Prize-Collecting Traveling Salesman Problem: a Reinforcement Learning Approach

Justin Ruiz*, Christopher Gonzalez*, Yutian Chen†, Bin Tang*
*Department of Computer Science, California State University Dominguez Hills
jruiz266@toromail.csudh.edu, chris.a.gonzalez97@gmail.com, btang@csudh.edu
†Economics Department, California State University, Long Beach, Yutian.Chen@csulb.edu

***Abstract*—Prize-Collecting Traveling Salesman Problem (PC-TSP) is a new variation of TSP and is defined as follows. Given a weighted complete graph $G(V, E)$ where node $i \in V$ has an available prize of p_i, and two nodes $s, t \in V$, 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 visited along the route reaches a pre-set quota while the distance along the route is minimized. In this paper, we propose a multi-agent reinforcement learning (MARL) framework for the PC-TSP. Our novel observation is that prize-collecting in PC-TSP is intrinsically related to cumulative reward maximization in reinforcement learning (RL). By integrating the prizes in PC-TSP into the reward model in RL, we design an efficient and effective MARL algorithm to solve the PC-TSP. Via extensive simulations under different network and RL parameters, we show that our learning algorithm delivers an average of 65.6% of less traveling distance compared to one existing handcrafted greedy algorithm. 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 PC-TSP.**

Keywords – Prize-collecting Traveling Salesman Problem, Multi-Agent Reinforcement Learning

I. Introduction

Background and Motivation. Traveling salesman problem (TSP) is one of the most famous combinatorial optimization problems in Computer Science. Given a weighted graph, TSP finds the shortest route for the salesman who starts in a city and visits all other cities, and finally comes back. Since it was first formulated mathematically in the 1930s, TSP has found numerous applications including planning and logistics, microchip manufacture, robotics, and DNA sequencing [15].

In this paper, we focus on a special category of TSP called prize-collecting TSP (PC-TSP). In contrast to the traditional TSP wherein all the nodes must be visited, in PC-TSP, each node has some amount of prize to be collected; the goal of the traveling salesman is to find a subset of the nodes to visit to collect some targeted amount of prizes. The targeted prize is called the *quota* for the salesman. PC-TSP is motivated by some of the latest business innovations and technological development in recent years. Below we give some motivating examples for PC-TSP.

Motivating Examples. Consider a Uber driver in autonomous driving, who starts from his home and drives to different locations to pick up and drop off customers and returns to his home. Given a sequence of customer ride requests (at different locations) offering different amounts of payments, one possible goal of the driver could be how to achieve a targeted amount of ride payment (i.e., quota) while minimizing his traveling distance to save time and gas. For example, a typical Uber driver could set a daily goal, say $1000, and stop working once that goal is achieved. Similarly, in an automated warehouse scenario, the time window delivery option of the merchandise (e.g., same-day priority delivery with high fees vs. 7-day free delivery) plays a central role in multi-robots scheduling of how to ship out merchandise within their time windows cost-effectively [22], [13].

Our Contribution. In this paper, we design a multi-agent cooperative reinforcement learning (MARL)-based algorithm to solve the PC-TSP. Unlike traditional computer algorithms, which are handcrafted by humans, in RL, an intelligent agent learns by iteratively interacting with the environment and adjusting their actions accordingly [26]. Therefore RL algorithm is more adaptive and robust in a dynamic network environment. In addition, as many network-related combinatorial problems, including PC-TSP, are NP-hard and thus take a prohibitively amount of time to solve optimally, RL becomes an ideal alternative to solve these problems time-efficiently. As such, RL is well utilized to solve network-related combinatorial optimization problems [4], [24].

Although PC-TSP has been studied extensively [6], [17], [9], none of the existing research utilizes RL techniques to solve the PC-TSP to the extent of our knowledge. We observe that two characteristics of the PC-TSP make RL a particularly good candidate to solve the problem. First, in the PC-TSP, to find the shortest route while collecting enough prizes, the traveling salesman must constantly make decisions along the way. Such sequential decision-making resembles the Markov decision process adopted in RL, wherein the outcomes are partly random and partly under the control of a decision-maker. Second, the goal of the traveling salesman in PC-TSP is to collect enough prizes while visiting different cities using the least cost route. This resembles the goal of RL, which is for the agent to learn an optimal policy that maximizes accumulative discounted rewards received at different states.

