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

***Abstract*—The Traveling Salesman Problem (TSP) is one of the most famous combinatorial problems in computer science. Inspired by a few emerging network applications, including robotic sensor networks and autonomous electric vehicles, we study a new variation of the TSP called the Budget-Constrained Traveling Salesman Problem (BC-TSP). Given a weighted complete graph $G(V, E)$ where node $i \in V$ has an available prize of p_i, two nodes $s, t \in V$, and a budget, the goal of the BC-TSP is to find the salesman a route from s to t to maximize his collected prizes while keeping his travel cost within the budget. We design a suite of algorithms, including an Integer Linear Programming (ILP)–based optimal solution, two iterative greedy algorithms, and a multi-agent reinforcement learning (MARL) algorithm to solve BC-TSP. We prove rigorously that our MARL algorithm converges to optimal. Via extensive simulations, we show that the MARL algorithm outperforms the two greedy algorithms and performs close to the ILP-based optimal solution (within 12.7% of the optimal). This demonstrates that MARL is an effective and efficient algorithm for solving BC-TSP. To the best of our knowledge, our work is the first one to apply the MARL technique to solve the BC-TSP problem.**

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

I. Introduction

Background and Motivation. The Traveling Salesman Problem (TSP) is arguably the most famous combinatorial problem in computer science, engineering, and operation research [29], [15]. In this paper, inspired by a few emerging network applications, 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 traveling salesman has a budget; the goal of the salesman is to visit a subset of the nodes to maximize the collected prizes while staying within his budget. Thus, unlike the TSP which only needs to sequence the nodes, the BC-TSP requires both selection and sequencing of the nodes and becomes more challenging.

PC-TSP is motivated by many emerging robotic applications [21], [28], wherein one or multiple robots are dispatched to accomplish some tasks such as search and rescue and planetary exploration. As robots are mainly powered by batteries, one critical goal for the untethered robot is to accomplish as many tasks as possible before returning to the charging station for recharge. One specific example is data collection in robotic sensor networks (RSNs) [26], where mobile robots are dispatched into sensor fields to collect sensory data. It has been shown that this approach can greatly prolong the network lifetime of sensor networks by migrating the energy bottleneck from sensor nodes to robots [44], [16].

Another motivating example is Uber driving with electric cars, wherein an Uber driver, starting from his home, picks up and drops off customers at different locations before getting to a charging station to recharge. Given a sequence of ride requests offering different payments and the maximum mileage provided by the electrical battery of the car, a natural question for the driver is how to maximize the number of payments before running out of battery power and getting to a charging station or home to recharge the vehicle.

BC-TSP is defined as follows. Given a weighted complete graph $G(V, E)$ where node $i \in V$ has an available prize of p_i, two nodes $s, t \in V$, and a budget, the goal of the salesman is to find a route from s to t to maximize his collected prizes while keeping his travel cost within the budget. Here, the *prizes* model any tasks that network applications try to accomplish, and the *budget* is a resource constraint in the network applications. Therefore, both prizes and budget are application-specific. For example, the prizes could be the importance of the search and rescue missions or the data to be collected in the RSNs; the budget could be the remaining battery power of the robots or the Uber driver's electric car, or the computing power of the agents in many AI/ML applications [25]. BC-TSP is NP-hard as when the budget is unconstrained, it degenerates into the TSP.

Our Contribution. In this paper, we design a suite of algorithms to solve BC-TSP. We first design an Integer Linear Program (ILP) to solve BC-TSP optimally. We then design two more 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). Our main contribution is a multi-agent cooperative reinforcement learning (MARL) algorithm with a provable convergence guarantee. Although PC-TSP has been studied extensively [36], [34], [4], [42], most of the existing research is handcrafted solutions that are not based on RL techniques.

Our novel observation is that RL is uniquely suitable for solving the PC-TSP. First and foremost, the prizes in PC-TSP and the rewards in the RL reward model are intrinsically related and the prizes in PC-TSP could be utilized to build powerful and efficient reward models in the RL. Second, the prize-maximization goal of the traveling salesman in PC-TSP resembles naturally the RL agent's goal of maximizing accumulative discounted rewards, making PC-TSP an ideal problem to be solved by the RL. Third, the nodes in the

BC-TSP can be treated as states in the RL and the traveling salesman moving from one node to another is the same as the agent transitioning from one state to another. When moving from one node to another, the traveling salesman must constantly make decisions to maximize his collected prizes while staying within his budget, which uniquely resembles the Markov decision process adopted in RL.

