Budget-Constrained Traveling Salesman Problem: a Cooperative Multi-Agent Reinforcement Learning Approach

King To Mak*, Christopher Gonzalez*, Zari Magnaye*, Jessica Gonzalez*, Yutian Chen†, Bin Tang*
*Department of Computer Science, California State University Dominguez Hills
{kmak5,cgonzalez393,zmagnaye1,jgonzalez1033}@toromail.csudh.edu, btang@csudh.edu
†Economics Department, California State University, Long Beach, Yutian.Chen@csulb.edu

***Abstract*—We study a new variation of the Traveling Salesman Problem (TSP) called the Budget-Constrained Traveling Salesman Problem (BC-TSP). BC-TSP is inspired by a few emerging network applications, such as robotic sensor networks. We design a prize-driven multi-agent reinforcement learning (MARL) framework to solve the BC-TSP. The main novelty of the framework, named P-MARL, is that it makes a connection between the prize maximization in BC-TSP and the cumulative reward maximization in reinforcement learning (RL) to design a more efficient MARL algorithm. In particular, P-MARL integrates the prizes available at nodes into the reward model of the MARL to guide the cooperative effort of multiple learning agents. Via extensive simulations using synthetic data of state capital cities of the U.S., we show that a) the P-MARL outperforms the existing *prize-oblivious* MARL work by collecting 28.8% of more prizes under the same budget constraints, b) it takes two orders of magnitudes of shorter training time than the state-of-the-art deep reinforcement learning-based approach while collecting 45.3% more prizes under the same budgets, and c) P-MARL collects prizes at least 91.9% of optimal obtained by the Integer Linear Programming (ILP) under different network parameters.**

Keywords – Budget-Constrained Traveling Salesman Problem, Multi-Agent Reinforcement Learning

I. Introduction

Background. The Traveling Salesman Problem (TSP) [26] is the most famous combinatorial optimization problem in computer science, engineering, and operation research. Since it was formulated mathematically in the 1930s, TSP has been applied to solve a wide range of applications ranging from traditional logistics of planning and control to recent developments in microchip manufacture, robotics, genome sequencing, and astronomy [11]. Traditional approaches to tackling TSP, which is NP-hard, include various optimal, approximation, and heuristics algorithms [26].

For the past few years, there has been a surge in solving combinatorial optimization problems, including the TSP, using reinforcement learning (RL) [15], [7], [31], [44], [42], [13]. This is partially fueled by the recent technological breakthrough of artificial intelligence (AI) and machine learning (ML), especially in the area of deep reinforcement learning (DRL) [22], [17], [25]. RL integrates machine learning and optimal control and models problem-solving as an intelligent agent interacting with the dynamic environment and taking actions to maximize its cumulative reward [38]. It is an algorithmic paradigm completely different from the above traditional and handcrafted approach. RL is particularly relevant to solving sequential combinatorial optimization problems such as TSP and the related vehicular routing problem (VRP), demonstrated by recent research progress [7], [31], [15], [44]. This is because when the traveling salesman or vehicles in the TSP and VRP decide to move from one node to another in a road network to serve the customers, it resembles the Markov decision process in RL, where the agent attempts to find an optimal policy that maps states into actions to maximize its collected reward. As such, RL has become an ideal alternative to solve many NP-hard combinatorial problems time-efficiently; see [30], [8] for the recent surveys in this burgeoning field.

Motivation. In this paper, we study a new variation of the TSP called the *Budget-Constrained Traveling Salesman Problem* (BC-TSP). In contrast to the traditional TSP, wherein the goal is to find a route to visit all the nodes in the most efficient manner, in BC-TSP, each node is associated with a *prize*, and the salesman has a *budget*; his goal is to visit as many nodes as possible to maximize the collected prizes while staying within his budget. Below is a motivating example.

Robotic Sensor Networks. In many robotic applications such as search and rescue and planetary exploration [17], [25], robots are dispatched to challenging environments to accomplish tasks of different importance. As a robot is mainly powered by batteries, it might exhaust its battery power before finishing all its assignment tasks. One critical goal is to schedule the untethered robot to accomplish as many critical tasks as possible before it returns to the charging station for recharging. One specific application is data collection in robotic sensor networks (RSNs) [23], where mobile robots are dispatched into large-scale sensor fields to collect sensory data. To optimize the performance of such RSN applications, how to schedule robots to collect as much useful information as possible before safely returning to the charging station is a new and challenging problem. It has not been studied by any of the existing work [12], [40], [41], [29]. BC-TSP can model any robotic application where robots or autonomous vehicles are dispatched to accomplish some tasks while their limited battery power is a significant obstacle to their lasting operation.

BC-TSP is NP-hard, as TSP is a special case of BC-TSP with an unconstrained budget. TSP only needs to sequence all the nodes. In contrast, maximizing the prizes under budget constraints in BC-TSP requires selecting a subset of the nodes and finding a sequence to visit them, making it a more challenging problem than the TSP.

Our Contributions. Unlike well-known combinatorial problems such as TSP and VRP, BC-TSP resembles RL in two ways. First, the prominent feature of the BC-TSP, *prizes* available at nodes, closely resembles the *rewards* in the RL. Such similarity provides an opportunity to design new RL algorithms to solve the BC-TSP better. Second, the objective of the BC-TSP, which is maximizing the prizes collected by the traveling salesman, closely resembles the RL agent's goal of maximizing the accumulative rewards. Both resemblances serve as a common ground upon which more powerful RL algorithms can be designed to solve the BC-TSP.

Despite the above observations, how to exploit the synergy between the prize maximization in BC-TSP and the cumulative reward maximization in RL to uncover more powerful RL algorithms remains largely unexplored by the research community. We ask the following question: *How can we take advantage of the unique problem feature of node prizes in BC-TSP to design more efficient and effective RL algorithms*?

We address this challenge by proposing a novel multi-agent reinforcement learning (MARL) framework, termed *prize-driven MARL* (P-MARL). P-MARL integrates the node prizes into the RL's reward model and action mechanism to not only guide the learning agents in maximizing their rewards but also yield an effective prize-collecting path under budget constraints. We take a unique *hybrid approach* in which multi-agents learn the node prizes independently and in parallel and then collaboratively find an efficient prize-routing path. Using a real-case application of 48 U.S. state capital cities, we show that P-MARL outperforms a well-known *prize-oblivious* MARL work called Ant-Q [16] by collecting up to 28.8% more prizes under different network parameters.

