A Reinforcement Learning Framework for Efficient Informative Sensing

Yongyong Wei, *Student Member, IEEE* and Rong Zheng, *Senior Member, IEEE*

Abstract—Large-scale spatial data can be collected using mobile robots with sensing and navigation capabilities. Due to limited battery lifetime and scarcity of charging stations, it is important to plan informative paths so as to maximize the utility of data given a limited travel budget, which is known as the informative path planning (IPP) problem. IPP is NP-hard, and existing solutions suffer from high complexity or low optimality. In this paper, we present a novel IPP solution based on reinforcement learning (RL). The basic idea is to learn the structural characteristics of informative paths, so informative paths can be predicted. As such, when budgets change, we avoid solving the problem from scratch and thus path planning efficiency can be improved dramatically. Among the 20 path planning experiments in two areas, the proposed RL based solution achieves the best path utility in 15 experiments, compared with state-of-the-art algorithms. More importantly, the inference complexity is linear with respect to the budget (equivalently, the maximum number of steps in RL), which is lower than other solutions. Despite the NP-hardness, the path planning process can be finished within a few seconds in our experiments on two graphs of different sizes.

Index Terms—Informative path planning, mobile sensing, reinforcement learning

1 Introduction

Large-scale spatial data (e.g., temperature, humidity, air quality, and location fingerprints) are essential for a wide range of applications. One common characteristic of these applications is that the data to be collected are location dependent. However, it is usually time-consuming and labor-intensive to collect those data manually. To reduce human efforts and automate the data collection process, one extensively investigated approach is to deploy wireless sensor networks (WSN) [1], which is widely used as a means of continuous environment monitoring. To exploit mobility, WSN with mobile elements [2] has also been considered. While individual sensing devices are typically at low costs, deploying and maintaining a large-scale WSN incur high capital and operational expenses.

For one-time or infrequent data collection, robotic technologies offer a viable and cost-effective alternative to fixed deployments [3]. A robot equipped with sensing devices can be controlled to traverse a target area and collect data along its path. Although robots can significantly reduce human efforts, they are battery powered and have limited lifetimes. Given a travel budget constraint (e.g., maximum travel distance or time), it is important to plan motion paths such that the state of the target area can be accurately estimated using the data collected.

One criteria for path planning is informativeness. In general, stationary spatial processes with different degrees of smoothness can be modeled as *Gaussian Processes* (GPs) [4] with appropriate kernel functions. Based on GPs, *mutual information* (MI) has been used to measure the informativeness of sensor placement [5]. In [6], [7], [8], MI is also used to measure the informativeness of a path for mobile sensing. The problem of finding the most informative path from a start location to a terminal location subject to a budget constraint is called *informative path planning* (IPP). Usually, for scenarios where IPP is beneficial, there are two salient characteristics. First, the informativeness is not uniformly distributed across the target field. This situation can arise when some sites have past observations. Second, the travel budgets of mobile agents are limited. Many real-world applications can benefit from IPP, such as precision agriculture [9], environmental monitoring [10] and location fingerprint collection [11].

IPP problems are generally formulated on graphs [7], [8], with vertices representing way-points and edges representing path segments. The utility[1] of a path can be associated with the vertices, edges or both from the path. In the special case where utility is limited to vertices and is additive, the IPP problem degenerates to the well-known Orienteering Problem (OP), which is known to be NP-hard [12]. Due to its complexity, existing solutions to IPP mostly adopt heuristics based search strategies such as greedy search [13] and evolutionary algorithms [11], [14]. These solutions often suffer from inferior performance. Furthermore, for the same environment, start and terminal locations, these solutions typically treat problem instances with different budgets as *independent*, and thus need to re-compute from scratch when the budget changes. Budget variations are not uncommon in practice due to different battery capacities, charging statuses or timing requirements.

In this work, we model IPP as a sequential decision problem. Given the start vertex on a graph, a path is constructed

• *The authors are with the Department of Computing and Software, McMaster University, Hamilton, ON L8S 4L8, Canada. E-mail: {weiy49, rzheng}@mcmaster.ca.*

Manuscript received 27 Apr. 2020; revised 22 Nov. 2020; accepted 24 Nov. 2020. Date of publication 26 Nov. 2020; date of current version 3 June 2022. (Corresponding author: Yongyong Wei.)
Digital Object Identifier no. 10.1109/TMC.2020.3040945

1. In this work, we use the terms utility and reward exchangeably.

sequentially by appending the next way-point vertex. We use reinforcement learning (RL) to *learn* the structural characteristics of informative paths and maximize the total future reward, which is equivalent to the utility of a path. In the context of IPP, the path utility is mainly determined by the landscape of the environment and locations of existing observations. Using the learned RL model, informative paths can be *predicted* given any input budget. The benefit is two-fold. First, we avoid learning different models for different budgets. Second, by considering problem instances with different budgets jointly, learning can be more efficient due to shared structures.

Compared with conventional RL tasks, IPP poses a few non-trivial challenges. IPP itself can be seen as a combinatorial optimization problem that consists of two NP-hard subproblems, i.e., which subset of vertices to visit and in which order to visit them. In IPP, the reward of an action depends on past actions. For instance, re-visiting a vertex can lead to less but non-zero reward. In addition, eligible paths are constrained by budgets and also terminal locations. As a result, RL needs to be tailored carefully to the problem setting. We propose a general reinforcement framework for IPP based on recurrent neural network (RNN). A novel action selection method is designed to accommodate path specifications and improve learning efficiency.

To evaluate the proposed approach, we consider the task of WiFi Received Signal Strength (RSS) collection in indoor environments. WiFi RSS measurements are widely used in fingerprint-based indoor localization solutions [15], [16], [17]. In [11], it is shown that data collected along informative paths tend to achieve better localization performance. In our experiments, real RSS data have been collected from two areas to estimate the hyperparameters of GPs and set up the environment of RL. In total, 20 different path specifications (different start/terminal vertices, or budget constraints) have been evaluated. Among them, the RL based IPP algorithm outperforms state-of-the-art methods in 15 configurations with higher informativeness. More importantly, our RL based path planning method is much more efficient than other solutions, and have a comparable run time as the simple greedy algorithm.