Despite the above similarities, it remains unclear as to what extent the prize-collecting in BC-TSP corresponds to cumulative reward maximization in RL and how the synergy between them can be further exploited to uncover more powerful and effective RL algorithms. In particular, how to integrate the prizes in PC-TSP into the RL reward model remains largely unexplored. In this paper, we address this question and design a multi-agent cooperative RL framework that integrates the prizes available at nodes into the reward model of the RL. Via extensive simulations under different network and RL parameters, we show that our learning algorithm is solution-effective and time-efficient. In particular, it delivers an average of 65.6% of less traveling distance compared to one handcrafted greedy algorithm. Furthermore, when changing the number of agents m from 1 to 5, our algorithm reduces the prize-collecting learning time by up to 66.9%, demonstrating the effectiveness of multi-agent collaboration in reducing the prize-collecting learning time in BC-TSP.

Unlike the handcrafted greedy algorithms, in MARL, intelligent agents learn cooperatively by interacting with the environment and adjusting their actions accordingly [37]. Therefore, the MARL algorithm is more adaptive and robust in a dynamic network environment and has become an ideal alternative to solve many network-related combinatorial problems time-efficiently [5], [32]. However, it remains unclear as to what extent the prize collection in BC-TSP corresponds to cumulative reward maximization in MARL. In this paper, we endeavor to integrate the prizes in BC-TSP into the RL reward model and design the first MARL algorithm to solve BC-TSP. Via extensive simulations, we show that the MARL algorithm outperforms the two greedy algorithms and performs close to the ILP-based optimal solution (within 12.7% of the optimal).

II. Related Work

In this section, we review all the theory work in PC-TSP, the existing research in robotic sensor networks, and the only existing work that applies RL to solve a graph-theoretical problem that is closely related to PC-TSP.

Existing PC-TSP Research. A few works from the theory and operations research community have studied BC-TSP [36], [34], [4]. In his Ph.D. thesis, Sokkappa [36] systematically studied BC-TSP. He proved the problem is NP-hard, and there is no fully polynomial approximation scheme that exists unless $P = NP$. He then 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 [4] presented a $(4 + \epsilon)$-approximation algorithm to the so-called budget prize collecting tree problem, which finds a subtree with maximum prizes while is the cost of the tree (i.e., the sum of all its edge weights) stays in a budget. Recently, 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. To the best of our knowledge, our work is the first one to apply the RL technique to solve the BC-TSP problem.

Recently, Ruiz et al. [35] studied the prize-collecting traveling salesman problem (PCTSP) using an RL approach. In PCTSP, the goal of the traveling salesman is to find a route from s to t such that the sum of the prizes of all the nodes along the route reaches a preset quota while the distance along the route is minimized. As the budget is not a constraint in PCTSP, it is less challenging than the BC-TSP, wherein the salesman has to constantly check if his remaining budget is sufficient enough for him to return to the destination. This calls for new RL algorithms and reward models to solve the BC-TSP.

With the recent technological breakthrough of artificial intelligence (AI) and machine learning (ML), especially in the area of deep reinforcement learning [25], robotic research and the design and development of robotic applications have entered a new phase [21], [28].

BC-TSP has been studied extensively in the theory community, which is demonstrated by extensive literature [36], [34], [4], to name a few. In his Ph.D. thesis, Sokkappa systematically studied BC-TSP. They proved the problem is NP-hard and there is no fully polynomial approximation scheme exists unless $P = NP$. They proposed branch-and-bound-based heuristics to solve the BC-TSP and also optimal solutions for several special cases.

Research in Robotic Sensor Networks (RSNs). RSNs [22], [26] have drawn lots of attention in recent years, wherein mobile robots are utilized to enhance the system performance of wireless sensor networks. As the PC-TSP is inspired by the data-collecting robots with limited battery power in the RSNs, we briefly review the existing work to motivate the theoretical contribution of our paper to this burgeoning field. Ma et al. [31] introduced mobile data collectors to gather data in large-scale sensor networks. In a single mobile collector case, the goal is to minimize the length of the data-gathering tour; in the multiple-collector case, the goal is to minimize the number of mobile collectors such that each subtour does not exceed some time constraint. They formulate the problems as mixed-integer programs and present heuristic data-gathering algorithms. Guo et al. [23], [40] extended it by considering wireless energy-charging. They formulated a network utility maximization problem considering energy balance and the bounded sojourn time of the mobile collector and designed a distributed algorithm. Salarian et al. introduced the rendezvous points (RPs) in the data-collecting process to form a hybrid moving pattern in which a mobile-sink node only visits RPs instead of all

the nodes. Sensor nodes that are not RPs forward their sensed data via multi-hop to the nearest RP. They designed a weighted rendezvous planning heuristic algorithm that enables a mobile sink to retrieve all sensed data within a given deadline while conserving the energy expenditure of sensor nodes. However, this approach assumes that sensors produce data with the same speed and have no limitation on buffer size. Wang et al. [41] relax these two assumptions and design efficient path planning for a reliable data gathering (EARTH) algorithm. Kim et al. [27], [43] considered multiple-drone-assisted search-and-reconnaissance scenarios and modeled them as the Travelling Salesman Problem with Neighborhood (TSPN). In TSPN, a node is considered as being visited by the traveler once their distance is within a certain threshold value. They designed approximation algorithms to minimize the task completion time and the largest time gap between two consecutive observations over the same point of interest.

