Budget-Constrained Traveling Salesman Problem: a Reinforcement Learning Approach

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

***Abstract*—In many robotic applications, robots are dispatched to accomplish tasks such as search and rescue and data collection. As robots are mainly powered by batteries, one critical goal for the untethered robot is to accomplish as many tasks as possible and then return to the charging station before its battery is depleted. Inspired by such robotic applications, we study 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 a route from s to t to maximize his collected prizes while keeping his travel cost within the budget. We design two greedy algorithms and a multi-agent reinforcement learning (MARL) algorithm to solve BC-TSP. Via extensive simulations, we show that the MARL algorithm outperforms the two greedy algorithms in most cases, demonstrating that MARL is a practical algorithm for solving BC-TSP.**

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

I. Introduction

With the recent technological breakthrough of artificial intelligence (AI) and machine learning (ML), especially in the area of deep reinforcement learning [9], robotic research and the design and development of robotic applications have entered a new phase [8], [11]. For example, tremendous research efforts have been devoted to developing robotic sensor networks (RSNs), where mobile robots are utilized to enhance the system performance of wireless sensor networks in environment monitoring, intrusion detection, and search and rescue [10]. Below, we give a motivating example.

A Motivating Example. Consider sensor networks for monitoring seismic activity [14] or deep water exploration [6], [15]. In these challenging environments, as it is not feasible to install a base station with a power outlet near the sensing field to collect data, autonomous underwater vehicles (AUVs) [4] and robots [18] are usually dispatched to the field to collect the data. As AUVs and robots are mainly powered by batteries, there is a limit on the time and distance that they can travel before returning to the charging depot. To optimize the performance of such RSN applications, scheduling the AUV or robot to collect as much data as possible within its battery power is critical and challenging.

To address the above challenge, we identify, formulate, and solve a new variation of the traveling salesman problem (TSP) called the budget-constrained traveling salesman problem (BC-TSP). TSP is one of the most famous combinatorial optimization problems in engineering, operation research, and computer science. With numerous applications, including DNA sequencing, chip design, and robotics [5], TSP studies how a traveling salesman can start from a source city, visit all other cities most efficiently, and return to the destination city. In contrast, in BC-TSP, we assume that each node has some amount of prize to be collected, and the goal for the salesman is to find a subset of the nodes to visit to maximize the collected prizes with the given budget. Here, the *budget* is a resource constraint upon the traveling salesman in any network applications modeled as BC-TSP. Therefore, the budget is application-specific; it could be the robots' battery power in the above-mentioned sensing scenario, the remaining gas level of the salesman's car, or the computing power of the agents in many of the AI/ML applications.

In this paper, we design a suite of efficient algorithms to solve BC-TSP. We first design two greedy heuristics in which the salesman iteratively makes decisions. In each iteration, it attempts to visit an unvisited node with the maximum available prize or the maximum prize-cost ratio. We then design a multi-agent cooperative reinforcement learning (MARL)-based algorithm to solve the BC-TSP. Unlike the handcrafted greedy algorithms, in MARL, intelligent agents learn cooperatively by interacting with the environment and adjusting their actions accordingly [17]. Therefore, the MARL algorithm is more adaptive and robust in a dynamic network environment. Via extensive simulations, we show that the MARL algorithm outperforms the two greedy algorithms in most cases.

Our previous work studied the prize-collecting traveling salesman problem (PCTSP) using a reinforcement learning approach [16]. 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, in which the salesman has to constantly check if he has enough budget left to return to the destination city.

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

A. Problem Formulation.

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 source node of the traveling salesman
t	The destination node of the traveling salesman
$\mathcal{B}$	The total budget of 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

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 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 must start and finish at the same node.

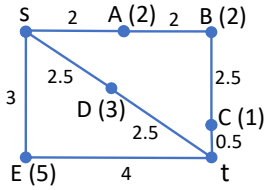


Fig. 1: An example.

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. Greedy Algorithms for BC-TSP

Below, we design 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. □

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 still exists 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. $B = B - w(r, v_k)$, $U = U - \{u\}$;
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, is $\mathcal{B}$ initially;
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.