The rest of this paper is organized as follows. In Section 2, related work to IPP and a background of RL are introduced briefly. The IPP problem is formulated in Section 3. We present the proposed solution in Section 4. Experimental results are shown in Section 5. Finally, we discuss and conclude our work in Section 6 and 7, respectively.

2 Related Work and Background

In this section, we review some related studies on IPP and give a brief introduction to RL.

2.1 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 concept of informativeness comes from MI in information theory. Originally, MI is adopted as a criteria of location optimization for sensor placement [5] when static sensors are deployed for spatial monitoring. When mobile agents like robots are available to travel across the target area for data collection, the criteria extends to path planning naturally, known as informative path planning [6].

One major challenge of IPP is that the problem is a combinatorial optimization problem with NP-hard [11] complexity. Thus, it is difficult to achieve optimality and efficiency at the same time. A few conventional solutions to IPP rely on the Recursive Greedy (RG) algorithm [18]. 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 with RG can be found in [6], [7]. Specifically, [7] also considers rewards from edges besides vertices. To reduce the high computation complexity of RG, [6] proposes spatial decomposition to create a coarse graph by grouping vertices into cells. The algorithm is then applied on the cell-based coarse graph. Unfortunately, doing so could compromise optimality compared with on the original graph. In [19], the authors discuss the scenarios whether paths have start or terminal restrictions and adopt greedy algorithms to schedule paths since all the three scenarios are NP-hard.

Evolutionary strategies have been investigated for IPP. In [11], a Genetic Algorithm (GA) based solution is presented, and experiments show that GA achieves a good trade-off between computation complexity and optimality. The authors in [20] model the path planning process as a control policy and a heuristic strategy is proposed by incrementally constructing the policy tree. In [21], [22], 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, they also consider re-planning and adaptive sampling so as to focus on regions of interests. The basic idea is to omit locations where the predicted measurements (based on GP) are below a certain threshold. When the objective of path planning is to detect abnormal or extreme observations, Bayesian Optimization is also leveraged [23]. In this work, we focus on the path planning stage, and informativeness does not depend on specific measurement values but only their locations. This is suitable for applications where the goals are to explore the environments and map the scalar field, like to survey the RSS signatures in indoor spaces for localization purpose or to map the temperature of a target field. In addition, both [21], [22] do not consider obstacles and assume the robots can travel anywhere in the continuous space. By contrast, we follow the graph based formulation [7], [8], which is advantageous especially when the sensing area has road-maps or obstacles since reachability could be constrained directly with edges.

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 subset (vertices) selection and path construction. Once the set of vertices are determined, a travelling salesman problem (TSP) solver can be utilized to construct a path with the minimum cost. In [24], a randomized algorithm is presented, where vertices are randomly added or removed repetitively to track the best subset. Similarly, in [25], way-points are added incrementally and the TSP solver is utilized to generate paths. However, in practical IPP scenarios, the aforementioned assumptions are not

necessary, since i) vertices could be visited multiple times, particularly when the graph is not complete; ii) rewards can be obtained while the agent is moving along the edges because sensors are recording the spatial data.

Due to the NP-hardness, most 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 location, the path planning process needs to be repeated from scratch, which further degrades efficiency. In our previous work [26], we found that transfer learning can mitigate this situation to some extent. In this work, we aim to train models that can make predictions when budgets change. Thus, path planning efficiency can be greatly improved.

2.2 Reinforcement Learning

RL [27] is a type of learning approach in machine learning. It is mainly used to solve sequential decision problems, where an agent interacts with the environment. In each step, the agent takes an action and receives a reward signal. The ultimate goal is to learn a decision policy such that the total reward received by the agent following the policy is maximized.

There are two main types of approaches towards RL, namely value-based and policy-based approaches. Representative algorithms include Q-learning and policy gradient, respectively. Specifically, Q-learning aims to learn a function mapping between state-actions and q-values. Once the function relation is learnt, the corresponding policy can be induced. On the other hand, policy gradient formalizes and learns the policy directly through gradient ascent. Researchers have also proposed to combine the value-based approach and the policy-based approach, which is the actor-critic [28] methods. A critic (value-based) is trained to evaluate the actions selected by the actor (policy-based). In many cases, actor-critic methods show better convergence property compared with policy gradient methods such as the reinforce algorithm.

In recent years, with the advancement of deep neural networks [29], deep reinforcement learning [30] 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 [31], [32] attempt to adopt RL in combinatorial optimization. In [31], 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 [32], 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.

Both [31], [32] assume that the graphs are complete. However, in the real world, presence of obstacles in spatial areas implies that the corresponding graphs have limited connectivity. In this work, our proposed solution allows the graph to be an arbitrary graph. In addition, to solve the TSP, it only needs to optimize the order of vertices. While in IPP, besides the order, it also needs to consider which subset of vertices to select together with other specifications such as budget, start and terminal vertices. These conditions make the problem even more challenging and solutions in [31], [32] are not feasible for IPP. To the best of our knowledge, this is the first time RL is investigated to solve the IPP problem.

3 Problem Formulation

The IPP problem operates on a graph. In practice, the graph can be created with points of interests as vertices, and an edge exists if two vertices are reachable. Another way is to create a grid graph if the target area is open. Next, we first define the general path planning problem and then extend it to IPP.

3.1 General Path Planning With Limited Budget

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

- $G = (\mathcal{V}, \mathcal{E})$ is the 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 is 2D shaped, thus $\mathbf{x}$ is a 2D coordinate. For each $e \in \mathcal{E}$, there is a corresponding 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 possibly be the depot of the robot or a charging station.
- $\mathcal{P} = [v_s, \ldots, v_k, \ldots, v_t]$ denotes a valid path[2], and $f(\mathcal{P})$ represents the corresponding utility.
- B represents the budget available for the path. We emphasize two scenarios: i) B is known and fixed; ii) B can be variable, this can be the case when the robot is not fully charged. Almost all the existing path planning algorithms assume the budget is fixed. When the budget changes, the natural solution is to re-run the algorithm with the updated budget, which is quite inefficient.

