BUDGET-CONSTRAINED TRAVELING SALESMAN PROBLEM: A COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING APPROACH

A Thesis

Presented

to the Faculty of

California State University, Dominguez Hills

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Vincent Mak

Spring 2024

THESIS: BUDGET-CONSTRAINED TRAVELING SALESMAN PROBLEM: A COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING APPROACH

AUTHOR: Vincent Mak

APPROVED:

Bin Tang, Ph.D.
Thesis Committee Chair

Sanaz Rahimi Moosavi, Ph.D.
Committee Member

Alexander Chen, Ph.D.
Committee Member

ACKNOWLEDGEMENTS

I extend my gratitude to my thesis advisor, Dr. Tang, for their consistent and thorough collaboration, and for imparting valuable insights into the research process. Additionally, I would like to express appreciation to Dr. Chen and Dr. Moosavi for their contributions as members of my committee and their dedication. Special thanks to NSF Grant 2240517 for their grant, which supported this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The Traveling Salesman Problem (TSP) is one of computer science's most famous combinatorial problems. This thesis studies a variation of the TSP called the Budget-Constrained Traveling Salesman Problem (BC-TSP). BC-TSP is inspired by a few emerging network applications, including robotic sensor networks and autonomous electric vehicles. Given a weighted complete graph G(V,E) where node i ∈ V has an available prize of p_i, two nodes s,t ∈ V, and a budget, the goal of the BC-TSP is to find the salesman a route from s to t to maximize his collected prizes while keeping his travel cost within the budget. To solve BC-TSP, we design a suite of algorithms, including a prize-driven multi-agent reinforcement learning algorithm (P-MARL) and a deep reinforcement learning approach utilizing Recurrent Neural Network (RNN). We give an analysis of the convergence between P-MARL and RNN using synthetic data of state capital cities of the U.S., we show that the P-MARL outperforms the RNN by collecting 45.3% more prizes while taking only 15.2% of its execution time. To our knowledge, our work is the first to design a MARL technique with guaranteed convergence to solve the BC-TSP problem.

CHAPTER 1 - INTRODUCTION

Background. The Traveling Salesman Problem (TSP) is arguably the most famous combinatorial problem in computer science, engineering, and operation research [24], [9]. In this thesis, inspired by a few emerging network applications, we study a new variation of the TSP called the Budget- Constrained Traveling Salesman Problem (BC-TSP). In contrast to the traditional TSP, wherein the goal is to find a route to visit all the nodes in the most efficient manner, in BC- TSP, each node is associated with a prize, and the traveling salesman has a budget; the goal of the salesman is to visit a subset of the nodes to maximize the collected prizes while staying within his budget.

BC-TSP is motivated by several robotic applications [16], [23], wherein one or multiple robots are dispatched to the field to accomplish tasks of different importance, such as search and rescue and planetary exploration. As robots are mainly powered by batteries, the robot might exhaust its battery power before finishing all its assignment tasks. When this takes place, one critical goal for the network operator is to schedule the untethered robot to accomplish as many important tasks as possible before returning to the charging station for recharge. One specific application is data collection in robotic sensor networks (RSNs) [21], where mobile robots are dispatched into sensor fields to collect sensory data. It has been shown that this approach can greatly prolong the network lifetime of sensor networks by migrating the energy bottleneck from sensor nodes with limited battery power to robots that can be recharged and re-dispatched indefinitely [45], [11].

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

BC-TSP is defined as follows. Given a weighted complete graph G(V, E) where node i ∈ V has an available prize of p_i, two nodes s, t ∈ V , and a budget, the goal of the salesman is to find a route from s to t to maximize his collected prizes while keeping his travel cost within the budget. Here, the prizes model the tasks of different importance that various network applications attempt to accomplish, and the budget is a resource constraint in the network applications. Therefore, both prizes and budget are application-specific. For example, the prizes could be the importance of the search and rescue missions or the value of the sensory data to be collected in the RSNs; the budget could be the remaining battery power of the robots or the Uber driver's electric car, or the computing power of the agents in many AI/ML applications [20].

BC-TSP is NP-hard; when the budget is unconstrained, it degenerates into the TSP. However, unlike the TSP, which only needs to sequence the nodes, the BC-TSP requires both selection and sequencing of the nodes, making it a more challenging problem than the TSP.