Existing PC-TSP Research. Existing PC-TSP works [10], [18], [7] considered that each vertex has a prize to collect and a penalty if not visited; the goal is to minimize the travel costs and penalties while visiting enough cities to collect a prescribed amount of prize money. Bienstock [13] was one of the first to propose a constant ratio (i.e., 2.5) approximation algorithm. It is based on linear programming relaxation using the ellipsoid method. Archer et al. [6] further improved the approximation ratio to $2-\epsilon$ using Lagrangian relaxation. If no penalty is considered, prize-collecting TSP becomes quota-TSP problem [8], [9]. Awerbush [9] proposed an $O(\log^2 R)$ approximation algorithm where R is the quota to collect. It is based on an approximation for the k-minimum-spanning-tree problem, which is finding a tree of the least weight that spans exactly k vertices on a graph. Ausiello et al. [8] studied the online version of the problem. To achieve a rigorous analysis of their performance bounds, all the above works involved complicated procedures (e.g., ellipsoid methods [13], [18] and Lagrangian relaxation [6], [24]), which cannot be easily implemented for emerging large-scale applications such as autonomous driving and automated warehouse.

Existing RL Research for TSP. Our work was inspired by ant-Q [19], an algorithmic framework combining the Q-learning algorithm [37] and ant colony intelligent behavior representing the collective collaboration of a large number of autonomous agents. They showed that ant-Q is an effective RL technique for solving combinatorial optimization problems, including TSP. Recently, Ottoni et al. [33] applied two RL techniques (i.e., Q-learning and SARSA) to solve TSP with refueling where uniform and non-uniform fuel prices are available at different locations. However, they did not consider the prize-collecting TSP, which is the topic of this paper. By integrating prize-collecting in TSP with the reward model in ant-Q, we are able to create a more powerful and efficient MARL algorithm that solves PC-TSP.

In recent years, deep reinforcement learning (DRL) has been increasingly utilized to solve TSP problems [12], [45], [17]. Utilizing neural network-based function approximation algorithms and RL, Bello et al. [12] presented a framework to tackle combinatorial optimization problems. They solved TSP by training a recurrent neural network and optimizing its parameters using a policy gradient method. Zhang et al. [45] used DRL to tackle a variant of TSP with a time window and rejections. In particular, a manager agent learns to assign customers to vehicles via a policy network based on Graph Isomorphism Network [**?**].

DRL is a powerful technique that can handle complex states and decision-making for agents. However, in this paper, we adopted RL instead of DRL to solve PC-TSP for the following reasons. First, by learning from a training set using neural networks to find patterns and make predictions, DRL is both time- and resource-consuming [12]. In contrast, RL does not rely upon such data sets and can dynamically adjust actions based on continuous feedback. Second, as the PC-TSP setup proposed in this paper has a low-dimensional and discrete setting in terms of the agent's states and actions, RL is sufficient to solve the PC-TSP without resorting to the complex neural network construction and computation required in DRL.

The closest work to ours is Gao et al. [20], which applied the ant-Q technique to solve a different and fundamental graph-theoretical problem called the k-stroll problem [11], [14], [38]. Given a weighted graph $G(V,E)$ and two nodes s, $t \in V$, and an integer k, k-stroll problem is to find the shortest path from s to t that visits at least k other nodes in the graph. In k-stroll, different nodes have the same prizes. PC-TSP generalizes it by considering that different nodes have different prizes available; thus, it is more challenging to solve.

III. Budget-Constrained Traveling Salesman Problem (BC-TSP)

A. Problem Formulation.

Given a weighted 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 $w(u,v)$, indicating the travel distance or cost on this edge. Each node $i \in V$ has a weight $p_i \geq 0 \in \mathbb{R}^+$, indicating the prize available at this node. Given any route $R = \{v_1, v_2, ...v_n\}$ in G, where $(v_i, v_{i+1}) \in E$, denote its cost as $C_R = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$ and its total prizes as $P_R = \sum_{i \in R} p_i$. Let $s, t \in V$ be the traveling salesman's source and destination nodes, respectively. Let $\mathcal{B}$ denote his budget, which indicates the distance he can travel before returning to t. The goal of the BC-TSP is to find a *prize-collecting route* $R_s = \{s = v_1, v_2, v_3, ..., v_n = t\}$ such that its total prize P_{R_s} is maximized while its cost $C_{R_s} \leq \mathcal{B}$. When $s = t$, the salesman starts and ends at the same node.

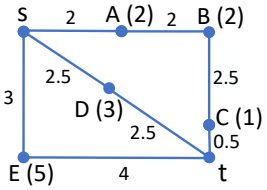


Fig. 1: An example.