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

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

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

$$\mathcal{P}^* = \arg\max_{\mathcal{P}\in\Psi} f(\mathcal{P}) \text{ s.t. } c(\mathcal{P}) \leq B, \tag{2}$$

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

One classic realization of the general path planning formulation is the OP [12], [33], [34]. 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, which is simply defined as the sum of rewards from individual vertices.

2. 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.

3.2 Informative Path Planning

IPP is another realization of the general path planning problem by specifying the reward function with the informativeness of the data collected along the path. Next, we present the details of $f(\mathcal{P})$ for IPP based on GPs and MI. We recommend [4] for more background on GPs.

Assume 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 a Gaussian distributed random variable, and all the variables $\mathbf{y}_\mathcal{V}$ at locations of $\mathcal{V}$ follow a joint multivariate Gaussian distribution,

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

where $m(\mathbf{x})$ is the mean function, $k(\cdot,\cdot)$ is the kernel, and $n = |\mathcal{V}|$ is the total number of vertices. 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 $n \times 2$ matrix of the locations of $\mathcal{V}$, and $\Sigma_\mathcal{V}$ is the $n \times n$ covariance matrix generated through the kernel.

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

$$H(\mathbf{y}_\mathcal{V}) = \frac{1}{2}\ln|\Sigma_\mathcal{V}| + \frac{n}{2}(1+\ln(2\pi)). \tag{3}$$

Given $\mathcal{P} = [v_s, \dots, v_k, \dots, v_t]$, suppose data will be collected by an agent along the path every d meter interval (depends on travel speed and sample frequency). The sample locations can be easily calculated using the physical positions of the vertices. We denote all the sample locations as $X_\mathcal{S}$ and the corresponding measurements as $\mathbf{y}_\mathcal{S}$. The posterior distribution of $\mathbf{y}_\mathcal{V}$ given $\mathbf{y}_\mathcal{S}$ is $\mathcal{N}(\boldsymbol{\mu}', \Sigma')$, where

$$\boldsymbol{\mu}' = m(X_\mathcal{V}) + K(X_\mathcal{V}, X_\mathcal{S})(K(X_\mathcal{S}, X_\mathcal{S}) + \sigma_n^2 I)^{-1} \mathbf{y}_\mathcal{S} - m(X_\mathcal{S})), \tag{4}$$

$$\Sigma' = K(X_\mathcal{V}, X_\mathcal{V}) - K(X_\mathcal{V}, X_\mathcal{S})(K(X_\mathcal{S}, X_\mathcal{S}) + \sigma_n^2 I)^{-1} K(X_\mathcal{S}, X_\mathcal{V}). \tag{5}$$

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

$$H(\mathbf{y}_\mathcal{V}|\mathbf{y}_\mathcal{S}) = \frac{1}{2}\ln|\Sigma'| + \frac{n}{2}(1+\ln(2\pi)). \tag{6}$$

The MI based reward can then be calculated with

$$f(\mathcal{P}) = \mathrm{MI}(\mathbf{y}_\mathcal{V}; \mathbf{y}_\mathcal{S}) = H(\mathbf{y}_\mathcal{V}) - H(\mathbf{y}_\mathcal{V}|\mathbf{y}_\mathcal{S}). \tag{7}$$

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

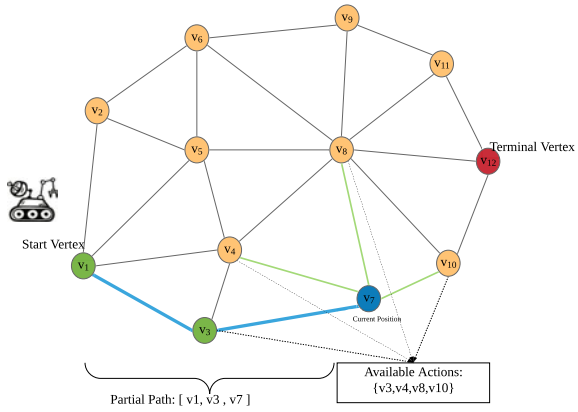


Fig. 1. Sequential decision process for IPP.

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 [1], the hyperparameters can be learned using measurements from the existing static sensors. On the other hand, in cases where no prior studies are available like in [5], [7], [18], a pilot data collection round 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

$$f_\mathcal{D}(\mathcal{P}) = \mathrm{MI}(\mathbf{y}_\mathcal{V}; \mathbf{y}_S \cup \mathbf{y}_\mathcal{D}) = H(\mathbf{y}_\mathcal{V}) - H(\mathbf{y}_\mathcal{V}|\mathbf{y}_\mathcal{S} \cup \mathbf{y}_\mathcal{D}). \tag{8}$$

In addition, although in our problem formulation we assume the environment is 2D, it is straightforward to extend 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 using GPs in a 3D environment can be found in [21]. 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, since the problem is NP-hard, brute force search is not computationally feasible in practice.

4 IPP Solution With RL

In this section, we present an IPP solution framework based on RL. RL concepts in the IPP context will be defined first, followed by a solution overview and details.

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. 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 vertex to travel, and the available vertices 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*.