To further validate the competitiveness of P-MARL, we design a series of handcrafted combinatorial algorithms for BC-TSP, including an optimal Integer Linear Program (ILP) and two time-efficient greedy heuristics, wherein the salesman iteratively visits an unvisited *budget-feasible node* with the maximum available prize or the maximum prize-cost ratio (defined later). Via extensive simulations, we show that P-MARL collects prizes at least 91.9% of optimal obtained by the ILP and constantly outperforms the handcrafted heuristics by collecting more prizes.

II. Related Work

This section reviews all the essential related work in BC-TSP, RSN, DRL, and MARL.

Existing BC-TSP Research. A few works from the theory and operations research community have studied BC-TSP [37], [34], [5]. In his Ph.D. thesis, Sokkappa [37] systematically studied BC-TSP. He proved the problem is NP-hard and that no fully polynomial approximation scheme exists unless P = NP. He proposed branch-and-bound-based heuristics to solve the BC-TSP and optimal solutions for several special cases. Other efforts have been developed to find approximation algorithms for problems closely related to BC-TSP. Levin [5] presented a $(4+\epsilon)$-approximation algorithm to the so-called budget prize collecting tree problem, which finds a subtree with maximum prizes while the cost of the tree (i.e., the sum of all its edge weights) stays in a budget. Paul et al. [34] improved it by a 2-approximation algorithm based on a primal-dual approach while maximizing the number of vertices visited (i.e., each vertex has the same prize). However, none of them adopted an RL approach, which is the main focus of this paper. Recently, Ruiz et al. [35] studied a related problem called prize-collecting traveling salesman problem, wherein the goal is to collect a quota of prizes while minimizing the incurred cost. Our work is the first to apply the MARL technique to solve the BC-TSP.

Research in Robotic Sensor Networks (RSNs). BC-TSP is inspired by the data-collection in RSNs, wherein robots with limited battery power are dispatched to the sensor field to collect sensory data [29], [19], [40], [36], [41], [24], [43]. Ma et al. [29] introduced mobile data collectors to gather data in large-scale sensor networks. They formulate the problem as a mixed-integer program and present heuristic data-gathering algorithms. Guo et al. [19], [40] extended it by introducing wireless energy-charging into mobile data collecting. They formulated a network utility maximization problem that considered energy balance and the bounded sojourn time of the mobile collector and designed a distributed algorithm.

The above work assumes enough battery power for the robots to collect all the data in the field, which is not a valid assumption in a large-scale sensor field. When sensor nodes generate sensory data with different values (i.e., the prizes), a critical question is how to schedule the robot to collect data of maximum prizes before returning to the charging station. Recently, Patil et al. [33], [32] solved a special case of BC-TSP in the context of RSNs where the robot must start from and end at the same depot node. They designed a series of handcrafted combinatorial algorithms, which inspired our own ones compared with the P-MARL.

Deep Reinforcement Learning (DRL) for Combinatorial Optimization. Recently, DRL has been utilized to solve many combinatorial problems [15], [7], [31], [44], [13]. Bello et al. [7] presented a DRL-based framework for solving combinatorial optimization problems using neural network-based function approximation algorithms. They solved the TSP by training a recurrent neural network (RNN) and optimizing its parameters using a policy gradient method. Nazari et al. [31] further extend it to solve vehicle routing problems, a generalization of TSPs, by considering a parameterized stochastic policy and applying a policy gradient algorithm to optimize its parameters. Dai et al. [13] combined RL and graph embedding to incrementally construct a solution for combina-

torial optimization problems. Refer to [30] for a review of all the DRL techniques for combinatorial optimization.

We argue that DRL is not the best learning technique for solving BC-TSP. First, existing research uses deep learning algorithms mainly for feature extraction in high-dimensional spaces [14]. This is not needed in BC-TSP, as its most prominent feature (i.e., the node prizes) is already available. Second, training deep neural networks involves multiple iterations of optimization algorithms such as gradient descent and its variants with many parameters [6]. This is a computationally intensive process that either takes time to converge to optimal or, many times, only produces an approximate solution with a large optimality gap [7]. With its effective reward model and action mechanism involving prizes directly, P-MARL is a much more efficient technique than DRL to solve the BC-TSP.

The closest DRL work to ours is by Wei et al. [42], which proposed an RNN algorithm to solve the informative path planning problem (IPP). In IPP, a robot is dispatched into a sensing field to collect the sensing information, called *mutual information*, which measures the informativeness of data (i.e., sensor placement) collected along a path in the field. The informativeness of the path can be associated with the vertices, edges, or both on the path. IPP aims to find the most informative path from a pre-defined start location to a terminal location subject to a budget constraint. When the informativeness is defined on the vertices and is additive, IPP becomes the well-known orienteering problem (OP) [39]. In OP, each vertex is associated with a reward, and the goal is to find a subset of vertices to collect a maximum reward amount within a budget constraint. As OP is very similar to the BC-TSP studied in this paper, we compare our MARL algorithm with the RNN algorithm in [42]. In particular, we show that P-MARL outperforms the IPP by collecting 45.3% more prizes while taking 27.7 to 54.5% of its execution time.

Multi-Agent Reinforcement Learning (MARL). Most of the classic MARL algorithms [9], [28], [21], [27], [18] are derived from Q-learning [38]. In particular, Ant-Q [16] is one of the most prominent MARL frameworks combining the collective effort in colony optimization [10] and learning agents in RL [38]. Although P-MARL is inspired by Ant-Q [16], it significantly differs from Ant-Q in its goals and techniques. Ant-Q is designed to solve the traditional traveling salesman problem to minimize the cost of visiting *all the nodes*. As such, it is *prize-oblivious* and does not consider the node prizes of the salesman. In contrast, the P-MARL is to solve the BC-TSP by maximizing the collected prizes while staying within the budget. We integrate the node prizes into the multi-agent learning process to select a subset of nodes to visit and find its sequence to visit. We show in simulation that P-MARL outperforms Ant-Q by collecting 28.8% more prizes with the same budgets.

III. Problem Formulation and Combinatorial Algorithms for BC-TSP

In this section, we formulate the BC-TSP and design combinatorial algorithms including ILP and greedy algorithms.

A. Problem Formulation.

Given a weighted complete graph $G(V, E)$, where V is a set of nodes and E is a set of edges. Each edge $(u, v) \in E$ has a weight $w(u, v)$ and each node $i \in V$ has a prize $p_i \geq 0$. The salesman has a budget of $\mathcal{B}$ and starts at node s, visits a set of nodes to collect prizes, and stops at destination node d. The goal of the BC-TSP is to find a *prize-collecting path* $R = \{s = v_1, v_2, v_3, ..., v_a = d\}$, $a \leq |V|$, such that the total prize available on this route $P_R = \sum_{i \in R} p_i$ is maximized while its cost $C_R = \sum_{i=1}^{a-1} w(v_i, v_{i+1}) \leq \mathcal{B}$. Here, the *prize* models the importance of different tasks that various network applications attempt to accomplish, and the *budget* is a resource constraint in the network applications. Both prizes and budgets are application-specific. For example, the prizes could be the importance of the search and rescue missions or the value of the sensory data in the RSNs; the budget could be the battery power of the robots or the computing power of the agents in many AI/ML applications [22]. When s=d, the salesman starts and ends at the same node.