EXAMPLE 1: Fig. 1 illustrate BC-TSP with budget $\mathcal{B} = 8$. The numbers on the edges are their weights, and the numbers in the parentheses are the prizes available at nodes. The optimal walk from s to t is s, E, t, C, and t, with a total prize of 6 and a total cost of 8. Other routes are not optimal.

For example, although the path of s, A, B, C, and t is within the budget with a cost of 7, its total prize is 5. □

B. Combinatorial Algorithms for BC-TSP

Below, we design an ILP-based optimal solution and two greedy heuristic algorithms viz. Algo. 1 and 2. We first give the below definition.

Definition 1: (**Budget-Feasible Nodes.**) Given the current node r the traveling salesman is located and his available budget B, the *budget-feasible nodes*, denoted as $\mathcal{F}(r, B)$, is s's unvisited neighbor nodes that the salesman can travel to and then return to destination node t with enough budget. That is, $\mathcal{F}(r, B) = \{u | (r, u) \in E \wedge (w(r, u) + w(u, t) \leq B) \wedge u \in U\}$, where U is the set of unvisited nodes. □

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

$$(A) \quad \max \sum_{i \in V-\{s,t\}} \sum_{j \in V-\{s\}} p_i \cdot x_{i,j} \tag{1}$$

s.t.

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

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

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

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

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

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

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 route starts at node s and ends at node t. 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 there is one global tour visiting all the selected vertices instead of multiple subtours each visiting only a subset of the selected vertices.

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 descending 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). 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 $O(|V|^2)$. Algo. 1 also works for the problem where $s = t$.

Algorithm 1: Greedy Algorithm 1 for BC-TSP.
Input: A complete weighted graph $G(V, E)$, s, t, and initial budget $\mathcal{B}$.
Output: A route R from s to t, 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 $V - \{s, t\}$;
r: the current node where the salesman is located;
B: current available budget, is $\mathcal{B}$ initially;

1. $r = s$, $R = \{s\}$, $C_R = P_R = 0$, $B = \mathcal{B}$, $U = V - \{s, t\} = \{v_1, v_2, ..., v_{|V|-2}\}$;
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 largest prize
4. **while** $(U \neq \phi \wedge \mathcal{F}(r, B) \neq \phi)$
5. **if** $(v_k \in \mathcal{F}(r, B))$
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. $R = R \cup \{t\}$, $C_R = C_R + w(r, t)$, $B = B - w(r, t)$;
14. **RETURN** R, C_R, P_R, B.

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, $pcr(u, v) = \frac{p_v}{w(u,v)}$. Algo. 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 $O(|V|^2)$.

Algorithm 2: Greedy Algorithm 2 for BC-TSP.
Input: A complete weighted graph $G(V, E)$, s, t, and $\mathcal{B}$.
Output: A route R from s to t, its cost C_R and prize P_R.
Notations: R: the current route found, starts from 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 $U = V - \{s, t\}$;

r: the node where the salesman is located currently;
B: current remaining budget, initially $\mathcal{B}$;
1. $r = s$, $R = \{s\}$, $C_R = P_R = 0$, $U = V - \{s, t\}$;
2. $B = \mathcal{B}$;
// if not all nodes are visited, and there are feasible nodes
3. **while** $(U \neq \phi \wedge \mathcal{F}(r, B) \neq \phi)$
4. Let $u = \text{argmax}_{v \in \mathcal{F}(r,B) \cap U} pcr(r, v)$;
5. $R = R \cup \{u\}$;
6. $C_R = C_R + w(r, u)$, $P_R = P_R + p_u$;
7. $B = B - w(r, u)$, $U = U - \{u\}$;
8. $r = u$;
9. **end while;**
10. $R = R \cup \{t\}$, $C_R = C_R + w(r, t)$, $B = B - w(r, t)$;
11. **RETURN** R, C_R, P_R, B.

EXAMPLE 2: In Fig. 1, both Algo. 1 and 2 give the solution of s, E, t, C, and t, with a total cost of 8 and a total prize of 6. □

IV. MARL Algorithm for BC-TSP

In this section, we first present the basics of RL and then our cooperative MARL framework for BC-TSP.

Reinforcement Learning (RL) [37]. We describe an agent's decision-making in an RL system as a Markov decision process (MDP), which is represented by a 4-tuple (S, A, t, r):

- 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.

In MDP, an agent learns an optimal policy that maximizes its accumulated reward. At a specific state $s \in S$, the agent takes action $a \in A$ to transition to state $t(s, a) \in S$ while receiving a reward $r(s, a) \in R$. The agent maintains a *policy* $\pi(s) : S \to A$ that maps its current state $s \in S$ into the desirable action $a \in A$. In the context of the BC-TSP, the states are all the nodes V, and the actions available for an agent at a node are all the edges emanating from this node. 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 in BC-TSP, when an agent at a node takes action (i.e., follows one of its edges), it will surely end up with the node on the other end of the edge.