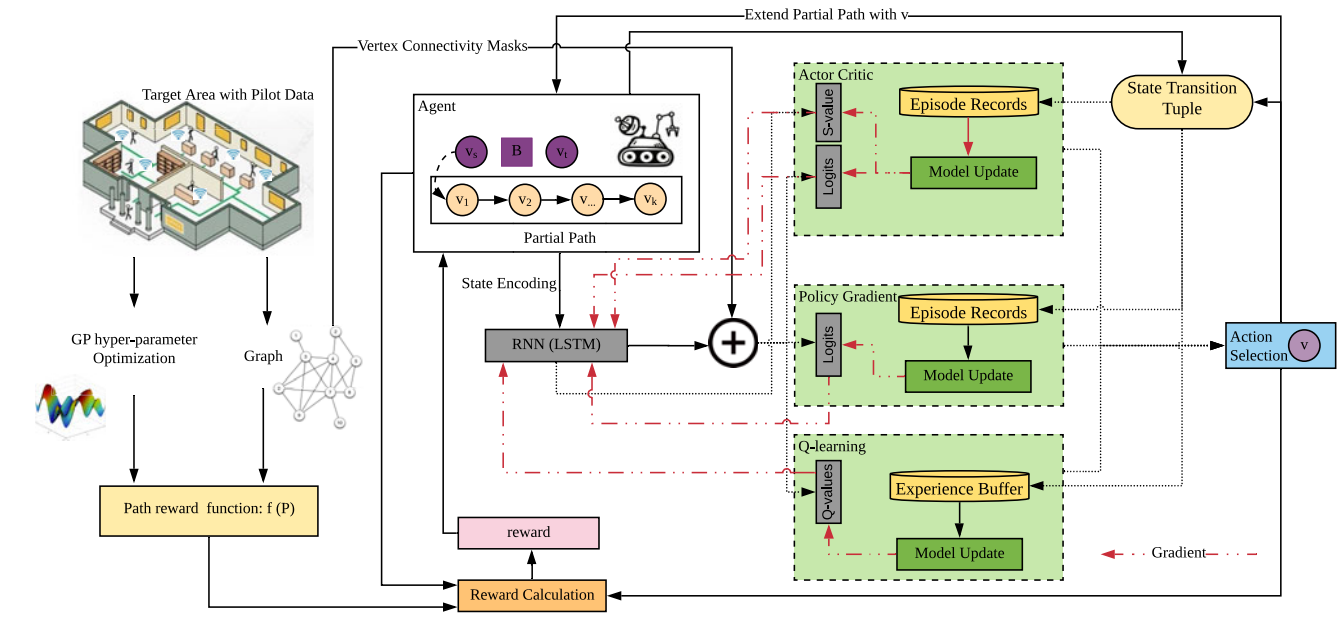


Fig. 2. Solution overview with reinforcement learning.

Formally, the decision process can be described by a Markov Decision Process (MDP) [35] $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$, 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[3] defined as $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The main effect of state transition here is that the partial path extends by one vertex.
- $\mathcal{R}$ is the reward function defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, 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 the decision making policy as $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$. 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

$$R_t = r_t + \gamma r_{t+1} + \ldots + \gamma^{T-t} r_T, \tag{9}$$

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

4.1 Solution Overview

Fig. 2 shows the overall architecture of the solution. For a typical path planning scenario, the input is the target area with a small amount of pilot data (or data collected from existing static sensor nodes). The area is discretized and a graph is created. As mentioned previously, the data to be collected are spatially correlated and assumed to be generated from an underlying GP. With the pilot data, a GP regression model is fitted and optimized to estimate the hyperparameters. Once the hyperparameters are estimated, the reward function $f_{\mathcal{D}}(\mathcal{P})$ defined in (8) is determined, which is used to evaluate the informativeness of paths and calculate rewards.

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 an 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

$$\mathbf{m}_1[i] = \begin{cases} 0 & v_i \in \text{adj}(v_k) \\ -\infty & \text{else} \end{cases}, \tag{10}$$

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 State Encoding

Fig. 3 illustrates the state encoding scheme. Given the state as $\langle \bar{\mathcal{P}}_k = [v_1, v_2, \ldots, v_k], v_s, v_t, B \rangle$, it is encoded as a $5 \times k$ matrix that can be fed into the 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

3. In this work we consider deterministic transitions.

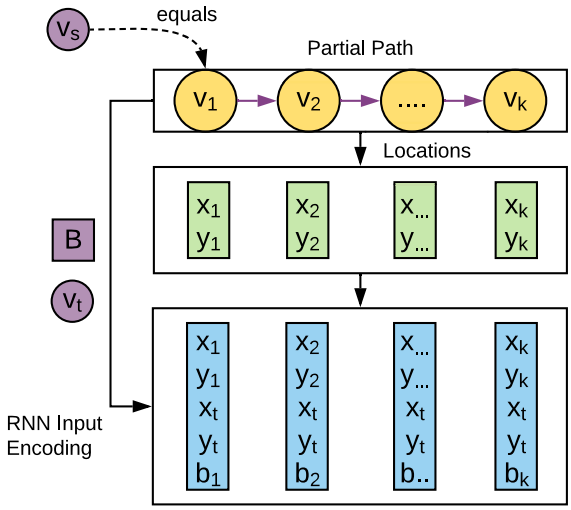


Fig. 3. Input encoding for the states.

$$\mathrm{b}_k = B - \sum_{i=2}^{k} c(v_{i-1}, v_i) \tag{11}$$

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

4.3 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 the reward signals. However, this simple solution will lead to a large number of invalid paths that do not terminate at v_t and thus waste computation resources. Next, we present our *constrained action selection* strategy that effectively reduces the search space.

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

$$\mathcal{A}(\bar{\mathcal{P}}_k) = \mathrm{adj}(v_k) = \{v \in \mathcal{V} : (v_k, v) \in \mathcal{E}\}. \tag{12}$$

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

$$\mathcal{A}'(\bar{\mathcal{P}}_k) = \{v \in \mathcal{A}(\bar{\mathcal{P}}_k) : c(v_k, v) + \mathrm{LCP}(v, v_t) \leq \mathrm{b}_k\}, \tag{13}$$

where $\mathrm{LCP}(v, v_t)$ denotes 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

$$a(\bar{\mathcal{P}}_k) = \begin{cases} \text{random } v \in \mathcal{A}'(\bar{\mathcal{P}}_k) & \text{with probability } \epsilon \\ \arg\max_{v \in \mathcal{A}'(\bar{\mathcal{P}}_k)} \mathbf{Q}_{\mathcal{V}}[v] & \text{with probability } 1-\epsilon \end{cases}. \tag{14}$$

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. Then we define another mask vector as

$$\mathbf{m}_2[i] = \begin{cases} 0 & v_i \in \mathcal{A}'(\bar{\mathcal{P}}_k) \\ -\infty & \text{else} \end{cases}. \tag{15}$$

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

$$\pi(a \,|\, \bar{\mathcal{P}}_k) = \mathrm{softmax}(\mathbf{logits} + \mathbf{m}_2), \tag{16}$$

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

4.4 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

$$r(\bar{\mathcal{P}}_k, a) = f([\bar{\mathcal{P}}_k, a]) - f(\bar{\mathcal{P}}_k). \tag{17}$$

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

$$r(\mathcal{P}) = f(\mathcal{P}) = \sum_{k=1}^{|\mathcal{P}|-1} r(\bar{\mathcal{P}}_k, a). \tag{18}$$

This is exactly the optimization goal as discussed in Section 3.2, 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.