Despite the above similarities, it remains unclear as to what extent the prize-collecting in PC-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 PC-TSP.

Paper Organization. The rest of the paper is organized as follows. Section II reviews the related work that solves PC-TSP. Section III formulates the PC-TSP and proposes an optimal algorithm and a greedy heuristic algorithm. In Section IV we propose our MARL algorithm. Section V compares all the algorithms and discusses the results. Section VI concludes the paper with a discussion of future works.

II. Related Work

In this section, we review the related work to illustrate the contributions of our work. We first review the existing PC-TSP research from the theory community and then review the existing works that use RL to solve TSP.

Existing PC-TSP Research. Existing PC-TSP works [9], [17], [6] 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 [12] 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. [5] further improved the approximation ratio to $2-\epsilon$ using Lagrangian relaxation. If no penalty is considered, prize-collecting TSP becomes quota-TSP problem [7], [8]. Awerbush [8] 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. [7] 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 [12], [17] and Lagrangian relaxation [5], [21]), 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 [18], an algorithmic framework combining the Q-learning algorithm [26] 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. [25] 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 [11], [29], [16]. Utilizing neural network-based function approximation algorithms and RL, Bello et al. [11] 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. [29] 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 [28].

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 [11]. 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. [19], which applied the ant-Q technique to solve a different and fundamental graph-theoretical problem called the k-stroll problem [10], [14], [27]. 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. Prize-Collecting Traveling Salesman Problem (PC-TSP)

A. Problem Formulation.

Given a complete and 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 the graph, 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 $P_R = \sum_{i \in R} p_i$. Note that if all the nodes on the path are distinct, this route is a path; otherwise, it is a walk. Let

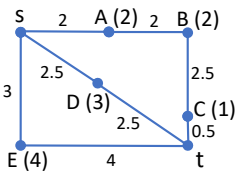


Fig. 1: Illustrating the PC-TSP.

TABLE I: Notation Summary

Notation	Description
$G(V, E)$	A complete graph with $\|V\|$ nodes and $\|E\|$ edges
$w(u, v)$	Weight of an edge $(u, v) \in E$
p_i	Prize available at node $i \in V$
s	The starting node of the traveling salesman
t	The ending node of the traveling salesman
$\mathcal{Q}$	The targeted quota by the traveling salesman
m	The number of agents
P_j	The total prizes collected by agent i, $0 \leq j \leq m$
α	The learning rate of each agent, $0 \leq \alpha \leq 1$
γ	The discount rate of each agent, $0 \leq \gamma \leq 1$
δ, β	Parameters weighing the relative importance of the Q-value and the edge length in the agent's action selection rule

$s, t \in V$ be the two nodes where a traveling salesman starts and finishes his travel, and let $\mathcal{Q}$ denote his targeted quota to collect during his trip. The goal of the PC-TSP is to find a route $R_s = \{s = v_1, v_2, v_3, ..., v_n = t\}$ such that its total prize $P_{R_s} \geq \mathcal{Q}$ while its cost C_{R_s} is minimized. Note that the traditional prize-collecting TSP problem with $s = t$ is a special case of the PC-TSP studied in this paper.

EXAMPLE 1: Fig. 1 is an illustrative example for PC-TSP for quota $\mathcal{Q} = 4$. The numbers on the edges are their weights, and the numbers in the parentheses are the prizes available at different nodes. The optimal walk from s to t is: s, D, t, C, and t, with a total cost of 6 and a total collected prize of 4. Other routes from s to t are not optimal; e.g., the path s, A, B, C, and t has a cost of 7 and a collected total prize of 5. □

B. Combinatorial Algorithms for PC-TSP

Below we solve PC-TSP by presenting two greedy heuristic algorithms viz. Algo. 1 and 2.