A widely used class of RL algorithms is value-based [37], [30], which finds the optimal policy based on the value function at each state s, $V_s^\pi = E\{\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) | s_0 = s\}$. The value at each state is the expected value of a discounted future reward sum with the policy π at state s. Here, γ $(1 \leq \gamma \leq 1)$ is the *discounted rate* that determines the importance of future rewards; the larger of the γ, the more important the future rewards. Recall that $r(s, \pi(s))$ is the reward received by the agent at state s by taking action following policy π.

Q-Learning. Q-learning is a family of value-based algorithms [37]. It learns how to optimize the quality of the actions in terms of the Q-value $Q(s, a)$. $Q(s, a)$ is defined as 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 as

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

where $1 \leq \alpha \leq 1$ is the *learning rate* that decides to what extent newly acquired information overrides old information in the learning process. In Eqn. 8, $\max_b Q(t, b)$ is the maximum reward that can be obtained from the next state t.

Multi-agent Reinforcement Learning (MARL) Algorithm. In our MARL framework for BC-TSP, there are multiple agents that all start from the node s. They work synchronously and cooperatively to learn the state-action Q-table and the reward table and take action accordingly. We first introduce the action rules for all the learning agents and then present our MARL algorithm.

Action Rule of Agents. Each agent follows the same *action rule* specifying the next node it moves to during the learning process. It consists of the following three scenarios.

- **Exploitation.** In exploitation, the agent always chooses the node
$$t = \text{argmax}_{u \in U \cap \mathcal{F}(s,B)} \{\frac{[Q(s, u)]^\delta \times p_u}{[w(s, u)]^\beta}\}$$
to move to. Here, U is the set of nodes not visited yet by the agent and $\mathcal{F}(s, B)$ is node s's budget-feasible nodes, and δ and β are preset parameters. That is, an agent, located at node s, always moves to an unvisited budget-feasible node u that maximizes the learned Q-value $Q(s, u)$ weighted by the length $w(s, u)$ and the prize p_u available at node 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 chooses exploration explained below.
- **Exploration.** In exploration, the agent chooses a node $t \in U \cap \mathcal{F}(s, B)$ to move to by the following distribution:
$$p(s, t) = \frac{([Q(s, t)]^\delta \times p_u)/[w(s, t)]^\beta}{\sum_{u \in U \cap \mathcal{F}(s,B)} ([Q(s, u)]^\delta \times p_u)/[w(s, u)]^\beta}.$$
That is, a node $u \in U \cap \mathcal{F}(s, B)$ is selected with probability $p(s, u)$, while $\sum_{u \in U \cap \mathcal{F}(s,B)} p(s, u) = 1$. The distribution $p(s, t)$ characterizes how good the nodes are at learned Q-values, the edge lengths, and the node prizes. The higher the Q-value, the shorter the edge length, and the larger the node prize, the more desirable the node is to move to.
- **Termination.** When $U \cap \mathcal{F}(s, B) = \phi$, which means the agent does not have an unvisited budget-feasible node to move to, the agent goes to destination t and terminates in this episode.

MARL Algorithm. Next, we present our MARL algorithm viz. Algo. 3, which consists of a learning stage for the m agents

(lines 1-32) and an execution stage for the traveling salesman (lines 33-39). The learning stage takes place in a preset number of episodes. Each episode consists of the below two steps.

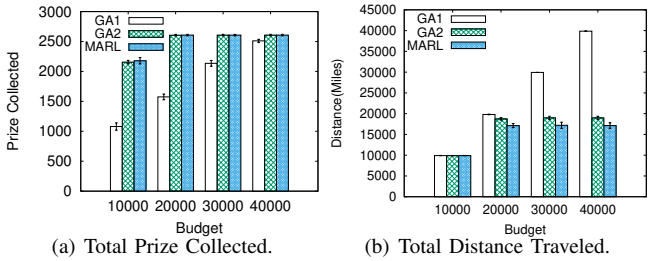
In the first step (lines 3-26), all the m agents are initially located at the starting node s with zero collected prizes. Then each independently follows the action rule to move to the next budget-feasible node to collect prizes and collaboratively updates the Q-value of the involved edges. This takes place in parallel for all the agents. When an agent can no longer find a feasible unvisited node to move to due to its insufficient budget, it terminates and goes to t (lines 8-14); in this case, it must wait for other agents to finish in this episode. Otherwise, it moves to the next node, collects the prize, and continues the prize-collecting process (lines 15-23). In either case, it updates the Q-values of the involved edge. Here, the prizes at each node can be collected multiple times by different agents (as this is the learning stage).

In the second step (lines 27-31), the m agents communicate with each other and find among the m routes the one with the maximum collected prizes. It then updates the reward value and Q-value of the edges of this route.