4.5 Reinforcement Learning Methods

The proposed architecture supports a variety of RL methods such as Q-learning [36], policy gradient [37] and actor-critic [38], with minor adaptations. RL itself is not considered as a contribution in this work. Readers interested in RL can consult [27] for more information. For completeness, we provide the basic steps of training typical RL models for IPP.

4.5.1 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 the 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

$$\ell(\theta) = \Big(\mathcal{Q}_\theta(s_t, a_t) - (r_t + \gamma \max_a \mathcal{Q}_\theta(s_{t+1}, a))\Big)^2. \tag{19}$$

To stabilise the training process, advanced techniques have been proposed in recent years like DDQN [39], Prioritized Experience Replay [40], which can be adopted directly in the solution.

4.5.2 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 as $\pi_\theta(a \mid s)$, and we denote one episode of length T as $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$. The expected total reward is

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{T} r_t\right], \tag{20}$$

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

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\left(\sum_{t=1}^{T} r_t\right)\right]. \tag{21}$$

The reinforce algorithm is a policy gradient approach that uses (21) 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 4.3.

4.5.3 Actor-Critic

One disadvantage of policy gradient is that the policy estimator suffers from a high variance [41], and it has to wait until the end of an episode to update the model. Actor-critic methods address this issue by approximating the gradient with

$$\nabla_\theta J(\theta) \approx \sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t), \tag{22}$$

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) [42],

$$A(s_t, a_t) = r_t + V_\phi(s_{t+1}) - V_\phi(s_t), \tag{23}$$

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. 2. Both networks share the RNN component for better efficiency.

4.6 Model Training and Path Inference

In our previous work [26], we considered the case when the budget B is given and fixed. As such, the agent learns to construct informative paths gradually, and there is no separate inference procedure required. The best path obtained during the training stage is the final output path. However, when the budget changes, like existing IPP solutions, the learning process needs to be repeated, which can be time-consuming. We proposed transfer learning to mitigate this issue to some extent. In this work, we address this issue by training models that can be used to predict informative paths with variable budgets. Thus, the efficiency can be improved greatly since only the inference process is required.

4.6.1 Model Training

Although the exact budget B is unknown in advance, we assume it falls in a range depending on the deployed 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 1. For each episode, a specific budget B is randomly sampled from $\mathcal{B}$ and used for training. Meanwhile, for every E episode, 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 1 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.

Algorithm 1. 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* $e = 1, 2, \ldots$ **do**
  randomly sample $B \in \mathcal{B}$;
  initialize $\bar{\mathcal{P}} = [v_s]$;
  episode records = {} ;
  for *step* $t = 1, 2, \ldots, T$ **do**
    selection action a_t with (14) or (16);
    execute a_t and calculate reward r_t with (17);
    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*/;
  calculate the loss of the specific RL model M ;
  update M with gradient descent;
  if $e \bmod E = 0$ **then**
    take a snapshot of M and add it into $\mathcal{M}$;
end
return $\mathcal{M}$

4.6.2 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 policies. Thus, we store a set of 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 2. With such a mechanism, the solution can accommodate variable budgets. For instance, when robots are not fully charged, their battery performance degrade, new robots introduced, or there are task deadlines.

Algorithm 2. Path Inference Procedure

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

```
initialize a path set P_M = {} ;
for M ∈ M do
    create an environment with v_s, v_t, B ;
    initialize P̄ = [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 P̄ to P_M;
        else
            P̄ = [P̄, a_t];
    end
end
return P* = arg max_{P∈P_M} f(P)
```

5 Experimental Results

In this section, as a practical IPP application, we consider the collection of Wi-Fi RSS, which has been extensively investigated for fingerprint-based indoor localization [15], [16], [17]. We first evaluate the impacts of constrained action selection in learning efficiency and also inspect the convergence property. 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 and path planning efficiency.

5.1 Graph Setting and Implementation

Two real-world indoor areas with different shapes are selected and discretized into grid graphs. The first area is an open area and the second area is a corridor. To estimate the hyperparamers of the underlying GP, a small amount of pilot WiFi signals are collected. The two areas are illustrated in Figs. 4 and 5, respectively. Note that the vertices are indexed with numbers, and later we will refer to the vertices using the numbers.

The environment is implemented in Python, and it tracks all the necessary information for state encoding and reward calculation, such as partial paths, remaining budgets and graph structures. To accelerate computation, shortest paths between all pairs of vertices are calculated and stored in advance, since the agent needs access to shortest paths at each step to filter out invalid actions. The APIs of the environment are similar to those in the OpenAI Gym[4], a reinforcement learning toolkit with a variety of environments.

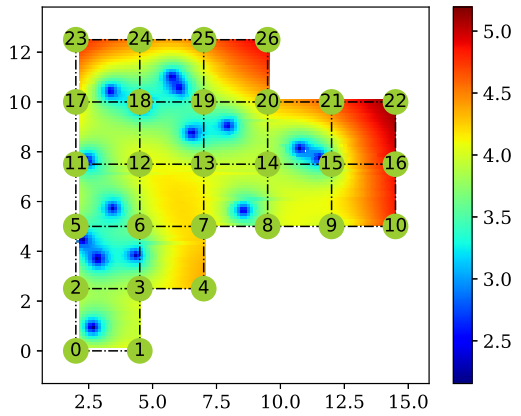


Fig. 4. Graph generated from Area One. The size of the whole area is approximately 12m $\times$ 13m. The *X* and *Y* axes show the dimensions in meters, and the color represents the uncertainty (entropy) of the predicted signals by fitting a GP with the pilot data. The grid graph has 26 vertices and are indexed with integers.

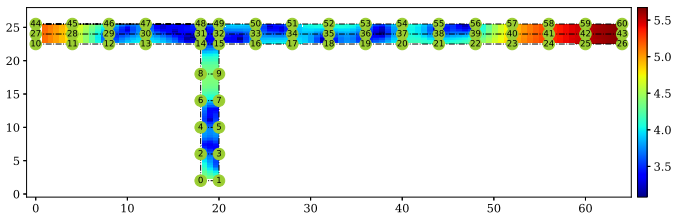


Fig. 5. Graph generated from Area Two. This area is a "T" shape corridor, with 25m in height and 64m in length. The graph has 61 vertices as shown by the green circles.