EXAMPLE 1: Fig. 1 illustrate BC-TSP with $\mathcal{B} = 8$. The numbers on the edges are their weights, and the numbers in the parentheses are the prizes available at nodes. Assume $s = E$ and $d = C$. The optimal route from E to C is E, D, B, and C, with a total prize of 8 and a total cost of 8. Other routes are not optimal. E.g., although the path of E, A, B, and C is within the budget with a cost of 7, its total prize is 7. □

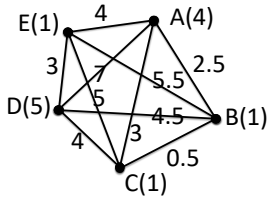


Fig. 1: Illustrating BC-TSP.

B. Combinatorial Algorithms for BC-TSP [33]

ILP Solution. We solve BC-TSP optimally by formulating it as an integer program ILP(A). *Decision variable* $x_{i,j}$ indicates if edge (i, j) is on the prize-collecting path (i.e., node j is visited immediately after node i is visited); $x_{i,j} = 1$ if so and 0 otherwise. We introduce $|V| - 1$ *position variables* u_i, $i \in V - \{s\}$, to indicate the order in which the nodes are visited. $u_s = 1$ as s is the starting node and $u_i < u_j$ indicates that node i is visited before node j (but not necessarily immediately). $u_i - 1$ equals the number of edges along the prize-collecting path from node s to node i.

$$(A) \qquad \max \sum_{i \in V \setminus \{d\}} \sum_{j \in V \setminus \{s\}} p_i \cdot x_{i,j} \tag{1}$$

s.t.

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j \in V \tag{2}$$

$$\sum_{j \in V \setminus \{s\}} x_{s,j} = \sum_{i \in V \setminus \{d\}} x_{i,d} = 1, \tag{3}$$

$$\sum_{i \in V \setminus \{d\}} x_{i,k} = \sum_{j \in V \setminus \{s\}} x_{k,j} \leq 1, \quad \forall k \in V \setminus \{s, d\} \tag{4}$$

$$\sum_{i \in V \setminus \{d\}} \sum_{j \in V \setminus \{s\}} w_{i,j} \cdot x_{i,j} \leq \mathcal{B}, \tag{5}$$

$$2 \leq u_i \leq |V|, \quad \forall i \in V \setminus \{s\} \tag{6}$$

$$u_i - u_j + 1 \leq (|V| - 1) \cdot (1 - x_{i,j}), \quad \forall i, j \in V \setminus \{s\} \tag{7}$$

Objective function 1 is to maximize the total collected prizes. Constraint 2 is the integer constraint of $x_{i,j}$. Constraint 3 guarantees that the prize-collecting path starts at node s and ends at node d. Constraint 4 ensures the connectivity of the path and that each node is visited at most once. Constraint 5 guarantees that the total traveling cost on the path does not exceed the given budget of $\mathcal{B}$. Constraints 6 and 7 combined are called Miller–Tucker–Zemlin (MTZ) Subtour Elimination Constraints [2]. They guarantee that one global tour visits all the selected vertices instead of multiple subtours, each visiting only a subset of the selected vertices.

Algorithm 1 Greedy Algorithm 1 for BC-TSP.

Input: A weighted complete graph $G(V, E)$, s, d, and $\mathcal{B}$.
Output: A route R from s to d, its cost C_R and prize P_R.
Notations: R: the current route found, initially $\{s\}$;
C_R: the length (i.e., the cost) of R, initially zero;
P_R: the prizes collected on R, initially zero;
U: the set of unvisited nodes, initially $V - \{s, d\}$;
r: the current node where the salesman is located;
B: current available budget, is $\mathcal{B}$ initially;

1: $r = s$, $R = \{s\}$, $C_R = P_R = 0$, $B = \mathcal{B}$, $U = V - \{s, d\} = \{v_1, v_2, ..., v_{|V|-2}\}$;
2: Sort nodes in U in descending order of their prizes; WLOG, let $p_{v_1} \geq p_{v_2} ..., \geq p_{v_{|V|-2}}$;
3: $k = 1$; // the index of the node with the largest prize
4: **while** $(U \neq \phi \wedge \mathcal{F}(r, B) \neq \phi)$ **do**
5: **if** $(v_k \in \mathcal{F}(r, B))$ **then**
6: $R = R \cup \{v_k\}$;
7: $C_R = C_R + w(r, v_k)$, $P_R = P_R + p_{v_k}$;
8: $B = B - w(r, v_k)$, $U = U - \{v_k\}$;
9: $r = v_k$;
10: **end if**
11: $k++$;
12: **end while**
13: $R = R \cup \{d\}$, $C_R = C_R + c(r, d)$, $B = B - c(r, d)$;
14: **return** R, C_R, P_R, B.

Definition 1: (**Budget-Feasible Nodes.**) Given the node r the traveling salesman is currently located and his available budget B, its *budget-feasible nodes*, denoted as $\mathcal{F}(r, B)$, is the set of unvisited non-destination nodes that the salesman can travel to and then reaches destination node d. That is, $\mathcal{F}(r, B) = \{u | (w(r, u) + w(u, d) \leq B) \wedge u \in U\}$, where U is the set of unvisited nodes, initially $U = V - \{s, d\}$. □

Greedy Algorithm 1. In Algo. 1, at any node, the salesman always visits a budget-feasible node with the largest prize. It first sorts all the nodes in the non-ascending order of their prizes (line 2) and then takes place in rounds (lines 4-12). In each round, with the current node r and the currently available budget B, it checks if there exist unvisited and budget-feasible nodes (line 4). If so, it visits the one with the largest available prize and updates all the information accordingly (lines 5-10). Ties are broken randomly. It stops when there are no unvisited nodes, or all the unvisited nodes are not budget-feasible (line 4), at which it goes to the destination node t and returns the route with its total cost, total prizes collected, and its remaining budget (lines 13 and 14). Its time complexity is $O(|V|^2)$.