Greedy Algorithm 1. Algo. 1 is a greedy algorithm that takes place in rounds. In each round, it visits an unvisited node with the maximum prize among all the unvisited nodes. It continues until the total amount of collected prizes reaches $\mathcal{Q}$. Its time complexity is $O(|V|^2)$. Note that Algo. 1 also works for PC-TSP problem where $s = t$.

Algorithm 1: Greedy Algorithm 1 for PC-TSP.
Input: A complete weighted graph $G(V, E)$, s, t, and $\mathcal{Q}$.
Output: A route R from s to t, its cost C_R and prize P_R.
Notations: R: the current route found, initially empty;
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;
u: the node where salesman moves to next;
1. $r = s$, $R = \phi$, $C_R = P_R = 0$, $U = V - \{s, t\}$;
2. **while** $(P_R < \mathcal{Q})$
3. Let $u = \text{argmax}_{z \in U} p_z$;
4. $R = R \cup \{u\}$, $U = U - \{u\}$;
5. $C_R = C_R + w(r, u)$, $P_R = P_R + p_u$;
6. $r = u$;
7. **end while;**
8. **RETURN** R, C_R, and P_R.

Greedy Algorithm 2. Algo. 2 is similar as Algo. 1, except that in each round, it visits an unvisited node with the maximum prize cost ratio defined next. Given an edge $(u, v) \in E$, and the traveling salesman is at node u, we define the *prize cost ratio* of going to v as the ratio between the prize available at v and the edge weight $w(u, v)$, and denote it as $pcr(u, v)$. That is $pcr(u, v) = \frac{p_v}{w(u,v)}$.

Algorithm 2: Greedy Algorithm 2 for PC-TSP.
Input: A complete weighted graph $G(V, E)$, s, t, and $\mathcal{Q}$.
Output: A route R from s to t, its cost C_R and prize P_R.
Notations: R: the current route found, initially empty;
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;
u: the node where salesman moves to next;
1. $r = s$, $R = \phi$, $C_R = P_R = 0$, $U = V - \{s, t\}$;
2. **while** $(P_R < \mathcal{Q})$
3. Let $u = \text{argmax}_{z \in U} pcr(r, z) = \text{argmax}_{z \in U} \frac{p_z}{w(r,z)}$;
4. $R = R \cup \{u\}$, $U = U - \{u\}$;
5. $C_R = C_R + w(r, u)$, $P_R = P_R + p_u$;
6. $r = u$;
7. **end while;**
8. **RETURN** R, C_R, and P_R.

EXAMPLE 2: In Fig. 1, both Algo. 1 and 2 give the solution of s, E, and t, with a total cost of 7 and a total prize of 4. □

IV. Multi-agent Reinforcement Learning (MARL) for PC-TSP

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

Reinforcement Learning (RL) [26]. 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 PC-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 PC-TSP well, as in PC-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 [26], [23], 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 [26]. 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

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

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. 1, $\max_b Q(t, b)$ is the maximum reward that can be obtained from the next state t.

Multi-agent Reinforcement Learning (MARL) Algorithm. In our MARL framework for PC-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 in any of the states. The common goal of all the agents is to learn and find a route starting from s, each visiting some nodes to collect enough prizes that equal to or a bit larger than $\mathcal{Q}$, and ending at t.

Node Selection in PC-TSP. For an agent located at any node s, the *node selection rule* specifies the next node t it moves to during its prize-collecting learning process. It combines the *exploration*, wherein an agent improves its knowledge about each action by exploring new actions, and *exploitation*, where an agent exploits its current estimated value and chooses the greedy approach to get the most reward.

In particular, in exploitation, the agent always chooses the node $t = \text{argmax}_{u \in U}\{\frac{[Q(s,u)]^\delta \times p_u}{[w(s,u)]^\beta}\}$ to move to. Here, U is the set of nodes not visited yet by the agent, and δ and β are preset parameters. That is, an agent, located at node s, always moves to a node t that maximizes the learned Q-value $Q(s, t)$ weighted by the length $w(s, t)$ of the edge (s, t) and the prize p_t available at node t. In exploration, the agent chooses a node $t \in U$ to move to by the following distribution: $p(s,t) = \frac{([Q(s,t)]^\delta \times p_u)/[w(s,t)]^\beta}{\sum_{u \in U}([Q(s,u)]^\delta \times p_u)/[w(s,u)]^\beta}$. When $q \leq q_0$, 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, exploration is selected. 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.