Finally, in the execution stage (lines 33-38), the traveling salesman starts from s, visits the node with the largest Q-value in the Q-table, and ends at t, collecting the prizes along the way. Note we set the initial Q-value and reward value for edge (u, v) as $\frac{p_u+p_v}{w(u,v)}$ and $\frac{-w(u,v)}{p_v}$, respectively, 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.

Algorithm 3: MARL Algorithm for PC-TSP.
Input: A graph $G(V, E)$, s, t, and a budget $\mathcal{B}$.
Output: A route R from s to t, C_R, and P_R.
Notations: i: index for episodes; j: index for agents;
U_j: set of nodes agent j not yet visits, initially $V - \{s, t\}$;
R_j: the route taken by agent j, initially empty;
B_j: the currently available budget of agent j, initially $\mathcal{B}$;
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 0;
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;
R^m: the route with the maximum prize so far;
P^m: the prize at the maximum-prize-route;
R: the final route found the MARL, initially empty;
$Q(u, v)$: Q-value of edge (u, v), initially $\frac{p_u+p_v}{w(u,v)}$;
$r(u, v)$: Reward of edge (u, v), initially 0;
α: learning rate, $\alpha = 0.1$;
γ: discount factor, $\gamma = 0.3$;
q_0: trade-off between exploration and exploitation, $q_0 = 0.5$;
δ, β: parameters in action rule; $\delta = 1$ and $\beta = 2$;
W: a constant value of 100;
epi: number of episodes in the MARL, $epi = 30000$;
0. $R^m = \phi$ (empty set), $P^m = 0$;
1. **for** $(1 \le i \le epi)$ // Learning stage
2. All the m agents are at node s, $r_j = s, 1 \le j \le m$;
3. **for** $(j = 1; j \le m; j{+}{+})$ // Agent j
4. $P_j = 0$, $B_j = \mathcal{B}_j$, $isDone_j = false$;
end for;
// At least one agent has not finished in this episode
5. **while** $(\exists\ j,\ 1 \le j \le m,\ isDone_j == false)$
6. **for** $(j = 1; j \le m; j{+}{+})$ // Agent j
7. **if** $(isDone_j == false)$ // Agent j has not finished
8. **if** $(U_j \cap \mathcal{F}(r_j, B_j) == \phi)$ // Agent j terminates
9. $isDone_j = true$;
10. $R_j = R_j \cup \{t\}$; // Agent j goes to t
11. $l_j = l_j + w(r_j, t)$, $B_j = B_j - w(r_j, t)$;
12. $Q(r_j, t) = (1 - \alpha) \cdot Q(r_j, t) +$
$\alpha \cdot \gamma \cdot \max_{z \in U_j \cap \mathcal{F}(t, B_j)} Q(t, z)$;
13. $r_j = t$;
14. **end if;**
15. **else**
16. Finds the next node s_j following action rule;
17. $R_j = R_j \cup \{s_j\}$;
18. $l_j = l_j + w(r_j, s_j)$, $B_j = B_j - w(r_j, s_j)$;
19. $P_j = P_j + p_{s_j}$; // Collect prize
20. $Q(r_j, s_j) = (1 - \alpha) \cdot Q(r_j, s_j) +$
$\alpha \cdot \gamma \cdot \max_{z \in U_j \cap \mathcal{F}(s_j, B_j)} Q(s_j, z)$;
21. $r_j = s_j$; // Move to node s_j;
22. $U_j = U_j - \{s_j\}$;
23. **end else;**
24. **end if;**
25. **end for;**
26. **end while;**
27. $j* = \text{argmax}_{1 \le j \le m} P_j$; // Route of largest prize
if $(P_{j*} > P^m)$
$P^m = P_{j*}$, $R^m = R_{j*}$;
28. **for** (each edge $(u, v) \in R_{j*}$)
29. $r(u, v) = r(u, v) + \frac{i \cdot W}{P_{j*}}$; // Reward value $r(u, v)$
30. $Q(u, v) \leftarrow (1 - \alpha) \cdot Q(u, v) +$
$\alpha \cdot [r(u, v) + \gamma \cdot \max_b Q(v, b)]$; // Update Q-value
31. **end for;**
end if;
32. **end for;** // End of each episode in learning stage
// Execution stage
33. $r = s$, $R = \{s\}$, $C_R = 0$, $P_R = 0$, $B = \mathcal{B}$;
34. **while** $(r! = t)$
35. $u = \text{argmax}_b Q(r, b)$;
36. $R = R \cup \{u\}, C_R = C_R + w(r, u), P_R = P_R + p_u$,
$B = B - w(r, u)$;
37. $r = u$;
38. **end while;**
39. **RETURN** R, C_R, P_R, B.

Discussions. There are epi episodes of learning. In each episode, the first step takes at most $m \cdot |V|$, where $|V|$ is the total number of nodes, and the second step takes at most $m + |E|$, where $|E|$ is the total number of edges. Thus the time complexity of Algo. 3 is $O(epi \cdot m \cdot |V|)$.