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 technique is used to avoid gradient explosion. Reward discount factor γ is set to 0.9, and a snapshot of the model is taken for every 1000 episode (E in Algorithm 1).

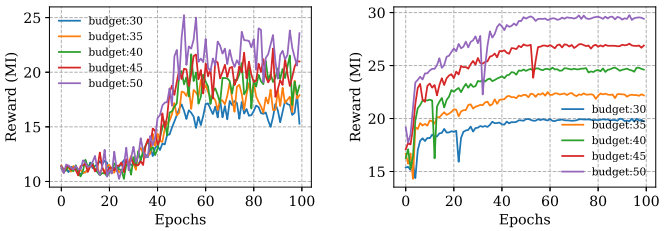
5.2 Comparison With Unconstrained Action Selection

In addition to the proposed constrained action selection, an alternative simple action selection scheme is implemented that considers all adjacent vertices as potential next waypoints as mentioned in Section 4.3.

We trained a double Q-learning [39] model with prioritized experience replay [40] using the two action selection schemes. In the experiment, the budget is fixed at a specific value. Fig. 6 and Fig. 7 show the average episode reward with the learning process in Area One and Two, respectively.

Similar to [36], 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,

4. https://gym.openai.com



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

Fig. 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.

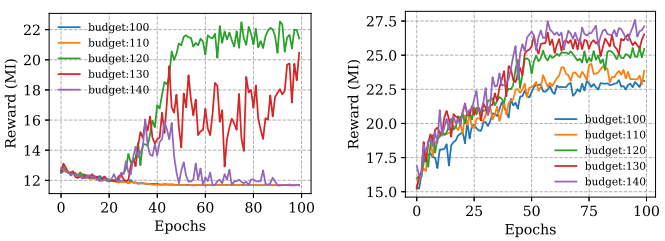
the rewards are low since most generated paths are invalid, i.e, failing to terminate at v_t. Thus, the agent will receive a penalty reward signal at the last step. The difference is more significant in Area Two since the graph size is larger than that of Area One. In a larger graph, blind searches have a smaller probability to construct a valid path. As can be seen from Fig. 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.

5.3 Convergence Using Different RL Methods

Next we inspect the training convergence property with 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 1. 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.

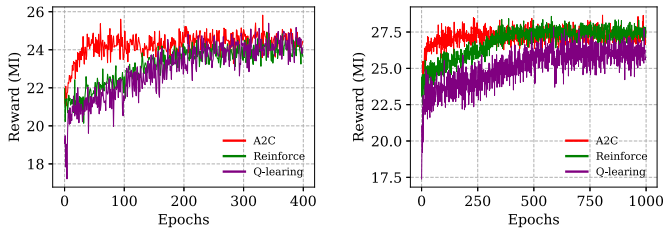
In addition, in later sections, we will consider two types of paths, namely

- *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., $v_s \neq v_t$.



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

Fig. 7. Average reward per episode with Q-learning in the graph from Area Two. The parameter settings are similar with Fig. 6.



(a) Area One (b) Area Two

Fig. 8. Average reward per episode during training with different RL methods.

For illustration purposes, for each episode, v_s in Area One is set to vertex 0, and v_t is randomly sampled 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 randomly sampled from vertices [0, 60].

Fig. 8 shows the average reward per episode for different RL methods. For Q-learning, note that episodes are generated through the ϵ-greed policy (14) with ϵ starts at 0.9 and decays to 0.1 gradually; while for the other two methods, episodes are generated from the policy network directly (16). 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.

5.4 Path Inference Performance

Next we show the performance when the models are utilized for path inference with Algorithm 2. We first show the performance of each snapshot and then the impact of training budget selection.

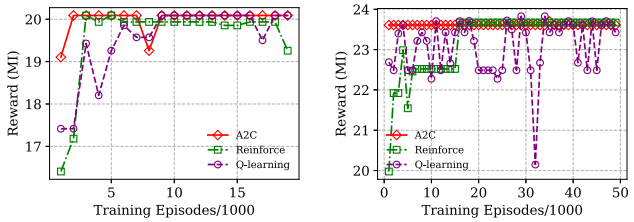
5.4.1 *Impact of Snapshots*

Snapshots captured during training are used to infer paths in the inference stage. Fig. 9 shows the inference 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. Although the models converge in later episodes, one single snapshot may still lead to a low inference reward. 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 of Q-learning or the reinforce algorithm.

5.4.2 *Impact of Training Budgets*

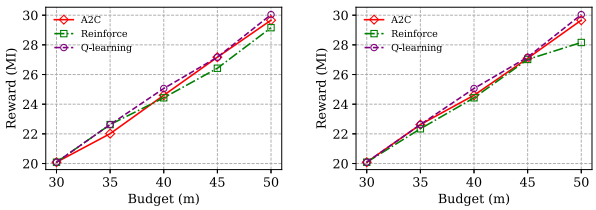
Recall that in Section 5.3 we trained the model with a set of budget $\mathcal{B}$ that is uniformly sampled from a range since the exact budget B is not known. Intuitively, the selection of $\mathcal{B}$ could affect the inference performance given a specific budget B.

We selected 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. Figs. 10 and 11 show the rewards obtained in Area One and Two, respectively. It can be seen that Q-learning and



(a) Area One (b) Area Two

Fig. 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 $v_s = 0, v_t = 0, B = 100$.



(a) (b)

Fig. 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].

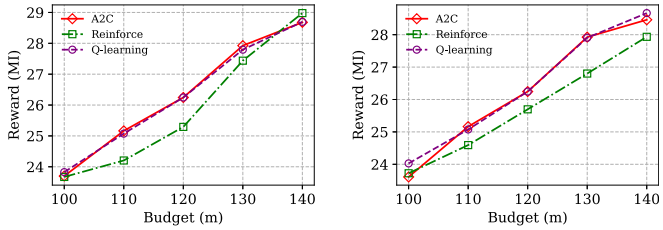
A2C have a similar performance and are less sensitive to the training budgets, while the reinforce algorithm is more sensitive, particularly in Area Two. This could be attributed to that the reinforce algorithm has a high variance in gradients.