Greedy Algorithm 2. Given an edge $(u, v) \in E$, and the traveling salesman is at node u, we define the *prize cost ratio* of visiting v, denoted as $pcr(u, v)$, as the ratio between the prize available at v and the edge weight $w(u, v)$. That is, $pcr(u, v) = \frac{p_v}{w(u,v)}$. Greedy Algorithm 2 is similar to Algo. 1, except it visits a budget-feasible node with the largest prize cost ratio in each round. Its time complexity is $O(|V|^2)$. We omit its pseudocode due to space constraints. Note that both greedy algorithms also work for the instances with $s = d$.

EXAMPLE 2: In Fig. 1, with $s = E$ and $d = C$, both greedy algorithms yield the optimal solution of E, D, B, and C, with a total cost of 8 and a total prize of 8. But in general, they are not optimal. □

IV. Prize-Driven Multi-Agent Reinforcement Learning Algorithm (P-MARL) for BC-TSP

In this section, we introduce the basic idea of RL, including Q-learning, in the context of prize-collecting in BC-TSP. Then, we introduce different stages and phases of prize-collecting on top of the Q-learning that are conducive to multi-agent learning. Finally, we present the P-MARL algorithm.

A. RL for BC-TSP

Prize-Collecting RL. We model an agent's decision-making for prize-collecting as a Markov decision process (MDP) represented by a 4-tuple (S, A, t, r) [38]:

- S is a finite set of *states*,
- A is a finite set of *actions*,
- $t : S \times A \to S$ is a *state transition function*, and
- $r : S \times A \to R$ is a *reward function*, where R is a real value reward.

The states S are all the nodes in a BC-TSP graph, and the actions A available at a node are all the other nodes, as we consider the complete graph. At any state $s \in S$, the agent takes action $a \in A$ to transition to another state $t(s, a) \in S$ while receiving a reward $r(s, a) \in R$. A *policy* $\pi(s) : S \to A$ of the agent maps its current state $s \in S$ into a desirable action $a \in A$. We consider a *deterministic* policy wherein, given the state, the policy outputs a specific action for the agent. A deterministic policy suits the BC-TSP well, as when an agent at a node takes action (i.e., follows one of its edges), it will reach the other node of the edge. Note that existing research of solving combinatorial optimization problems using DRL (e.g., [13], [42]) treats the sequence of visited nodes or partial solution as a stage, as they incrementally search for an optimal solution using RL and graph embedding techniques. Our approach is completely different, wherein multiple agents independently and collaboratively find the optimal route for BC-TSP. Thus each node is treated as a different state.

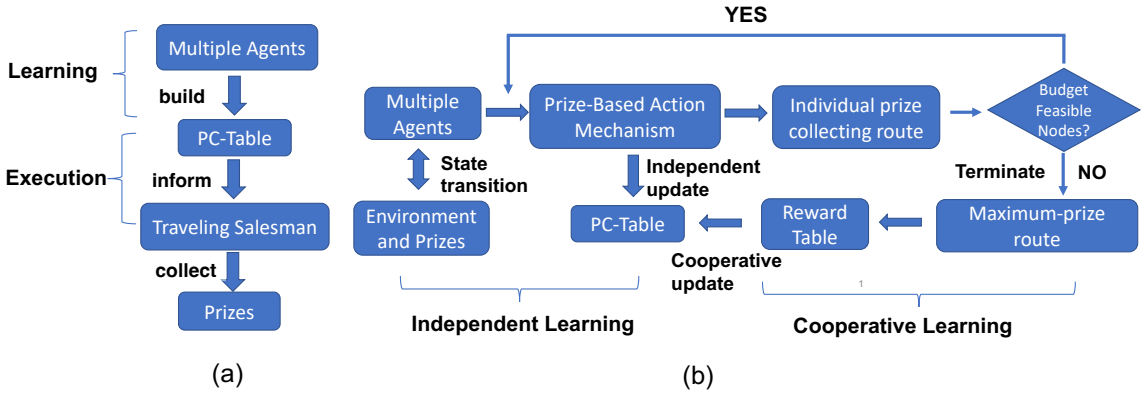


Fig. 2: (a) Two stages of P-MARL: learning stage and execution stage. (b) Workflow of an episode in the learning stage. Each episode has two phases: independent learning and cooperative learning. In the independent learning phase, each agent independently takes actions and updates the PC-Table. In the cooperative learning phase, all the agents cooperatively update the reward table and PC-Table using the maximum-prize route found in this episode.

Q-Learning. Q-learning is a family of value-based RL algorithms [38] wherein an agent learns an optimal policy that maximizes its accumulated reward. To achieve that, it learns how to optimize the quality of its actions in terms of the Q-value $Q(s,a)$. $Q(s,a)$ is the expected discounted sum of future rewards obtained by taking action a from state s following an optimal policy. The optimal action at any state is the action that gives the maximum Q-value. For an agent at state s, when it takes action a and transitions to the next state t, $Q(s,a)$ is updated using the Bellman equation below as a simple value iteration update until no more improvement:

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a)+\alpha\cdot[r(s,a)+\gamma\cdot \max_b Q(t,b)]. \quad (8)$$

In Eqn. 8, $1 \leq \alpha \leq 1$ is the *learning rate* and $\max_b Q(t,b)$ is the maximum reward that can be obtained from the next state t. For BC-TSP, starting from source node s, the traveling salesman follows the Q-table by moving to node $r = \text{argmax}_b Q(s,b)$ to collect its prize p_r; this takes place until it reaches the destination d. As the Q-table encodes the prize-collecting path the traveling salesman takes, we also refer to it as *PC-Table*.

B. P-MARL for BC-TSP

P-MARL augments Q-learning into a multi-agent scenario that consists of two stages. In *learning stage*, the agents *independently* and *cooperatively* build the PC-Table, and in *execution stage*, the traveling salesman follows the PC-Table to find the prize-collecting path and to collect the prizes. Fig. 2(a) shows these two stages.

The learning stage itself takes place in episodes. We propose a hybrid approach to effectively learn prize-collecting using PC-Table and divide an episode into two phases, as shown in Fig. 2(b). In the *independent learning phase*, each agent independently follows a *prize-based action mechanism* and updates the PC-Table. In the *cooperative learning phase*, agents cooperatively update the reward table and PC-Table to encode prize-collecting paths with significant prizes. We show via extensive simulations that this hybrid approach, wherein agents collaborate upon their in-parallel and individual exploration, greatly empowers the learning experiences of the multi-agents for prize-collecting. Below, we illustrate these two phases.

Independent Learning Phase. It consists of a *prize-based action mechanism* and an *independent update of the PC-Table*.