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

Fig. 2: Comparing MARL, GA1, and GA2.

A. Existing Work.

Wei et al. [42] proposed Deep Reinforcement Learning algorithm to solve the informative path planning problem (IPP), wherein a robot is dispatched into a sensing field to collect the sensing information. Such information, called *mutual information*, measures the informativeness of data (i.e., sensor placement) collected along a path in the field. 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, 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 visit to maximize the collected reward within a budget constraint. However, below we show that when the reward is associated with the nodes, although the RL agent attempts to maximize the accumulated reward in the learning process, it does not necessarily receive the maximum profit, which is the difference between its received reward and its spent cost. We illustrate this fundamental issue of the RL reward model using the below IPP example.

V. Performance Evaluation

Experiment Setup. We write our own simulator in Java on a Windows 10 with AMD Processor (AMD Ryzen 7 3700X 8-Core) and 16GB of DDR4 memory. We refer to the Algo. 1 as **GA1**, Algo. 2 as **GA2**, and the MARL algorithm Algo. 3 as **MARL**. We compare them on traveling salesman tours of 48 state capital cities on the US mainland [3]. 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]. Each data point in our plots is an average of 20 runs with a 95% confidence interval; in each run, a state capital city is randomly chosen as the source and destination city.

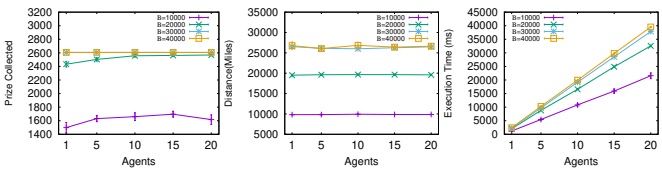
Comparing MARL, GA1, and GA2. Fig. 2 compares all three algorithms by varying the budgets. Fig. 2(a) shows the total prizes collected. We observe that MARL and GA2 outperform GA1, and the performance differences are more prominent at smaller budgets. As GA1 always tries to collect the largest prize available, it could travel long distances, thus exhausting its budget quickly. Fig. 2(b) shows that at smaller budgets, all three algorithms travel the same distances to collect prizes. This is because they all have exhausted their budgets. At larger budgets, MARL yields less distance cost than GA2, which has less cost than GA1. These demonstrate that the MARL algorithm is more efficient (distance-wise) and effective (prize-wise) than the handcrafted greedy algorithms.

Impacts of Number of Agents m on MARL. Next, we study the impact of the number of agents m on the MARL's performance. We vary m from 1, 5, 10, 15, to 20, and the budget $\mathcal{B}$ from 10,000, 20,000, 30,000, to 40,000. Fig. 3(a) shows that for each m, the higher the $\mathcal{B}$, the larger the collected prize. When $\mathcal{B} = 30,000$ and $40,000$, it has collected all the available prizes in the network. However, varying m seems to have no clear effect on the collected prizes. This shows the total collected prize does not depend on m in MARL. Fig. 3(b) shows the traveled distance of the MARL w.r.t. m and $\mathcal{B}$. The higher the $\mathcal{B}$, the more distances it can travel. Again, we observe that varying m does not seem to affect the traveled distance of the salesman. Finally, Fig. 3(c) shows for each $\mathcal{B}$, with the increase of m, the execution time of the MARL algorithm increases. This is because we have a prefixed number of episodes of 25,000, each of which takes more time to execute when more agents join the learning process.

Comparing ILP with MARL. Finally, we compare the ILP-based optimal solution with the MARL. As ILP takes a long time to execute, we focus on smaller cases wherein only 10 state capital cities are randomly chosen for the comparison. Fig. 4 (a) and (b) show their total prizes collected and total distance traveled, respectively, by varying the initial amount of budget. We observe that ILP always performs better than MARL, as ILP is an optimal solution. We also observe that when the budget is small, ILP outperforms MARL by around 33.5%. However, with the increase in the available budget, the performance difference between ILP and MARL gets less and less. At the largest budget of 8000, the performance difference is only 12.7%. This shows that MARL is indeed a competitive algorithm for PC-TSP, especially when there is a sufficient amount of budget.

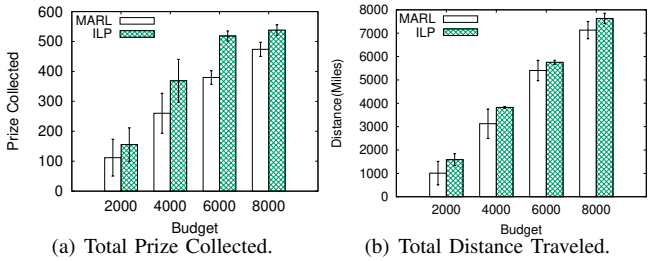
VI. Conclusions and Future Work

We proposed an algorithmic problem called budget-constrained TSP (BC-TSP) that arises from many robotic applications, wherein robots are dispatched to accomplish some tasks with limited battery power. Such applications include



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

Fig. 3: MARL with varying number of agents m.