5.5 Comparison With Other IPP Solutions

In this section, we compare the rewards achieved through reinforcement learning with those from some other IPP algorithms. As shown from 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 given a specification. Specifically, as can be seen from Fig. 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 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.

The following algorithms have been implemented for comparison:

Brute Force Tree Search. The brute force tree search tries to enumerate all the paths from v_s to v_t under the budget constraint and tracks the path with the highest reward. A stack is maintained to store the partial paths and branches are searched similar to the depth-first-search traverse. Here v_s can be seen as the root of the search tree. To improve efficiency, a search branch is pruned when v_t is encountered or the budget is exhausted. The advantage of this approach is that it can provide the optimal path. However, due to the complexity, this method can only be applied on the graph of Area One with a small budget, given a maximum of 72 hours.



(a) (b)

Fig. 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].

Recursive Greedy Algorithm. The Recursive Greedy (RG) algorithm is adapted from [18]. Originally, RG is designed for the orienteering problem. The basic idea is introduced in Section 2. The overall algorithm logic remains the same, but the reward function is defined as the informativeness of paths as introduced in Section 3.2.

Greedy Algorithm. The greedy algorithm is implemented following [11]. Vertices are selected greedily based on the marginal reward-cost ratio, and a Stainer TSP solver is implemented based on [43] to generate paths since the graph is not complete.

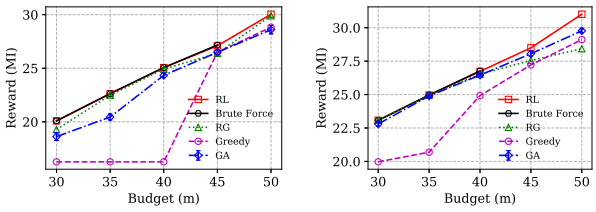
Genetic Algorithm. The Genetic Algorithm is also implemented according to [11]. Each valid path represents a chromosome, and a set of individuals (paths) are initialized. For each generation, a crossover and a mutation process are implemented. After a number of generations, the path with the maximum MI is considered as the final solution. 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 2) is executed using the snapshots captured during training (Section 4.6.1).

It can be seen from Fig. 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.

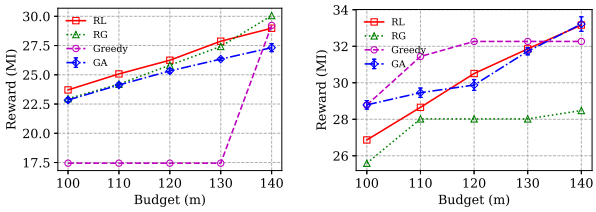
The graph from Area Two contains 61 vertices, with budgets larger than that in Area One, which leads to exponential increase in terms of search space. Fig. 13 shows the results from RL, RG, GA and the greedy approach. In the tour case, RL outperforms the other algorithms in four out of the five budget settings. However, in the non-tour case in Fig. 13b, 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,



(a) tour (b) non-tour

Fig. 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.



(a) tour (b) non-tour

Fig. 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.

another important aspect is computation efficiency, which is demonstrated next.

5.6 Path Planning Efficiency

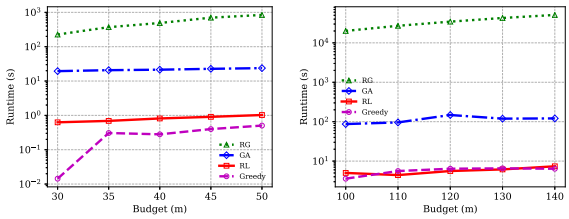
RG suffers from a high computation complexity with $O((2nB)^I \cdot T_f)$ [18], where n is the number of vertices and T_f is the maximum time to evaluate the reward function on a given set of vertices, and I is the recursion depth. In our experiments, I is set to two in both cases. A larger recursion 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 the 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 (4 GHz Intel Core i7, 16 GB RAM, *without* GPU) is shown in Fig. 14. 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 (finish 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 $|\mathcal{M}|$.

6 Discussion

We observed that in the smaller graph, 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.



(a) Area One (b) Area Two

Fig. 14. Approximate run time of different algorithms for the graph of area one and two on iMac (4GHz, Intel Core i7).

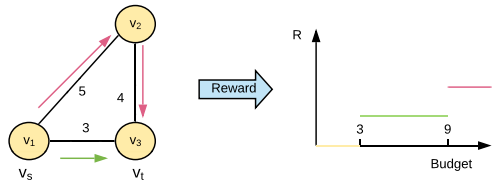


Fig. 15. The actual Q-value is discontinuous with respect to budget. In this simple example, the graph makes a triangle. When $v_s = v_1$ and $v_t = v_3$, the minimal budget required from v_s to v_t is is 3. The best path $(v_1 \rightarrow v_3)$ and corresponding reward will not change until budget increases to 9 $(v_1 \rightarrow v_2 \rightarrow v_3)$.

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.

Second, 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.

Third, the underlying true path reward function is actually a discontinuous function [44] 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. 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.

7 Conclusion

In this paper, we presented a RL framework for informative path planning. To address the unique challenges posed by path constraints, a novel action selection module is designed with the assistance of shortest paths. Compared with the unconstrained action selection strategy, it has a better efficiency and optimality. Under the framework, we implemented Q-learning, actor-critic and the reinforce algorithm, and compared with other state-of-the-art algorithms. The RL based solution achieves better path utility in most cases, and is much more efficient since it only needs to run the inference process using the neural network. Our future research direction is to investigate the IPP problem for multiple cooperative robots.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, 2002.

[2] M. D. Francesco, S. K. Das, and G. Anastasi, "Data collection in wireless sensor networks with mobile elements: A survey," *ACM Trans. Sensor Netw.*, vol. 8, no. 1, 2011, Art. no. 7.

[3] Y. Wang and C.-H. Wu, "Robot-assisted sensor network deployment and data collection," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom.*, 2007, pp. 467–472.