Prize-based Action Mechanism. Located at node r, an agent moves to the next node t to collect prizes by taking one of the three actions below.

i) Exploitation, in which the agent always chooses the node

$$t = \text{argmax}_{u \in U \cap \mathcal{F}(r,B)} \frac{[Q(r,u)]^\delta \times p_u}{[w(r,u)]^\beta} \quad (9)$$

to move to. Here, U is the set of nodes not visited yet by the agent and $\mathcal{F}(r,B)$ is the set of budget-feasible nodes of the agent with budget B, and δ and β are preset parameters. That is, an agent always moves to an unvisited budget-feasible node u that maximizes the learned Q-value $Q(r,u)$ weighted by the prize p_u available at node u and the length $w(r,u)$. When $q > q_0$, where q is a random value in [0, 1] and q_0 $(0 \leq q_0 \leq 1)$ is a preset value, exploitation is selected; otherwise, the agent takes the action of exploration below.

ii) Exploration, in which the agent moves to a node t by following probability distribution:

$$p(r,t) = \frac{([Q(r,t)]^\delta \times p_t)/[w(r,t)]^\beta}{\sum_{u \in U \cap \mathcal{F}(r,B)} ([Q(r,u)]^\delta \times p_u)/[w(r,u)]^\beta}. \quad (10)$$

That is, a node $u \in U \cap \mathcal{F}(r,B)$ is selected with probability $p(r,u)$, while $\sum_{u \in U \cap \mathcal{F}(r,B)} p(r,u) = 1$. $p(r,t)$ characterizes how good to move from node r to node t in terms of the $Q(r,t)$, $w(r,t)$ and p_t. The smaller the $w(r,t)$ and the larger the $Q(r,t)$ and p_t, the more desirable to move to t.

iii) Termination, in which the agent does not have unvisited budget-feasible nodes to move to, i.e., $U \cap \mathcal{F}(r,B) = \phi$. It goes to destination t and terminates in this episode.

Eqns. 9 and 10 are the classic exploration-exploitation tradeoff in RL, however, with node prizes considered to facilitate the learning of the prize-collecting in BC-TSP. Note that different agents in the learning stage can collect the prize at each node multiple times.

Independent Update of PC-Table. After moving from node r to node t following the above action mechanism, each agent independently updates the PC-Table as follows:

$$Q(r,t) = (1-\alpha)\cdot Q(r,t) + \alpha\cdot\gamma\cdot \max_{b\in U\cap\mathcal{F}(t,B)} Q(t,b). \quad (11)$$

Compared to the original Q-learning update Eqn. 8, one difference of Eqn. 11 is that it doesn't include the reward value $r(r,t)$ in updating $Q(r,t)$. This is because all agents work independently and have not yet found the route with a large prize. Instead, each agent uses its own discounted next-state evaluation to update the PC-Table, independently contributing to its update with their own experiences.

All agents repeat the above independent learning process until they reach the destination d before exhausting budgets. At this point, they each have found their prize-collecting paths and enter the *cooperative learning phase* stated below.

Cooperative Learning Phase. This phase mainly consists of a *prize-based reward model* for the agents to update the PC-Table cooperatively.

Prize-based Reward Model. The reward model works as follows. First, the cooperative agents communicate with each other and compare the prizes of their prize-collecting paths. They find the one, say, R_{max}, with the maximum prize, say, P_{max}. Second, for each edge (u,v) in R_{max}, they increase its reward value $r(u,v)$ as follows:

$$r(u,v) = r(u,v) + \frac{W}{P_{max}}, \qquad W \text{ is a constant.} \quad (12)$$

Finally, they update the Q-value $Q(u,v)$ of those edges $(u,v) \in R_{max}$ as follows:

$$Q(u,v) = (1-\alpha)\cdot Q(u,v) + \alpha\cdot[r(u,v) + \gamma\cdot \max_b Q(v,b)].$$

Each episode, with the above independent and cooperative learning phases, is repeated until a fixed number of episodes is reached or the PC-Table is no longer updated. Next, we present our P-MARL algorithm, viz. Algo. 2.

P-MARL Algorithm. It begins with all the m agents at the source node s with a collected prize of p_s (lines 2-5). It consists of two stages viz. a *learning stage* wherein the agents independently and cooperatively encode the prize-collecting path into the PC-Table (lines 1-29) and an *execution stage* for the traveling salesman to follow the PC-Table to visit the nodes and collect the prizes (lines 30-35).

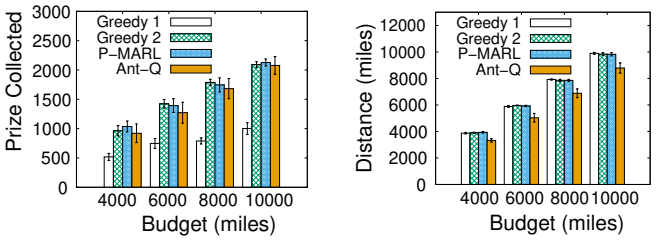
In the *independent learning phase* (lines 6-23), when an agent has enough budget, it independently follows the action mechanism and updates the PC-Table (lines 9-14). Otherwise, it goes to destination d and stops for this episode while waiting for other agents to terminate (lines 16-19). In the *cooperative learning phase* (lines 24-28), the m agents compare their prize-collecting paths to find the maximum-prize route and update the reward value and Q-value of the edges of this route.

Finally, in the *execution stage* (lines 30-35), the salesman starts from s, visits the node with the largest Q-value in the PC-Table, and ends at d, collecting the prizes along the way. It returns the route, its cost and prize, and the remaining budget.

Algorithm 2 P-MARL Algorithm for BC-TSP.

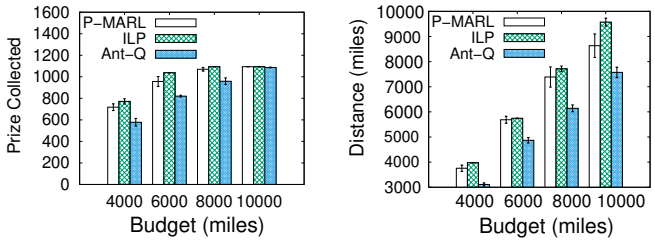
Input: A weighted complete graph $G(V,E)$, s, d, and $\mathcal{B}$.
Output: A prize-collecting path R from s to d, C_R, and P_R.
Notations: i: index for episodes; j: index for agents;
epi: number of episodes in the P-MARL;
B_j: the available budget of agent j in an episode, initially $\mathcal{B}$;
U_j: set of nodes agent j not yet visits, initially $V-\{s,d\}$;
R_j: the route taken by agent j, initially $\{s\}$;
l_j: the cost (i.e., the sum of edge weights) of R_j, initially 0;
P_j: the prizes collected on R_j, initially p_s;
r_j: the node where agent j is located currently;
s_j: the node where agent j moves to next;
$isDone_j$: agent j has finished in this episode, initially false;
$Q(u,v)$: Q-value of edge (u,v), initially $\frac{p_u+p_v}{w(u,v)}$;
$r(u,v)$: reward value of (u,v), initially 0;
R, C_R, P_R: the prize-collecting path, its cost, and its prize;