Motivation. Our main contribution is a cooperative multi-agent reinforcement learning (MARL) framework that specifically leverages the prizes available at nodes to achieve efficient learning of prize-collecting in BC-TSP. Unlike those above handcrafted combinatorial algorithms for BC-TSP, in MARL, intelligent agents learn cooperatively by interacting with the environment and thus are more adaptive in a dynamic network environment [36]. As such, RL has become an ideal alternative to solve many NP-hard combinatorial problems time-efficiently [27]. RL is particularly relevant to solving sequential combinatorial problems such as TSPs and vehicular routing problems (VRPs), demonstrated by recent research [6], [28]. This is because when the traveling salesman or vehicles in the TSP and VRP decide to move from one node to another in a road network to serve the customers, it resembles the Markov decision process adopted in RL, where the agents transition among states of the environment.

By studying the BC-TSP, however, we observe that the benefits of RL and combinatorial optimization problems to each other are somewhat mutual. That is, not only can RL help to design efficient learning algorithms to solve combinatorial problems, but the BC-TSP serves as a lens through which the existing RL paradigm and techniques can be scrutinized and further improved. First and foremost, the prominent feature of the BC-TSP, which are available prizes at nodes, closely resembles the rewards in the RL, the signals agents receive when interacting with the environment. As such, the prizes in BC-TSP could be utilized to build more effective reward models in the RL compared to other combinatorial problems such as TSP and VRP. Second, the prize-maximization goal of the traveling salesman in BC-TSP resembles naturally the RL agent's goal of maximizing accumulative discounted rewards. Such resemblance serves as the common ground upon which BC-TSP can be tapped to design more powerful RL algorithms.

Despite the above observations, how to exploit the synergy between the prize maximization in BC-TSP and the cumulative reward maximization in RL to uncover more powerful and effective RL algorithms remains largely unexplored by the research community. In this thesis, we thus ask the following question: *How can we take advantage of the unique problem feature of node prizes in BC-TSP to design time-efficient RL algorithms with guaranteed convergence?*

Our Contribution. We address this new challenge by designing a cooperative multi-agent RL (MARL) framework. This framework, termed *prize-driven* MARL (P-MARL), integrates the prizes available at nodes into the reward model of the RL via the cooperative effort of multiple learning agents. In particular, P-MARL consists of an active exploration mechanism for all the cooperative agents and a novel reward update mechanism in P-MARL called *multiplicative increase reward update* that constantly aligns the best-inferred action of a traveling salesman in the execution stage with the prize-collecting information learned by multi-agents in the training stage. We show that P-MARL can find an optimal prize-collecting route in the execution as long as the multi-agents have found it in the training stage. We show that our learning algorithm is solution-effective and time- efficient via extensive simulations under different network and RL parameters. In particular, it outperforms an existing work by collecting 45.3% more prizes while taking only 15.2% of its execution time.

Thesis Organization. The rest of the thesis is organized as follows. Section 2 reviews all the related work. Section 3 formulates the BC-TSP. In Section 4, we introduce reinforcement learning and how it is used to solve BC-TSP. Section 5 and 6 present the P-MARL and RNN algorithms and its mechanism. Section 7 compares our algorithms with the existing research and discusses the results. Section 8 concludes the thesis with a discussion of future works.

CHAPTER 2 - RELATED WORK

In this section, we review all the theory work in BC- TSP and the existing research in robotic sensor networks that inspires the BC-TSP. We also review the deep reinforcement learning (DRL)-based approach for combinatorial optimization and argue why we adopt a multi-agent RL (MARL) approach instead of DRL to solve the BC-TSP.

Existing BC-TSP Research. A few works from the theory and operations research community have studied BC-TSP [35], [30], [4]. In his Ph.D. thesis, Sokkappa [35] systematically studied BC-TSP. He proved the problem is NP-hard and that no fully polynomial approximation scheme exists unless P = NP. He proposed branch-and-bound-based heuristics to solve the BC-TSP and optimal solutions for several special cases. Other efforts have been developed to find approximation algorithms for problems closely related to BC-TSP. Levin [4] presented a $(4 + \varepsilon)$-approximation algorithm to the so-called budget prize collecting tree problem, which finds a subtree with maximum prizes while the cost of the tree (i.e., the sum of all its edge weights) stays in a budget. Recently, Paul et al. [30] improved it by a 2-approximation algorithm based on a primal-dual approach while maximizing the number of vertices visited (i.e., each vertex has the same prize). However, none of them adopted an RL approach, which is the main focus of this thesis. To our knowledge, our work is the first to apply the MARL technique to solve the BC-TSP problem.

Research in Robotic Sensor Networks (RSNs). RSNs [17], [21] have drawn lots of attention in recent years, wherein mobile robots are utilized to enhance the system performance of wireless sensor networks. As the BC-TSP is inspired by the data-collecting robots with limited battery power in the RSNs, we briefly review a few representative works [26], [18], [39], [32], [40], [22], [44]. Ma et al. [26] introduced mobile data collectors to gather data in large-scale sensor networks. In a single mobile collector case, the goal is to minimize the length of the data-gathering tour; in the multiple-collector case, the goal is to minimize the number of mobile collectors such that each subtour does not exceed some time constraint.