The above node selection rule is based on ϵ-greedy exploration [26], wherein an agent selects a random action with probability ϵ and selects the best action, which corresponds to the highest Q-value, with probability $1-\epsilon$. It was also applied to solve a virtual network function placement problem [19]. However, our node selection rule augments it by considering node weights (i.e., prizes).

MARL Algorithm. Next, we present our MARL algorithm viz. Algo. 3, which consists of a learning stage (lines 1-29) and an execution stage (lines 30-36). There are m agents in the learning stage, which takes place in a preset number of episodes. Each episode consists of the below two steps.

In the first step (lines 3-23), all the m agents are initially located at the starting node s with zero collected prizes. Then each independently follows the node selection rule to move to the next node to collect prizes and collaboratively updates the Q-value of the involved edge. This continues in parallel among all the agents until they each collect enough prizes $\mathcal{Q}$. In this process, some agents may finish collecting prizes earlier than others; in this case, they must wait for others to collect their prizes (lines 5-17). Here we assume the prizes at each node can be collected multiple times. Once all the agents collect enough prizes, they arrive at the destination t and update the Q-table one more time (lines 18-23).

In the second step (lines 24-28), it finds among the m routes the one with the shortest distance and updates the reward value and Q-value of the edges that belong to this shortest route.

Finally, in the execution stage (lines 30-36), the traveling salesman starts from s and ends at t while traveling to the next node that maximizes the Q-value from that node. It collects the prizes at each visited node. 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 on edge, the more valuable of the edge for the salesman to travel.

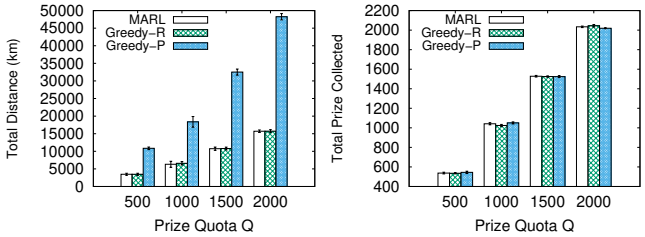
Algorithm 3: MARL Algorithm for PC-TSP.
Input: A graph $G(V, E)$, s, t, and a quota $\mathcal{Q}$.
Output: A route R from s to t, C_R, and P_R.
Notations: i: index for nodes; j: index for agents;
U_j: set of nodes agent j not yet visits, initially $V - \{s, t\}$;
R_j: the route taken by agent j, initially empty;
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;
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 $\frac{-w(u,v)}{p_v}$;
α: learning rate, $\alpha = 0.1$;
γ: discount factor, $\gamma = 0.3$;
δ, β: parameters in node selection rule; $\delta = 1$ and $\beta = 2$;
W: a constant value of 10;
epi: number of episodes in the MARL;
1. **for** ($1 \leq k \leq epi$) // Learning stage
2. All the m agents are at node s, i.e., $r_j = s, 1 \leq j \leq m$;

