Budget-Constrained Prize-Collecting Traveling Salesman Problem: a Reinforcement Learning Approach

I. Budget-Constrained Prize-Collecting Traveling Salesman Problem (BC-PC-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 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 $s, t \in V$ be the two nodes where a traveling salesman starts and finishes his travel, and let $\mathcal{B}$ denote its budget, which indicates the distance he can travel. The goal of the BC-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} is maximized while its cost $C_{R_s} \leq \mathcal{B}$. Note it is possible that $s = t$.

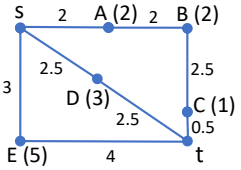

Fig. 1: Illustrating the BC-PC-TSP.

EXAMPLE 1: Fig. 1 is an illustrative example for BC-PC-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 different nodes. The optimal walk from s to t is s, E, t, C, and t, with a total collected prize of 6 and a total cost of 8. Other routes with budget $\mathcal{B}$ 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. Note a prize on a node can only be collected once even though the node can be visited multiple times. □

B. Combinatorial Algorithms for BC-PC-TSP

Below we solve BC-PC-TSP by presenting two greedy heuristic algorithms viz. Algo. 1 and 2. We first give the below definition.

Definition 1: (**Feasible Set.**) Given the current node s wherein the traveling salesman is located and his current available budget B, the set of s's neighbor nodes that the salesman can travel to while still having enough budgets to go to destination node t is called node s's *feasible set*. We denote s's feasible set with remaining budget B as $\mathcal{F}(s, B)$. That is, $\mathcal{F}(s, B) = \{u | (s, u) \in E \wedge (w(s, u) + w(u, t) \leq B)\}$. We say a node $u \in \mathcal{F}(s, B)$ is feasible to s (that is, the salesman can move from s to u without worrying about there is not enough budget to finally move to t). □

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
$\mathcal{B}$	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

Greedy Algorithm 1. Algo. 1 is a greedy algorithm. It first sorts all the nodes in the descending order of their prizes (lines 3 and 4), and then takes place in rounds (lines 6-14). In each round, with the current node s and the currently available budget B, it checks if the unvisited node with the largest prize is in s's feasible set (line 7). If so, it visits this node, updates the route and increases the collected prizes, and updates the available budget by deducting the cost of getting to this node (lines 8-10); from this new current node, it continues with the process. If not, it checks if the unvisited node with the second-largest prize is in s's feasible set (line 12). It stops when it does not have enough budget to visit any of the rest unvisited nodes (line 13), then it finally visits the destination node t and returns the route with its cost, prizes collected, and its left budget. Its time complexity is $O(|V|^2)$. Note that Algo. 1 also works for the problem where $s = t$.

Algorithm 1: Greedy Algorithm 1 for BC-PC-TSP.
Input: A complete weighted graph $G(V, E)$, s, t, and 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, 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 available budget, is $\mathcal{B}$ initially;
1. $r = s$, $R = \{s\}$, $C_R = P_R = 0$, $U = V - \{s, t\}$;
2. $B = \mathcal{B}$;
3. Sort all the nodes in $U = \{v_1, v_2, ..., v_{|V|-2}\}$ in descending

order of their prizes;
4. WLOG, let $p_{v_1} \geq p_{v_2} ..., \geq p_{v_{|V|-2}}$;
5. $k = 1$; // the index of the node with largest prize
6. **while** $(B > 0)$
7. **if** $(v_k \in \mathcal{F}(r))$
8. $R = R \cup \{v_k\}$, $C_R = C_R + w(r, v_k)$, $P_R = P_R + p_{v_k}$;
9. $B = B - w(r, v_k)$;
10. $r = v_k$;
11. **end if;**
12. $k++$;
13. **if** $(k == |V| - 1)$ **break**; // Have checked all nodes in U
14. **end while;**
15. $R = R \cup \{t\}$, $C_R = C_R + w(r, t)$, $B = B - w(r, t)$;
16. **RETURN** R, C_R, P_R, B.

Greedy Algorithm 2. Algo. 2 is similar as Algo. 1, except that in each round, it visits an unvisited feasible node with the largest 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, $pcr(u, v) = \frac{p_v}{w(u,v)}$.