They formulate the problems as mixed-integer programs and present heuristic data-gathering algorithms. Guo et al. [18], [39] extended it by introducing wireless energy-charging into mobile data collecting. They formulated a network utility maximization problem considering energy balance and the bounded sojourn time of the mobile collector and designed a distributed algorithm.

All of the above work assumes enough battery power for the mobile unit or robots to collect all the data in the field. In a large-scale sensor field, it is possible that the robot does not have enough battery power to visit all the sensor nodes. When sensor nodes generate sensory data with different priorities and values (i.e., the prizes), a critical question is how to schedule the robot to visit them and collect data on maximum prizes before returning to the charging station to recharge.

Deep Reinforcement Learning (DRL) for Combinatorial Optimization. DRL, which combines RL and artificial neural networks with representation learning of large data sets, is a powerful technique that has been widely used for a diverse set of applications [5]. Recently, DRL is utilized to solve many combinatorial problems [14], [6], [28], [46]. Bello et al. [6] presented a DRL-based framework for solving combinatorial optimization problems using neural network-based function approximation algorithms. They solved the TSP by training a recurrent neural network (RNN) and optimizing its parameters using a policy gradient method. Nazari et al. [28] further extend it to solve vehicle routing problems, a generalization of TSPs, by considering a parameterized stochastic policy and applying a policy gradient algorithm to optimize its parameters. It can produce the solution as a sequence of consecutive actions in real time without re-training every new problem instance. Training a graph neural network [33] with RL on an unlabeled training set of graphs, Drori et al. [14] developed a unified framework using RL with a Graph Neural Network (GNN) representation for learning to approximate different combinatorial problems over graphs. The trained network can output approximate solutions to new graph instances in linear running time. Very recently, Zhang et al. [46] 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 [43]. Refer to [27] for a comprehensive review of all the DRL techniques for combinatorial optimization.

The closest DRL work to ours is Wei et al. [41], which proposed an RNN algorithm to solve the informative path planning problem (IPP). In IPP, a robot is dispatched into a sensing field to collect the sensing information, called mutual information, which measures the informativeness of data (i.e., sensor placement) collected along a path in the field. The informativeness of the path can be associated with the vertices, edges, or both on the path. IPP aims to find the most informative path from a pre-defined start location to a terminal location subject to a budget constraint. When the informativeness is defined on the vertices and is additive, IPP becomes the well-known orienteering problem (OP) [38].

In OP, each vertex is associated with a reward, and the goal is to find a subset of vertices to collect a maximum reward amount within a budget constraint. As OP is very similar to the BC-TSP studied in this thesis, we compare our MARL algorithm with the RNN algorithm in [41]. In particular, we show that P-MARL outperforms the IPP by collecting 45.3% more prizes while taking only 15.2% of its execution time.

We argue that DRL is not the best learning technique to solve the BC-TSP. First, DRL is particularly effective in handling complex tasks with high-dimensional and continuous input spaces, wherein data sets have a large number of features or dimensions. In contrast, BC-TSP has a low-dimensional and discrete setting regarding the agent's states and actions, wherein nodes are the states and edges represent the actions that can be taken when an agent is in a state. Second, feature extraction, which identifies the data set's most discriminating characteristics to reduce the data dimensionality, is one of the main strengths of deep learning algorithms [10]. However, the most prominent feature of a BC-TSP instance, the prizes available at nodes, is already available and does not need further processing. As such, feature extraction is an overkill for the BC-TSP. Third, training deep neural networks with a large number of parameters (weights and biases) involves optimization algorithms such as gradient descent and its variants [5]). These algorithms iterate over the entire dataset multiple times, adjusting parameters to minimize a defined loss function. As such, it is a computationally intensive process that either takes time to converge to an optimal solution or, many times, it only produces an approximate solution with a large optimality gap [6].