3. **for** $(j = 1; j \leq m; j{+}{+})$ // Agent j
4. $P_j = 0$; // Initial prize collected by agent j is zero
end for;
// At least one agent has not collected enough prizes
5. **while** $(\min\{P_j | 1 \leq j \leq m\} < \mathcal{Q})$
6. **for** $(j = 1; j \leq m; j{+}{+})$ // Agent j
7. **if** $(P_j < \mathcal{Q})$ // Has not collected enough prize
8. Finds the next node s_j following action rule;
9. $R_j = R_j \cup \{s_j\}$;
10. $l_j = l_j + w(r_j, s_j)$;
11. $P_j = P_j + p_{s_j}$; // Collect prize
12. $Q(r_j, s_j) = (1-\alpha) \cdot Q(r_j, s_j) +$
$\alpha \cdot \gamma \cdot \max_{z \in U_j} Q(s_j, z)$; // Update Q-value
13. $r_j = s_j$; // Move to node s_j;
14. $U_j = U_j - \{s_j\}$;
15. **end if;**
16. **end for;**
17. **end while;**
18. **for** $(j = 1; j \leq m; j{+}{+})$ // Agent j ends at node t
19. $R_j = R_j \cup \{t\}$;
20. $l_j = l_j + w(r_j, t)$;
21. $Q(r_j, s_j) = (1-\alpha) \cdot Q(r_j, s_j) +$
$\alpha \cdot \gamma \cdot \max_{z \in U_j} Q(s_j, z)$; // Update Q-value
22. $r_j = t$;
23. **end for;**
24. $j* = \text{argmin}_{1 \leq j \leq m} l_j$; // Route of smallest cost
25. **for** (each edge $(u, v) \in R_{j*}$)
26. $r(u, v) = r(u, v) + \frac{W}{l_{j*}}$; // Reward value $r(u, v)$
27. $Q(u, v) \leftarrow (1-\alpha) \cdot Q(u, v) +$
$\alpha \cdot [r(u, v) + \gamma \cdot \max_b Q(v, b)]$; // Update Q-value
28. **end for;**
29. **end for;** // End of each episode in learning stage
30. $r = s$, $R = \phi$; // Execution stage
31. **while** $(r! = t)$
32. $u = \text{argmax}_b Q(r, b)$;
33. $R = R \cup \{u\}, C_R = C_R + w(r, u), P_R = P_R + p_u$;
34. $r = u$;
35. **end while;**
36. **RETURN** R, C_R, and P_R.

Discussions. We leave the convergence study of Algo. 3 as a future work. As the first step, we will study if Algo. 3 can find the optimal prize-collecting route when we increase the number of episodes in learning. 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 $O(epi \cdot m \cdot |V|)$.

V. Performance Evaluation

Experiment Setup. Algo. 1 as **Greedy-P**, which visits the node with the largest prize until the quota is collected, and Algo. 2 as **Greedy-R**, as it visits the node with the largest prize-distance ratio in each round. For comparison, and We refer to our multi-agent reinforcement learning algorithm Algo. 3 as **MARL**.



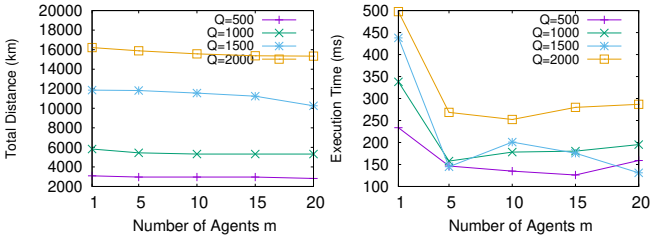
(a) Total Travel Distance. (b) Total Collected Prize.

Fig. 2: Comparing MARL, Greedy-P, and Greedy-R.

We compare our designed algorithms on traveling salesman tours of US capital cities [3]. The prize at each city is a random number in [1, 100]. Given the latitude and longitude of each city, we can find the distance between any pair of capital cities using Haversine formula [1] and construct a complete graph of them. We write our own simulator in Java on a Windows 10 with AMD Processor (AMD Ryzen 7 5800X 8-Core) and 16GB of memory. In all the plots, each data point is an average of 20 runs with a 95% confidence interval. The values of the MARL-related parameters can be found in Algo. 3.

Comparing MARL, Greedy-P, and Greedy-R. Fig. 2 compares all three algorithms by varying the prize quota $\mathcal{Q}$ to be collected. Fig. 2(a) shows the total prize-collecting distance yielded by all three algorithms. We observe that MARL performs much better than Greedy-P, with an average of 65.6% of less traveling distance. We also observe MARL and Greedy-R perform very close to each other. This shows that the MARL is a competitive learning algorithm compared to handcrafted greedy algorithms. Fig. 2(b) shows the actual prizes collected by each algorithm, where all the algorithms collect a bit over the required prize quota $\mathcal{Q}$.