1: **for** $(1 \le i \le epi)$ **do**
2: **for** $(j=1; j\le m; j{+}{+})$ **do**
3: $r_j = s$, $isDone_j = false$; // All m agents are at s
4: $P_j = p_s$, $B_j = \mathcal{B}_j$;
5: **end for**
6: **while** $(\exists\ j,\ 1\le j\le m,\ isDone_j == false)$ **do**
7: **for** $(j=1; j\le m; j{+}{+})$ **do**
8: **if** $(isDone_j == false)$ **then**
9: **if** $(U_j \cap \mathcal{F}(r_j, B_j) \neq \phi)$ **then**
10: Finds the next node s_j following action rule;
11: $R_j = R_j \cup \{s_j\}$, $P_j = P_j + p_{s_j}$;
12: $l_j = l_j + w(r_j, s_j)$, $B_j = B_j - w(r_j, s_j)$;
13: $Q(r_j,s_j) = (1-\alpha)\cdot Q(r_j,s_j) + \alpha\cdot\gamma\cdot \max_{b\in U_j\cap\mathcal{F}(s_j,B_j)} Q(s_j,b)$;
14: $r_j = s_j$, $U_j = U_j - s_j$;
15: **else**
16: $isDone_j = true$, $R_j = R_j \cup \{d\}$;
17: $l_j = l_j + w(r_j,d)$, $B_j = B_j - w(r_j,d)$;
18: $Q(r_j,d) = (1-\alpha)\cdot Q(r_j,d) + \alpha\cdot\gamma\cdot \max_{b\in U_j\cap\mathcal{F}(t,B_j)} Q(t,b)$;
19: $r_j = d$; // Arrive at destination node d
20: **end if**
21: **end if**
22: **end for**
23: **end while**
24: $j^* = \text{argmax}_{1\le j\le m} P_j$;
25: **for** (each edge $(u,v) \in R_{j*}$) **do**
26: $r(u,v) = r(u,v) + \frac{W}{P_{j*}}$; // W is a constant
27: $Q(u,v) = (1-\alpha)\cdot Q(u,v) + \alpha\cdot[r(u,v) + \gamma\cdot \max_b Q(v,b)]$;
28: **end for**
29: **end for**
30: $r = s$, $R = \{s\}$, $C_R = 0$, $P_R = p_s$, $B = \mathcal{B}$;
31: **while** $(r != d)$ **do**
32: $u = \text{argmax}_b Q(r,b)$;
33: $R = R \cup \{u\}, P_R = P_R + p_u$, $C_R = C_R + w(r,u)$, $B = B - w(r,u)$, $r = u$;
34: **end while**
35: **return** R, C_R, P_R, B.



(a) Total Prize. (b) Total Distance.

Fig. 3: Performance comparison in large networks.



(a) Total Prize. (b) Total Distance.

Fig. 4: Performance comparison with ILP in small networks.

We set the initial Q-value of edge (u, v) as $\frac{p_u+p_v}{w(u,v)}$, to reflect the fact that the more prizes available and less length of an edge, the more valuable of the edge for the salesman to travel.

Discussions. There are two ways to end the training. One is to run a pre-specified number of epi episodes for training. In this case, in each episode, the first phase takes at most $m \cdot |V|$, where $|V|$ is the total number of nodes, and the second phase takes at most $m+|E|$, where $|E|$ is the total number of edges. Thus, the time complexity of Algo. 2 is $O(epi \cdot m \cdot |V|)$. The second way is to terminate the training when the global-best route is no longer updated after a specified number of episodes. We use both approaches in our simulations.

We leave the convergence study of Algo. 2 as a future work. Gutjahr et al. [20] proposed a graph-based framework to study the ant system's convergence. The framework is based on a construction graph assigned to an instance of the optimization problem under consideration, encoding feasible solutions by walks. Given a problem instance, the solutions generated can converge with a high probability of being arbitrarily close to the optimal solution. However, as a general framework, it does not tackle specific combinatorial problems, including the BC-TSP. Considering the simple definition and the problem features inherent in the BC-TSP problem, studying the convergence of the P-MARL is promising future research.

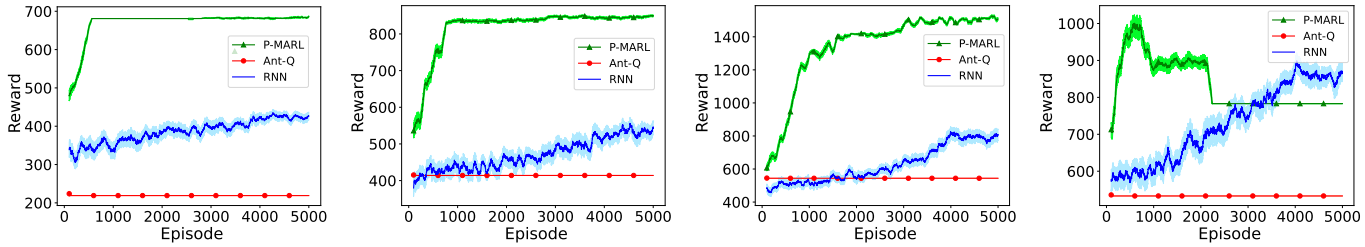
V. Performance Evaluation

Experiment Setup. We use the traveling salesman tour of 48 state capital cities on the US mainland [4]. Given the latitude and longitude of each city, the distance between any two cities can be computed using the Haversine formula [1]. The prize at each city is a random number in [1, 100]. Given that the shortest distance visiting all 48 continental US capitals is 12,930 miles [4], we vary the budget from 4,000 miles to 10,000 miles so that budget serves as a constraint. We set the number of agents m as 5 in all cases. Each data point in our plots is an average of 10 runs with a 95% confidence interval; in each run, a city is randomly chosen as the source and destination for the traveling salesman.

We refer to our P-MARL algorithm Algo. 2 as **P-MARL**. Our comparison consists of two parts. The first part compares P-MARL with various non-DRL algorithms (Figs. 3 and 4). We refer to the optimal ILP solution viz. ILP(A) as **ILP**, the two greedy heuristics as **Greedy1** and **Greedy2**, respectively. We refer to the Ant-Q algorithm [16], which is a MARL oblivious of the prizes, as **Ant-Q**. For fair comparison under the budgets in BC-TSP, we modify the Ant-Q algorithm (Fig. 1 in [16]) such that each agent must return to the destination node when there is insufficient budget. The hyper-parameters in P-MARL and Ant-Q are: $\alpha = 0.1$, $\gamma = 0.3$, $q_0 = 0.5$, $\delta = 1$, $\beta = 2$, $W = 1500$, and number of episodes in the training stage $epi = 5000$. We write our simulator in Java on a Windows 10 with an AMD Processor (AMD Ryzen 7 3700X 8-Core) and 16GB of DDR4 memory.

