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

Zari Magnaye[1], Jessica Gonzalez[1], Aaron Na Jee Malone[1], Yutian Chen[2], Bin Tang[1]

[1] Department of Computer Science, California State University Dominguez Hills
[2] Economics Department, California State University, Long Beach

Abstract

The Traveling Salesman Problem (TSP) is one of the most well-known combinatorial optimization problems in computer science and engineering. Inspired by a few robotic applications, this paper studies 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 salesman 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. We use data on the 48 state capital cities on the US mainland to show that the MARL algorithm collects more prizes than two handcrafted greedy algorithms while traveling less distances. This demonstrates that MARL is an effective and efficient algorithm for solving BC-TSP. To our knowledge, our work is the first to apply the MARL technique to solve the BC-TSP problem.

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

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most well-known combinatorial problems in computer science and engineering [6, 10]. This paper studies a new variation of the TSP called the Budget-Constrained Traveling Salesman Problem (BC-TSP). BC-TSP is inspired by many emerging robotic applications, wherein 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.

In this paper, 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 computer science. With numerous applications, including DNA sequencing, chip design, and robotics [6], TSP studies how a traveling salesman can start from a source city, visit all other cities, and return to the destination city efficiently. In contrast, in BC-TSP, we assume that the traveling salesman has a budget and each node has some amount of prize to be collected. 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 and is application-specific. It could be the robots' battery power in the RSN scenario mentioned above or the computing power of the agents in many of the AI/ML applications [9].

In this paper, we design a suite of algorithms to solve BC-TSP. We first design two intuitive greedy heuristics, wherein the salesman iteratively decides to visit a node with the maximum available prize or the maximum prize-cost ratio (which will be defined later). 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 [18]. 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 [12]. 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 can collect more prizes than the two greedy algorithms while traveling less distances.

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

2.1. Problem Formulation.

Given a weighted complete graph $G(V, E)$, where V is a set of nodes and E is a set of edges. Each edge $(u, v) \in E$ has a weight $w(u, v)$, 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. This prize can be only collected once by the salesman. Given any two nodes v_1 and v_n and a route between them $R = \{v_1, v_2, ...v_n\}$ 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 reaching 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 $C_{R_s} \leq \mathcal{B}$.

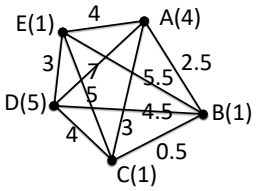


Figure 1. An example.

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. Assume $s = E$ and $t = C$. The optimal walk from E to C is E, D, B, and C, with a total prize of 8 and a total cost of 8. Other routes are not optimal. For example, although the path of E, A, B, and C is within the budget with a cost of 7, its total prize is 7. □

2.2. Combinatorial Algorithms for BC-TSP

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 unvisited and budget-feasible nodes still exist (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.

EXAMPLE 2: In Fig. 1, with $s = E$ and $t = C$, both Algo. 1 and 2 yield the optimal solution of E, D, B, and C, with a total cost of 8 and a total prize of 8. But in general, they are not optimal, as shown in Section 4. □

3. 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) [18]. Agent's decision-making in an RL system is a Markov decision process (MDP) 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.

Q-learning is a family of value-based RL algorithms [18]. 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 obtained from the next state t.

Multi-agent Reinforcement Learning (MARL) Algorithm. In our MARL framework for BC-TSP, multiple agents 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.** 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 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.

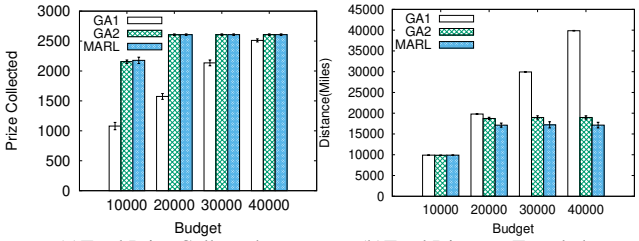
In the first step (lines 3-26), all the m agents are initially located at the starting node s with zero collected prizes. Then, each independently follows the action rule to move to the next budget-feasible node to collect prizes and collaboratively update the involved edges' Q-value. This takes place 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 BC-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.

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

4. 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 travels less distance than GA1. These demonstrate that the MARL algorithm is more efficient (distance-wise) and effective (prize-wise) than the handcrafted greedy algorithms.

5. Conclusions

We proposed an algorithmic problem called budget-constrained TSP (BC-TSP) that arises from emerging robotic applications wherein robots are dispatched to accomplish some tasks with limited battery power. Such applications include robotic sensor networks, electric 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. To our knowledge, our work is the first to apply the RL technique to solve the BC-TSP problem.

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

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

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

[7] 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*.

[8] 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.

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

[11] Michael L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

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

[13] 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)*.

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

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

[19] 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.

[20] 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.

Address for correspondence:

Zari Magnaye
Department of Computer Science, California State University Dominguez Hills
zmagnaye1@toromail.csudh.edu