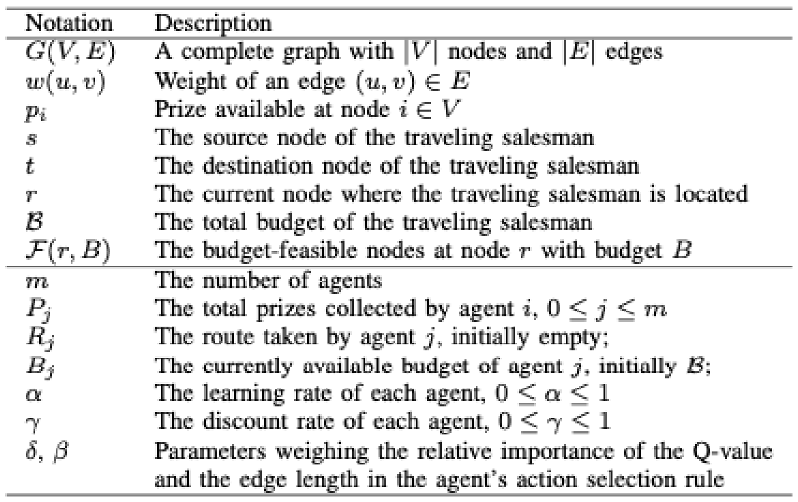
Instead, we propose a MARL-based learning framework called P-MARL to solve the BC-TSP. In our framework, multiple agents share their learning from the environment while cooperatively updating the same Q-table to find the optimal route. We give a theoretical proof of the convergence of our MARL algorithm. Via extensive simulation, we show that P-MARL outperforms the existing DRL-based approach by Wei et al. [41], [42] by collecting 45.3% more prizes while taking only 15.2% of its execution time.

Other Related Work. Our work was inspired by Ant-Q [15], an algorithmic framework combining the Q-learning algorithm [36] and ant colony optimization [8]. Ant-Q models intelligent behavior and 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. However, Ant-Q does not always converge to the optimal solution [13]. Our P-MARL algorithm further improves Ant-Q by introducing an active exploration mechanism for all the cooperative agents and a novel reward update mechanism in P-MARL called *multiplicative increase reward update* that constantly updates the Q-table to reflect the latest global-best route that has been found so far. We show that P-MARL not only collects more prizes than Ant-Q but also converges to optimal as long as the global optimal is found in the training stage. Recently, Ruiz et al. [31] studied the prize-collecting traveling salesman problem (PCTSP) using an RL approach. In PCTSP, the goal of the traveling salesman is to find a route from s to t such that the sum of the prizes of all the nodes along the route reaches a preset quota while the distance along the route is minimized. As the budget is not a constraint in PCTSP, it is less challenging than the BC-TSP, wherein the salesman has to constantly check if his remaining budget is sufficient enough for him to return to the destination. Besides, as it is straightly derived from Ant-Q, it does not always converge. These call for new RL algorithms and reward models to solve the BC-TSP efficiently and optimally.

CHAPTER 3 - 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) ∈ E has a weight $w(u, v)$, indicating the travel distance or cost on this edge. Each node $i \in V$ has a weight $p_i \geq 0 \in$ R+, indicating the prize available at this node. Given any two nodes v_1 and v_n and a route between them R = $\{v_1, v_2, \dots, v_n\}$ 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 it is 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 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 $R_s = \{ s = v_1, v_2, \dots, v_t = t\}$ such that its total prize P_R is maximized while its cost $C_R \leq$ B. When $s = t$, the salesman starts and ends at the same node. Table 1 summarizes the notations used throughout the thesis.

Table 1 - Notation Summary

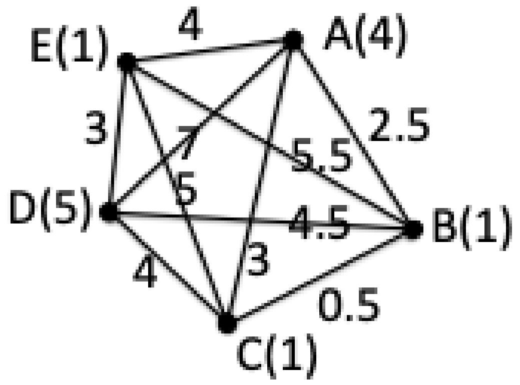


Notation	Description
$G(V, E)$	A complete graph with $\|V\|$ nodes and $\|E\|$ edges
$w(u, v)$	Weight of an edge $(u, v) \in E$
p_i	Prize available at node $i \in V$
s	The source node of the traveling salesman
t	The destination node of the traveling salesman
r	The current node where the traveling salesman is located
$\mathcal{B}$	The total budget of the traveling salesman
$\mathcal{F}(r, B)$	The budget-feasible nodes at node r with budget B
m	The number of agents
P_j	The total prizes collected by agent i, $0 \leq j \leq m$
R_j	The route taken by agent j, initially empty;
B_j	The currently available budget of agent j, initially $\mathcal{B}$;
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

EXAMPLE 1: Fig. 1 illustrate BC-TSP with budget 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.