For the second part, we compare with the state-of-the-art DRL approach [42], which designed a recurrent neural network (RNN) based Q-learning algorithm to solve the informative path planning problem. We refer to this approach as **RNN**. As this is the latest DRL work that solves a similar problem, we compare P-MARL and Ant-Q with RNN (Figs. 5, 6, and Table I). As RNN adopts PyTorch [3], the deep learning platform, for a fair comparison, we implement RNN, P-MARL, and Ant-Q in PyTorch to take advantage of its full feature of building deep learning models. This part is implemented on a MacOS 13.3.1 with an Apple M1 Pro processor and 16 GB of unified memory.

Performance Comparison in Large Networks. We first compare all algorithms using the large network of 48 cities by varying the budgets. Fig. 3(a) shows in terms of total prizes collected, P-MARL $>$ Greedy2 $>$ Ant-Q $>$ Greedy1. This demonstrates that P-MARL is the most competitive among the four prize-collecting algorithms for BC-TSP. Greedy2 comes next as it uses the prize-cost ratio to balance prizes collected and distance traveled. Ant-Q comes third, as it is prize-oblivious and does not try to maximize prizes. Greedy1 collects significantly fewer prizes than the other three. Although it always collects the largest size possible, and for the same reason, it could travel long distances to do so and thus exhaust its budget quickly, resulting in the least amount of prizes being collected. In particular, the P-MARL outperforms Ant-Q by up to 16.9%. Fig. 3(b) shows that PMARL, Greedy1, and Greedy2 travel the same distances, constrained by the given budget. In contrast, Ant-Q travels a significantly smaller distance and does not exhaust its budget. This is because to maximize the prizes while staying within budget, we introduce "budget-feasible nodes" for those three algorithms. Ant-Q does not have this facility. Instead, as its goal is to minimize the travel cost, it tends to collect prizes in its proximity. Although it has some unused budget, it is not big enough for the salesman to



(a) Budget = 4,000 miles. (b) Budget = 6,000 miles. (c) Budget = 8,000 miles. (d) Budget = 10,000 miles.

Fig. 5: Comparing P-MARL, Ant-Q, and RNN in the training stage.

travel further to collect prizes.

Performance Comparison with ILP in Small Networks. Next, we compare P-MARL and Ant-Q with the ILP to investigate how close their performance is to the optimal solution. As ILP takes a long time to execute in 48-city cases, we randomly chose 20 cities for comparison. Fig. 4(a) shows that in most cases, P-MARL collects prizes at least 91.9% of optimal prizes by the ILP, while collecting up to 28.8% more prizes than Ant-Q. However, when the budget is 10,000, all three algorithms collect around the same amount of prize. This is because this is enough budget for all the algorithms to visit all the 20 nodes, thus collecting the same prize. Fig. 4(b) shows Ant-Q travels less than P-MARL and ILP, focusing on short prize-collecting distances.

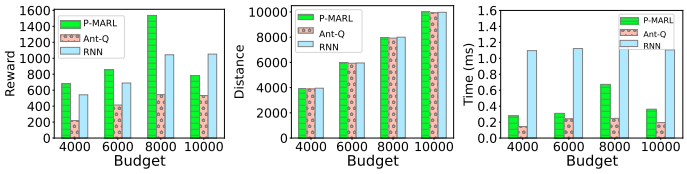
Comparing P-MARL, Ant-Q, and RNN. Next, we implement P-MARL, Ant-Q, and RNN in PyTorch, which has a full feature of deep learning models. We compare them in both the training and execution stages. Fig. 5 shows their rewards (i.e., the prize of the maximum-prize route found in each episode) in the training stage of 5,000 episodes. We randomly chose a city as the source and destination node. The rewards for RNN are computed as a running average from 100 episodes and with a 95% confidence interval following [42]. It shows that under various budgets, in terms of total prizes collected, P-MARL $>$ RNN $>$ Ant-Q, again demonstrating that P-MARL is the most competitive prize-collecting algorithm for BC-TSP besides the ILP. Furthermore, P-MARL learns to increase its prizes quickly and smoothly, as shown by the steep climb in early episodes and the flat plateau in later episodes. In contrast, RNN learns slowly and with high variations of the collected prizes at different episodes. This can be attributed to the "black box" nature of neural networks, including RNNs, which are a computationally intensive process that takes time to compute and, many times, only produces a solution with a large optimality gap [7].

On the other hand, Ant-Q, being a prize-oblivious MARL algorithm, does not seek to increase the prizes at all in the learning process, resulting in the least amount of prizes in all three algorithms and a plateau from the early stage of episodes. Table I shows their running time of the learning stage (i.e., the training time) under different budgets. As RNN trains deep neural networks with a large number of weight and bias parameters and involves computation-intensive optimization algorithms such as gradient descent, its training time is two orders of magnitudes longer than P-MARL and Ant-Q. Ant-Q has a shorter training time than P-MARL, as it focuses on the shortest prize-collecting path.

Budget (miles)	4000	6000	8000	10000
P-MARL	11.27	14.04	26.87	18.99
Ant-Q	5.93	11.19	10.69	9.42
RNN	692.62	968.34	1122.51	1221.34

TABLE I: Comparing training time of P-MARL, Ant-Q, and RNN.

Finally, we compare the final prize-collecting results in the execution stage, wherein a traveling salesman follows the learned Q-table to visit different cities to collect the prizes. Fig. 6(a) shows that P-MARL outperforms RNN and Ant-Q in collecting more prizes in most cases (except when at the large budget of 10,000), revealing P-MARL is an effective prize-collecting algorithm. In particular, P-MARL collects up to 45.3% more prizes than RNN. Fig. 6(b) shows that all the algorithms are constrained by the budget; as such, they all travel the same distance. Fig. 6(c) comparing the execution time of three algorithms. It shows that P-MARL takes only 27.7 to 54.5% of the execution time of RNN, due to the high computational cost of DRL [6]). We also observe that Ant-Q has the smallest execution time. As Ant-Q finds the shortest-distance route within the budget constraint instead of P-MARL's maximum-prize route of a possibly longer distance, it takes less time for the salesman to travel, resulting in less execution time than P-MARL.



(a) Total Prize. (b) Total Distance. (c) Execution Time.