Impacts of Number of Agents m on MARL. Next, we study the impacts of the number of agents m on the performance of the MARL, by varying m from 1, 5, 10, 15, to 20. Fig. 3(a) shows the total distances corresponding to different prize quotas $\mathcal{Q} = 500, 1000, 1500, 2000$. It shows that the higher the $\mathcal{Q}$, the longer distance of the prize-collecting route. However, for each fixed $\mathcal{Q}$, varying m does not have much effect on the resultant total distance of the MARL. This shows the distance only depends on $\mathcal{Q}$ in our MARL algorithm. Fig. 3(b) shows the execution time of the MARL w.r.t. m and $\mathcal{Q}$. We observe that m significantly affects the time needed for the prize-collecting learning process. For a fixed $\mathcal{Q}$, with increasing of m from 1 to 5, the execution time of the MARL algorithm decreases dramatically. In particular, Table II shows the percentage of learning time reduction for this transition could be as much as 66.9%. This shows that cooperation among agents is very effective in reducing the prize-collecting learning time in PC-TSP. However, we also observe that when further increasing m to 10, the execution time of the MARL algorithm gets flattened, and even increases when increasing



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

Fig. 3: Performance of the MARL by varying number of agents m.

m to 15 and 20. This can be explained as follows. When the number of agents gets larger, the agents who finish prize-collecting early must wait for other agents to complete their prize-collecting; this likelihood increases when more agents are learning simultaneously. How to find an optimal number of agents that minimizes the learning time becomes an interesting question for us to explore in future work.

TABLE II: The percentage of learning time reduction when changing the number of agents m from 1 to 5.

Prize Quota $\mathcal{Q}$	500	1000	1500	2000
Learning Time Reduction (%)	37.3	53.3	66.9	46.0

VI. Conclusions and Future Work

Prize-Collecting Traveling Salesman Problem (PC-TSP) has recently drawn attention from the research community as it can be used to model emerging business applications, including autonomous driving and automated warehouse. In this paper, we propose a multi-agent reinforcement learning (MARL) framework for the PC-TSP. We observe that prize-collecting in PC-TSP is intrinsically related to cumulative reward maximization in reinforcement learning. We design an efficient and effective MARL algorithm to solve the PC-TSP. Our solution is not only comparable to or outperforms existing handcrafted greedy algorithms, but also demonstrates the effectiveness of multi-agent collaboration in reducing the Q-learning time in PC-TSP. In future work, we will study the convergence of our algorithm. We will also investigate how to find an optimal number of agents that minimize the prize-collecting learning time in PC-TSP. Although TSP has been studied extensively using the MARL approach, PC-TSP has not to the extent of our knowledge. As PC-TSP is a fundamental problem, the MARL techniques developed in this paper could possibly apply to any applications where prize-collecting is relevant.

Acknowledgment

This work was supported by NSF Grants CNS-2240517, CNS-2131309, INCLUDES-2137791, and the Google exploreCSR program.

References

[1] Haversine formula. https://en.wikipedia.org/wiki/Haversine_formula.
[2] Ibm cplex optimizer. https://www.ibm.com/analytics/cplex-optimizer.
[3] Traveling salesman tour of us capital cities. https://www.math.uwaterloo.ca/tsp/data/usa/index.html.
[4] Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134, 2021.
[5] 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.
[6] 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.
[7] 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.
[8] 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.
[9] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.
[10] M. Bateni and J. Chuzhoy. Approximation algorithms for the directed k-tour and k-stroll problems. In *APPROX/RANDOM 2010*.
[11] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.
[12] 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.
[13] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.
[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] 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.
[17] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188 – 205, 2005.
[18] L. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *ICML*, 1995.
[19] 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)*.
[20] W.J. Gutjahr. A graph-based ant system and its convergence. *Future Generation Computer Systems*, 16:873–888, 2000.
[21] 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.
[22] M. P. Li, P. Sankaran, M. E. Kuhl, R. Ptucha, A. Ganguly, and A. Kwasinski. Task selection by autonomous mobile robots in a warehouse using deep reinforcement learning. In *Winter Simulation Conference*, 2019.
[23] Michael L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.
[24] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *CoRR*, 2020.
[25] 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.
[26] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. The MIT Press, 2020.
[27] 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*.
[28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.
[29] 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.