In each round, at the current node r, it first checks if there are still unvisited feasible nodes that can be visited with the currently available budget B. If not, it terminates by going to destination t and returning the total prizes and costs of the route and the left budget (lines 4, 22, 23). Otherwise, it sorts the unvisited nodes in terms of their pcr to the current node r (lines 5-6) and finds the unvisited and feasible node with the largest pcr (lines 7-9). If this is successful, the salesman moves to such next node (which becomes the current node r), updates the traveling distance and collected prizes, and updates the list of unvisited nodes (lines 10-14). At this new current node r, it then repeats the process by sorting its unvisited feasible set with pcr, finding if it can go to one with the largest pcr from r (lines 3-7). If so, it moves to this node (lines 10-14); otherwise (i.e., it cannot find an unvisited node that is within its current budget), it terminates and returns (lines 18, 22, 23).

Algorithm 2: Greedy Algorithm 2 for BC-PC-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 available budget, is $\mathcal{B}$ initially;
$flag$: if there are still unvisited nodes that can be visited from the current node r with available budget B, initially true;
1. $r = s$, $R = \{s\}$, $C_R = P_R = 0$, $U = V - \{s, t\}$;
2. $B = \mathcal{B}$, $flag = true$;
3. **while** $(B > 0)$
4. **if** $(flag == false)$ **break**;
5. Sort $u \in U = \{v_1, v_2, ..., v_{|U|}\}$ in descending order of $prc(r, u)$;
6. WLOG, let $pcr(r, v_1) \geq pcr(r, v_2) ..., \geq pcr(r, v_{|U|})$;
7. $k = 1$; // the index of the unvisited node with largest pcr
8. **while** $(k \leq |U|)$
9. **if** $(v_k \in \mathcal{F}(r))$
10. $R = R \cup \{v_k\}$;
11. $C_R = C_R + w(r, v_k)$, $P_R = P_R + p_{v_k}$;
12. $B = B - w(r, v_k)$, $U = U - \{v_k\}$;
13. $r = v_k$;
14. **break**;
15. **end if;**
16. **else**
17. $k++$; // try next unvisited node with largest prize
18. **if** $(k == |U| + 1)$ $flag = false$;
19. **end else;**
20. **end while;**
21. **end while;**
22. $R = R \cup \{t\}$, $C_R = C_R + w(r, t)$, $B = B - w(r, t)$;
23. **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. □

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

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

Reinforcement Learning (RL) [3]. 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-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 BC-PC-TSP well, as in BC-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 [3], [2], 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 [3]. 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-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 as many prizes as possible while staying within the traveling budget $\mathcal{B}$, and ending at t.

Action Rule in BC-PC-TSP. For an agent located at any node s, the *action 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. When none of its unvisited neighbors is feasible, the agent goes to destination t and terminates in this episode. The action rule is specified below: 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.

- **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 feasible set with remaining budget B, 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 $w(s, u)$ of the edge (s, t) and the prize p_u available at node u.
- **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 $U \cap \mathcal{F}(s, B) = \phi$, which means it does not have a neighbor in its feasible set that is not visited. In this case, the agent goes to destination t and terminates in this episode.

The above action rule is based on ϵ-greedy exploration [3], 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 [1]. 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 1-32) and an execution stage (lines 33-39). 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 3-26), 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 node to collect prizes and collaboratively updates the Q-value of the involved edge. 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, they must wait for others to finish. Otherwise, it moves to the next node and 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), it finds among the m routes the one with the maximum collected prizes and updates the reward value and Q-value of the edges that belong to this 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 which 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 budget $\mathcal{B}$.
Output: A route R from s to t, C_R, and P_R.
Notations: 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;

1. **for** $(1 \le k \le epi)$ // Learning stage
2. All the m agents are at node s, i.e., $r_j = s, 1 \le j \le m$;
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** $(U_j \cap \mathcal{F}(r_j, B_j) == \phi)$ (empty set) // 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. $Q(r_j,t) = (1-\alpha) \cdot Q(r_j,t) +$
$\alpha \cdot \gamma \cdot \max_{z \in U_j \cap \mathcal{F}(t,B_j)} Q(t,z)$; // Update Q-value
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. $Q(r_j,s_j) = (1-\alpha) \cdot Q(r_j,s_j) +$
$\alpha \cdot \gamma \cdot \max_{z \in U_j \cap \mathcal{F}(s_j,B_j)} Q(s_j,z)$; // Update Q-value
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 maximum 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
33. $r = s$, $R = \{s\}$, $C_R = 0$, $P_R = 0$, $B = \mathcal{B}$; // Execution stage
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. 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 $O(epi \cdot m \cdot |V|)$.

REFERENCES

[1] 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)*.
[2] Michael L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.
[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. The MIT Press, 2020.