[4] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, vol. 2, no. 3, 2006.

[5] C. Guestrin, A. Krause, and A. P. Singh, "Near-optimal sensor placements in gaussian processes," in *Proc. 22nd ACM Int. Conf. Mach. Learn.*, 2005, pp. 265–272.

[6] A. Singh, A. Krause, C. Guestrin, W. J. Kaiser, and M. A. Batalin, "Efficient planning of informative paths for multiple robots," in *Proc. Int. Joint Conf. Artif. Intell.*, 2007, vol. 7, pp. 2204–2211.

[7] J. Binney, A. Krause, and G. S. Sukhatme, "Informative path planning for an autonomous underwater vehicle," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 4791–4796.

[8] A. Meliou, A. Krause, C. Guestrin, and J. M. Hellerstein, "Nonmyopic informative path planning in spatio-temporal models," in *Proc. AAAI*, vol. 10, no. 4, 2007, pp. 16–7.

[9] P. Tokekar, J. V. Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1498–1511, Dec. 2016.

[10] L. Bottarelli, M. Bicego, J. Blum, and A. Farinelli, "Orienteering-based informative path planning for environmental monitoring," *Eng. Appl. Artif. Intell.*, vol. 77, pp. 46–58, 2019.

[11] Y. Wei, C. Frincu, and R. Zheng, "Informative path planning for location fingerprint collection," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 3, pp. 1633–1644, Jul.–Sep. 2020.

[12] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *Eur. J. Oper. Res.*, vol. 209, no. 1, pp. 1–10, 2011.

[13] S. Yu, J. Hao, B. Zhang, and C. Li, "Informative mobility scheduling for mobile data collector in wireless sensor networks," in *Proc. IEEE Global Commun. Conf.*, 2014, pp. 5002–5007.

[14] A. Viseras, R. O. Losada, and L. Merino, "Planning with ants: Efficient path planning with rapidly exploring random trees and ant colony optimization," *Int. J. Adv. Robot. Syst.*, vol. 13, no. 5, pp. 1–16, 2016. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/1729881416664078

[15] C. Wu, Z. Yang, Y. Liu, and W. Xi, "WILL: Wireless indoor localization without site survey," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 4, pp. 839–848, Apr. 2013.

[16] Z. Yang, C. Wu, and Y. Liu, "Locating in fingerprint space: Wireless indoor localization with little human intervention," in *Proc. 18th ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 269–280.

[17] C. Li, Q. Xu, Z. Gong, and R. Zheng, "TuRF: Fast data collection for fingerprint-based indoor localization," in *Proc. IEEE Int. Conf. Indoor Positioning Indoor Navigation*, 2017, pp. 1–8.

[18] C. Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs," in *Proc. 46th Annu. IEEE Symp. Foundations Comput. Sci.*, 2005, pp. 245–253.

[19] S. Yang, K. Han, F. Wu, and G. Chen, "On designing quality-aware steering algorithms for large-scale mobile crowdsensing," in *Proc. 15th Annu. IEEE Int. Conf. Sens., Commun. Netw.*, 2018, pp. 1–9.

[20] R. A. MacDonald and S. L. Smith, "Active sensing for motion planning in uncertain environments via mutual information policies," *Int. J. Robot. Res.*, vol. 38, no. 2–3, pp. 146–161, 2019.

[21] G. Hitz, E. Galceran, M.-È. Garneau, F. Pomerleau, and R. Siegwart, "Adaptive continuous-space informative path planning for online environmental monitoring," *J. Field Robot.*, vol. 34, no. 8, pp. 1427–1449, 2017.

[22] M. Popović *et al.*, "An informative path planning framework for UAV-based terrain monitoring," *Auton. Robots*, vol. 44, pp. 889–911, 2020.

[23] R. Marchant and F. Ramos, "Bayesian optimisation for intelligent environmental monitoring," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 2242–2249.

[24] S. Arora and S. Scherer, "Randomized algorithm for informative path planning with budget constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4997–5004.

[25] K.-C. Ma, L. Liu, and G. S. Sukhatme, "Informative planning and online learning with sparse gaussian processes," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4292–4298.

[26] Y. Wei and R. Zheng, "Informative path planning for mobile sensing with reinforcement learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2020, pp. 864–873.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT press, 2018.

[28] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[30] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*. [Online]. Available: http://arxiv.org/abs/1701.07274

[31] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Bk9mxlSFx

[32] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6348–6358.

[33] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Res. Logistics*, vol. 34, no. 3, pp. 307–318, 1987.

[34] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *Eur. J. Oper. Res.*, vol. 255, no. 2, pp. 315–332, 2016.

[35] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement Learning*. Berlin, Germany: Springer, 2012, pp. 3–42.

[36] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[37] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[38] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Trans. Syst., Man, Cybern., C, Appl. Rev.*, vol. 42, no. 6, pp. 1291–1307, Nov. 2012.

[39] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[40] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Representations*, 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[41] C. Wu *et al.*, "Variance reduction for policy gradient with action-dependent factorized baselines," in *Proc. 6th Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1tSsb-AW

[42] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[43] W. Cook, "Concorde tsp solver," 2015. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/

[44] S. Shibusawa and T. Shibuya, "Reinforcement learning in the environment where optimal action value function is partly discontinuous," in *Proc. IEEE 55th Annu. Conf. Soc. Instrum. Control Engineers Japan*, 2016, pp. 1545–1550.

Yongyong Wei (Student Member, IEEE) received THE BS degree from Sun Yat-sen University, China, in 2010, and the MS degree in computer application technology from the University of Chinese Academy of Sciences, China, in 2013. He is currently working toward the PhD degree on mobile computing, under the supervision or Dr. Rong Zheng at McMaster University, Canada. He worked in SAP Labs China from 2013 to 2017. His current research interests include mobile computing, indoor localization, and machine learning.

Rong Zheng (Senior Member, IEEE) is currently a professor with the Department of Computing and Software, McMaster University. She is an expert in wireless networking, mobile computing and mobile data analytics. She received the National Science Foundation CAREER Award in 2006, and was a Joseph Ip distinguished engineering fellow from 2015–2018. She is currently an editor of *IEEE Transactions on Mobile Computing*.