III. 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) [17]. 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 [17], [12], 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 [17]. 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 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 and 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 $q \leq q_0$, 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, 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 an agent is located at node s and $U \cap \mathcal{F}(s, B) = \phi$, it does not have an unvisited budget-feasible node. In this case, 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

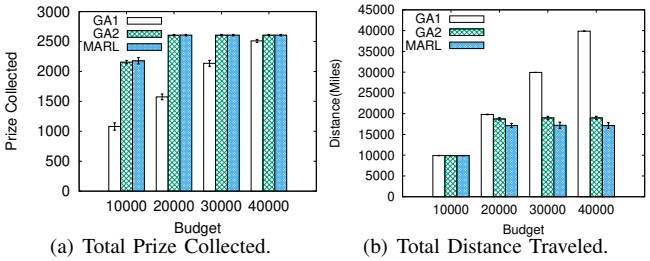
parallel for all the agents. When an agent can no longer find a feasible unvisited node to move to due to its insufficient budget, it terminates and goes to t (lines 8-14); in this case, it must wait for other agents to finish in this episode. Otherwise, it moves to the next node, collects the prize, and continues the prize-collecting process (lines 15-23). In either case, it updates the Q-values of the involved edge. Here, we assume the prizes at each node can be collected multiple times (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: 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, $\gamma = 0.3$;
q_0: trade-off between exploration and exploitation, $q_0 = 0.5$;
δ, β: parameters in node selection rule; $\delta = 1$ and $\beta = 2$;
W: a constant value of 100;
epi: number of episodes in the MARL, $epi = 30000$;

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
28. **for** (each edge $(u, v) \in R_{j*}$)
29. $r(u, v) = r(u, v) + \frac{W}{P_{j*}}$; // 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;**
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.



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

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

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 $O(epi \cdot m \cdot |V|)$.

IV. Performance Evaluation

Experiment Setup. We write our own simulator in Java on a Windows 10 with AMD Processor (AMD Ryzen 7 3700X

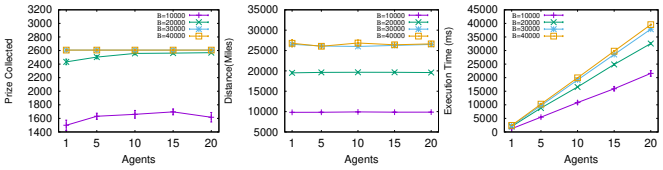
8-Core) and 16GB of DDR4 memory. We refer to the Algo. 1 as **GA1**, Algo. 2 as **GA2**, and the MARL algorithm Algo. 3 as **MARL**. We compare them on traveling salesman tours of 48 state capital cities on the US mainland [2]. Given the latitude and longitude of each city, the distance between any two 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.

V. 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 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. 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:



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

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

Do m agents in t episodes and one agent in $m \times t$ episodes have the same learning performance to solve BC-TSP?

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] Traveling salesman tour of us capital cities. https://www.math.uwaterloo.ca/tsp/data/usa/index.html.
[3] Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134, 2021.
[4] S. Basagni, L. Boloni, P. Gjanci, C. Petrioli, C. A. Phillips, and D. Turgut. Maximizing the value of sensed information in underwater wireless sensor networks via an autonomous underwater vehicle. In *Proc. of INFOCOM*, 2014.
[5] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40, may 2021.
[6] R. W. L. Coutinho, A. Boukerche, and S. Guercin. Performance evaluation of candidate set selection procedures in underwater sensor networks. In *Proc. of IEEE ICC 2019*.
[7] 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)*.
[8] 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.
[9] J. Hua, L. Zeng, G. Li, and Z. Ju. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21(4), 2021.
[10] 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.
[11] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. 32(11), 2013.
[12] Michael L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.
[13] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *CoRR*, 2020.
[14] D. E. Phillips, M. Moazzami, G. Xing, and J. M. Lees. A sensor network for real-time volcano tomography: System design and deployment. In *Proc. of IEEE ICCCN 2017*.
[15] M. Rahmati and D. Pompili. Uwsvc: Scalable video coding transmission for in-network underwater imagery analysis. In *Proc. of IEEE MASS 2019*.
[16] J. Ruiz, C. Gonzalez, Y. Chen, and B. Tang. Prize-collecting traveling salesman problem: a reinforcement learning approach. In *Proc. of IEEE ICC*, 2023.
[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. The MIT Press, 2020.
[18] A. Wichmann, T. Korkmaz, and A. S. Tosun. Robot control strategies for task allocation with connectivity constraints in wireless sensor and robot networks. *IEEE Transactions on Mobile Computing*, 17(6):1429–1441, 2018.