Fig. 6: Comparing P-MRL, Ant-Q, RNN in the execution stage.

VI. Conclusions

Budget-constrained TSP (BC-TSP) uniquely arises from emerging network applications, including robotic sensor networks, electric cars in ride-sharing, and automated warehouses, wherein robots or autonomous vehicles with limited battery power are dispatched to accomplish some tasks. With

its theoretical roots, BC-TSP can model any sequential decision problem targeting tasks of different importance under resource constraints. We utilized the unique problem features of BC-TSP to design a MARL algorithm called P-MARL. Using a real-case application, we show that P-MARL outperforms the state-of-the-art DRL work, prize-oblivious MARL work, and traditional handcrafted combinatorial algorithms while performing close to the ILP optimal solution under different network parameters. To our knowledge, our work is the first to utilize problem features of combinatorial optimization problems in general and BC-TSP in particular to design efficient MARL algorithms. We hope this perspective can stir some discussion about how to utilize problem features and cooperations of simple RL agents to better solve combinatorial optimization problems, in contrast to many existing works that resort to the brute force of DRL.

ACKNOWLEDGMENT

This work was supported by NSF Grant CNS-2240517 and the Google exploreCSR Grant.

REFERENCES

[1] Haversine formula. https://en.wikipedia.org/wiki/Haversine_formula.

[2] Miller–tucker–zemlin (mtz) subtour elimination constraint. https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html.

[3] Pytorch deep learning libraries. https://pytorch.org/.

[4] Traveling salesman tour of us capital cities. https://www.math.uwaterloo.ca/tsp/data/usa/index.html.

[5] A better approximation algorithm for the budget prize collecting tree problem. *Operations Research Letters*, 32(4):316–319, 2004.

[6] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017.

[7] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

[8] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Euro. Jour. of Oper. Res.*, 290(2):405–421, 2021.

[9] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artif. Intell.*, 136(2):215–250, 2002.

[10] B. Chandra Mohan and R. Baskaran. A survey: Ant colony optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 39(4):4618–4627, 2012.

[11] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40, may 2021.

[12] L. Chen, W. Wang, H. Huang, and S. Lin. On time-constrained data harvesting in wireless sensor networks: Approximation algorithm design. *IEEE/ACM Transactions on Networking*, 24(5):3123–3135, 2016.

[13] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Proc. of NIPS 2017*.

[14] S. Dara and P. Tumma. Feature extraction by using deep learning: A survey. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1795–1801.

[15] I. Drori et al. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 19–24, 2020.

[16] L. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *ICML*, 1995.

[17] L. C. Garaffa, M. Basso, A. A. Konzen, and E. P. de Freitas. Reinforcement learning for mobile robotics exploration: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3796–3810, 2023.

[18] A. Greenwald and K. Hall. Correlated-q learning. In *20th Int. Conf. Mach. Learn. (ICML-03)*.

[19] S. Guo, C. Wang, and Y. Yang. Joint mobile data gathering and energy provisioning in wireless rechargeable sensor networks. *IEEE Transactions on Mobile Computing*, 13(12):2836–2852, 2014.

[20] W.J. Gutjahr. A graph-based ant system and its convergence. *Future Generation Computer Systems*, 16:873–888, 2000.

[21] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proc. of ICML-98*.

[22] J. Hua, L. Zeng, G. Li, and Z. Ju. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21(4), 2021.

[23] H. Huang, A. V. Savkin, M. Ding, and C. Huang. Mobile robots in wireless sensor networks: A survey on tasks. *Computer Networks*, 148:1–19, 2019.

[24] D. Kim, L. Xue, D. Li, Y. Zhu, W. Wang, and A. O. Tokuta. On theoretical trajectory planning of multiple drones to minimize latency in search-and-reconnaissance operations. *IEEE Transactions on Mobile Computing*, 16(11):3156–3166, 2017.

[25] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. 32(11), 2013.

[26] Gilbert Laporte. The traveling salesman problem: An overview of the exact and approximate algorithms. *European Journal of Oper. Res.*, 59:231–247, 1992.

[27] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proc. of ICML 2000*.

[28] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the ICML 1994*.

[29] M. Ma, Y. Yang, and M. Zhao. Tour planning for mobile data-gathering mechanisms in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 62(4):1472–1483, 2013.

[30] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134, 2021.

[31] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio et al., editor, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[32] S. Patil, Y. Chen, and B. Tang. A convergent multi-agent reinforcement learning algorithm for dynamic data collection in robotic sensor networks. Technical report, CSUDH-TR-2024-1, 2024.

[33] S. Patil, Y. T. Chen, and B. Tang. Revisiting data collection in robotic sensor networks: A budget-constrained traveling salesman perspective. In *Proc. of IEEE MASS*, 2024.

[34] A. Paul, D. Freund, A. Ferber, D. B. Shmoys, and D. P. Williamson. Prize-collecting tsp with a budget constraint. In *25th Annual European Symposium on Algorithms (ESA 2017)*.

[35] J. Ruiz, C. Gonzalez, Y. Chen, and B. Tang. Prize-collecting traveling salesman problem: a reinforcement learning approach. In *Proc. of IEEE ICC*, 2023.

[36] H. Salarian, K. W. Chin, and F. Naghdy. An energy-efficient mobile-sink path selection strategy for wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 63(5):2407–2419, 2014.

[37] P. R. Sokkappa. The cost-constrained traveling salesman problem, 1991. Ph.D. Thesis, Stanford University.

[38] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. The MIT Press, 2020.

[39] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Oper. Res.*, 255(2):315 – 332, 2016.

[40] C. Wang, S. Guo, and Y. Yang. An optimization framework for mobile data collection in energy-harvesting wireless sensor networks. *IEEE Transactions on Mobile Computing*, 15(12):2969–2986, 2016.

[41] Y.-C. Wang and K-C. Chen. Efficient path planning for a mobile sink to reliably gather data from sensors with diverse sensing rates and limited buffers. *IEEE Trans. on Mobi. Com.*, 18(7):1527–1540, 2019.

[42] Y. Wei and R. Zheng. Informative path planning for mobile sensing with reinforcement learning. In *IEEE INFOCOM 2020*.

[43] L. Xue, D. Kim, Y. Zhu, D. Li, W. Wang, and A. O. Tokuta. Multiple heterogeneous data ferry trajectory planning in wireless sensor networks. In *IEEE INFOCOM 2014*, pages 2274–2282.

[44] R. Zhang, C. Zhang, Z. Cao, W. Song, P. S. Tan, J. Zhang, B. Wen, and J. Dauwels. Learning to solve multiple-tsp with time window and rejections via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2022.