdsensed wifi fingerprints for indoor localization. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 268–277. IEEE.

[83] Piwońska, A. (2010). Genetic Algorithm Finds Routes in Travelling Salesman Problem with Profits. *Zeszyty Naukowe Politechniki Białostockiej. Informatyka*, **5**, 51–65.

[84] Popović, M., Vidal-Calleja, T., Hitz, G., Chung, J. J., Sa, I., Siegwart, R., and Nieto, J. (2020). An informative path planning framework for uav-based terrain monitoring. *Autonomous Robots*, pages 1–23.

[85] Potvin, J.-Y. (1996). Genetic Algorithms for the Traveling Salesman Problem. *Annals of Operations Research*, **63**(3), 337–370.

[86] Raghaven, T., Ferguson, T. S., Parthasarathy, T., and Vrieze, O. (2012). *Stochastic games and related topics: In honor of professor LS Shapley*, volume 7. Springer Science & Business Media.

[87] Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian process for machine learning*. MIT press.

[88] Restuccia, F., Ghosh, N., Bhattacharjee, S., Das, S. K., and Melodia, T. (2017). Quality of information in mobile crowdsensing: Survey and research challenges. *ACM Transactions on Sensor Networks (TOSN)*, **13**(4), 1–43.

[89] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

[90] Schwaighofer, A., Grigoras, M., Tresp, V., and Hoffmann, C. (2004). Gpps: A

gaussian process positioning system for cellular networks. In *Advances in Neural Information Processing Systems*, pages 579–586.

[91] Shibusawa, S. and Shibuya, T. (2016). Reinforcement learning in the environment where optimal action value function is partly discontinuous. In *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 1545–1550. IEEE.

[92] Shin, B., Lee, J. H., Lee, T., and Kim, H. S. (2012). Enhanced Weighted K-nearest Neighbor Algorithm for Indoor Wi-Fi Positioning Systems. In *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, volume 2, pages 574–577. IEEE.

[93] Singh, A., Krause, A., Guestrin, C., Kaiser, W. J., and Batalin, M. A. (2007). Efficient Planning of Informative Paths for Multiple Robots. In *IJCAI*, volume 7, pages 2204–2211.

[94] Singh, A., Krause, A., Guestrin, C., and Kaiser, W. J. (2009). Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, **34**, 707–755.

[95] Sivaraj, R. and Ravichandran, T. (2011). A Review of Selection Methods in Genetic Algorithm. *International journal of engineering science and technology*, **3**(5), 3792–3797.

[96] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[97] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[98] Tasgetiren, M. F. and Smith, A. E. (2000). A Genetic Algorithm for the Orienteering Problem. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 910–915. IEEE.

[99] Torres-Sospedra, J., Montoliu, R., Martínez-Usó, A., Avariento, J. P., Arnau, T. J., Benedito-Bordonau, M., and Huerta, J. (2014). Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In *2014 international conference on indoor positioning and indoor navigation (IPIN)*, pages 261–270. IEEE.

[100] Torteeka, P. and Chundi, X. (2014). Indoor Positioning based on Wi-Fi Fingerprint Technique using Fuzzy K-nearest Neighbor. In *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*, pages 461–465. IEEE.

[101] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[102] Van Otterlo, M. and Wiering, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer.

[103] Vansteenwegen, P., Souffriau, W., and Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, **209**(1), 1–10.

[104] Wang, R., Wang, W., Aung, M. S. H., Ben-Zeev, D., Brian, R., Campbell, A. T., Choudhury, T., Hauser, M., Kane, J., Scherer, E. A., and et al. (2017a). Predicting symptom trajectories of schizophrenia using mobile sensing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **1**(3).

[105] Wang, X. and Sandholm, T. (2002). Reinforcement learning to play an optimal nash equilibrium in team markov games. *Advances in neural information processing systems*, **15**, 1603–1610.

[106] Wang, Y.-T., Li, J., Zheng, R., and Zhao, D. (2017b). ARABIS: an Asynchronous Acoustic Indoor Positioning System for Mobile Devices. *arXiv preprint arXiv:1705.07511*.

[107] Wei, E. and Luke, S. (2016). Lenient learning in independent-learner stochastic cooperative games. *The Journal of Machine Learning Research*, **17**(1), 2914–2955.

[108] Wei, Y., Frincu, C., and Zheng, R. (2020). Informative path planning for location fingerprint collection. *IEEE Trans. Netw. Sci. Eng.*, **7**(3), 1633–1644.

[109] Wen, Y., Shi, J., Zhang, Q., Tian, X., Huang, Z., Yu, H., Cheng, Y., and Shen, X. (2014). Quality-driven auction-based incentive mechanism for mobile crowd sensing. *IEEE Transactions on Vehicular Technology*, **64**(9), 4203–4214.

[110] Wolpert, D. H. and Tumer, K. (2002). Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pages 355–369. World Scientific.

[111] Wu, C., Yang, Z., and Liu, Y. (2014). Smartphones based crowdsourcing for indoor localization. *IEEE Transactions on Mobile Computing*, **14**(2), 444–457.

[112] Wu, C., Xu, J., Yang, Z., Lane, N. D., and Yin, Z. (2017). Gain without pain: Accurate wifi-based localization using fingerprint spatial gradient. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **1**(2).

[113] Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., and Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. *arXiv preprint arXiv:1803.07246*.

[114] Xu, J., Chen, H., Qian, K., Dong, E., Sun, M., Wu, C., Zhang, L., and Yang, Z. (2019). Ivr: Integrated vision and radio localization with zero human effort. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **3**(3).

[115] Xu, Q. and Zheng, R. (2016). Mobibee: a mobile treasure hunt game for location-dependent fingerprint collection. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1472–1477.

[116] Zhang, J., He, T., Sra, S., and Jadbabaie, A. (2019). Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*.

[117] Zhang, X., Yang, Z., Sun, W., Liu, Y., Tang, S., Xing, K., and Mao, X. (2015). Incentives for mobile crowd sensing: A survey. *IEEE Communications Surveys & Tutorials*, **18**(1), 54–67.

[118] Zhong, W., Suo, Q., Ma, F., Hou, Y., Gupta, A., Qiao, C., and Su, L. (2019). A reliability-aware vehicular crowdsensing system for pothole profiling. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **3**(4).