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

Fig. 4: Comparing ILP and MARL.

robotic sensor networks, electrical cars in ride-sharing, and automated warehouses. We designed two greedy algorithms and a multi-agent reinforcement learning (MARL) algorithm to solve BC-TSP. The MARL algorithm performs better than the handcrafted greedy algorithms in both distance costs and prizes collected and performs close to the ILP-based optimal solution (within 12.7% of the optimal). To the best of our knowledge, our work is the first one to apply the RL technique to solve the BC-TSP problem. As an ongoing and future work, we will study performance guarantees and the convergence of the greedy and MARL algorithms. We will also answer the question below: Do m agents in t episodes and one agent in $m \times t$ episodes have the same learning performance to solve BC-TSP?

As another future direction, we will also compare our MARL approach with deep reinforcement learning (DRL)-based approach [12], [45], [17]. Utilizing neural network-based function approximation algorithms and RL, DRL recently became an effective framework for tackling combinatorial optimization problems [12], [45], [17]. DRL is a powerful technique that can handle complex states and decision-making for agents. In this work, we use MARL instead of DRL as agent learning in neural networks is mainly treated as a black box, while in our work, we have control of all the ins and outs of agent learning. In future work, we will compare our MARL approach and DRL approach in terms of their solution quality and time efficiency.

ACKNOWLEDGMENT

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

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] Traveling salesman tour of us capital cities. https://www.math.uwaterloo.ca/tsp/data/usa/index.html.
[4] A better approximation algorithm for the budget prize collecting tree problem. *Operations Research Letters*, 32(4):316–319, 2004.
[5] Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134, 2021.
[6] A. Archer, M. H. Bateni, M. T. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting steiner tree and tsp. In *IEEE FOCS*, 2009.
[7] C. Archetti, M. G. Speranza, and D. Vigo. Vehicle routing problems with profits. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, page 273–297, 2014.
[8] G. Ausiello, M. Demange, L. Laura, and V. Paschos. Algorithms for the on-line quota traveling salesman problem. *Information Processing Letters*, 92:89–94, 2004.
[9] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.*, 28(1):254–262, February 1999.
[10] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.
[11] M. Bateni and J. Chuzhoy. Approximation algorithms for the directed k-tour and k-stroll problems. In *APPROX/RANDOM 2010*.
[12] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.
[13] D. Bienstock, M. X. Goemans, and D. Simchi-Levi D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.
[14] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *IEEE FOCS 2003*.
[15] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40, may 2021.
[16] Mario Di Francesco, Sajal K. Das, and Giuseppe Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. 8(1), 2011.
[17] O. Dogan and A. Alkaya. A novel method for prize collecting traveling salesman problem with time windows. In *Intelligent and Fuzzy Techniques for Emerging Conditions and Digital Transformation*. Springer International Publishing, 2022.
[18] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188 – 205, 2005.
[19] L. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *ICML*, 1995.
[20] L. Gao, Y. Chen, and B. Tang. Service function chain placement in cloud data center networks: a cooperative multi-agent reinforcement learning approach. In *the 11th EAI International Conference on Game Theory for Networks (GameNets 2021)*.
[21] 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.
[22] Y. Gu, F. Ren, Y. Ji, and J. Li. The evolution of sink mobility management in wireless sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 18(1):507–524, 2016.
[23] 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.
[24] M. Haouari, S.B. Layeb, and H.D. Sherali. The prize collecting steiner tree problem: models and lagrangian dual optimization approaches. *Comput Optim Appl*, 40:13–39, 2008.
[25] J. Hua, L. Zeng, G. Li, and Z. Ju. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21(4), 2021.
[26] 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.
[27] 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.
[28] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. 32(11), 2013.
[29] Gilbert Laporte. The traveling salesman problem: An overview of the exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1992.
[30] Michael L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.
[31] 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.
[32] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *CoRR*, 2020.

[33] A.L.C. Ottoni, E.G. Nepomuceno, and M.S.d. Oliveira et al. Reinforcement learning for the traveling salesman problem with refueling. *Complex Intell. Syst.*, 8:2001–2015, 2022.

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

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

[38] V. Tran, J. Sun, B. Tang, and D. Pan. Traffic-optimal virtual network function placement and migration in dynamic cloud data centers. In *IEEE IPDPS 2022*.

[39] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 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 Transactions on Mobile Computing*, 18(7):1527–1540, 2019.

[42] Y. Wei and R. Zheng. Informative path planning for mobile sensing with reinforcement learning. In *IEEE INFOCOM 2020*, 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] Y. Yang and M. Zhao. Optimization-based distributed algorithms for mobile data gathering in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(10):1464–1477, 2012.

[45] 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.