Figure 1 - BC-TSP Example



CHAPTER 4 - REINFORCEMENT LEARNING

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 × A → S is a state transition function, and
- r: S × A → 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 ∈ A to transition to state $t(s, a) \in S$ while receiving a reward $r(s, a) \in R$. The agent maintains a policy $\pi(s) : S \rightarrow A$ 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 [36], [25], 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 $(0 \leq \gamma \leq 1)$ 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.

4.1 Q-Learning

Q-learning is a family of value-based algorithms [36]. 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

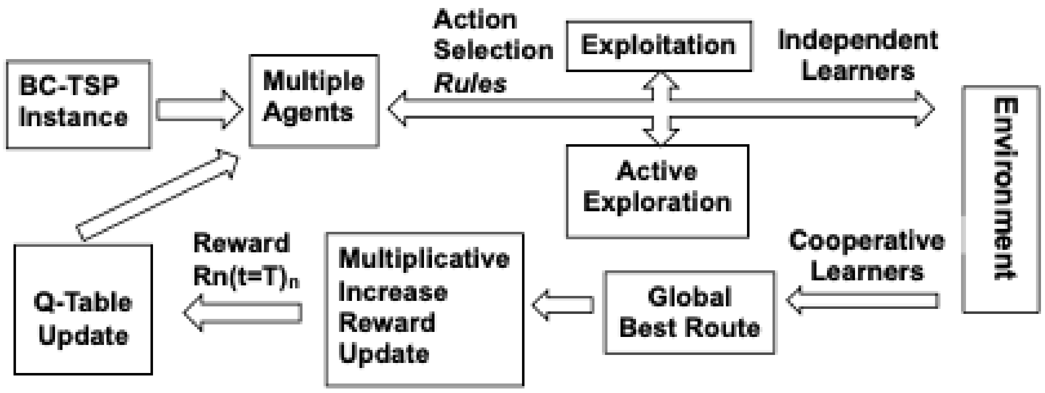
$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r(s, a) + \gamma \cdot max_b Q(t, b)] \tag{1}$$

where $0 \leq \alpha \leq 1$ is the learning rate that decides to what extent newly acquired information overrides old information in the learning process. In equation 1, $max_b Q(t, b)$ is the maximum reward that can be obtained from the next state t.

CHAPTER 5 - COOPERATIVE P-MARL ALFORITHM

To extend the Q-learning approach outlined above to multi-agent scenarios [7], we propose a hybrid model where agents act both independently and cooperatively. In the independent learning phase, each agent adheres to an action selection rule (defined subsequently), determining its next move to collect prizes while adhering to budget constraints. During this phase, agents are unaware of other agents and act independently. In the cooperative learning phase, once all agents have determined their prize-collecting routes, they communicate and evaluate if a new global-best route has been discovered in this episode. The global-best route is the route with the highest accumulated prizes since the trial's inception. Figure 2 illustrates the workflow of the P-MARL approach.

Figure 2 - P-MARL workflow



In this setup, multiple agents start from node s and collaboratively learn the state-action Q-table and reward table while acting synchronously. We first introduce the action selection rule for all learning agents and then outline the P-MARL algorithm.

5.1 Action Selection Rule of Agents

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

Exploitation. In exploitation, equation 2, the agent selects a node to visit randomly. In this context, U represents the set of nodes that the agent has not visited, and $F(s, B)$ is the set of nodes that are feasible within the agent's budget, and δ and β are preset parameters. When q > q0, where q is a random value in [0,1] and q0 (0≤q0 ≤1) is a preset value, exploitation is selected; otherwise, the agent chooses exploration explained below.

$$t = argmax_{u \in U \cap F(s,B)} \frac{[Q(s,u)]^{\delta} * p_u}{[w(s,u)]^{\beta}} \tag{2}$$

Exploration. In active exploration, equation 3, the agent chooses a node $t \in U \cap F(s, B)$ to move to by the following probability distribution:

$$p(s,t) = \frac{\frac{\left([C(s,t)]^{\delta} * p_u\right)}{[w(s,t)]^{\beta}}}{\sum_{u \in U \cap F(s,B)} \frac{\left([C(s,u)]^{\delta} * p_u\right)}{[w(s,u)]^{\beta}}} \tag{3}$$

A node $u \in U \cap F(s, B)$ is selected with the highest probability $p(s, u)$, while $\sum_{u \in U \cap F(s,B)} p(s, u) = 1$. Here, $C(u, v) = \frac{p_u + p_v}{w(u,v)}$ remains unchanged in the learning process. The distribution $p(s, t)$ describes the desirability of moving to the next node t based on factors such as edge lengths and node prizes. Nodes with shorter edge lengths and higher node prizes are more desirable destinations.

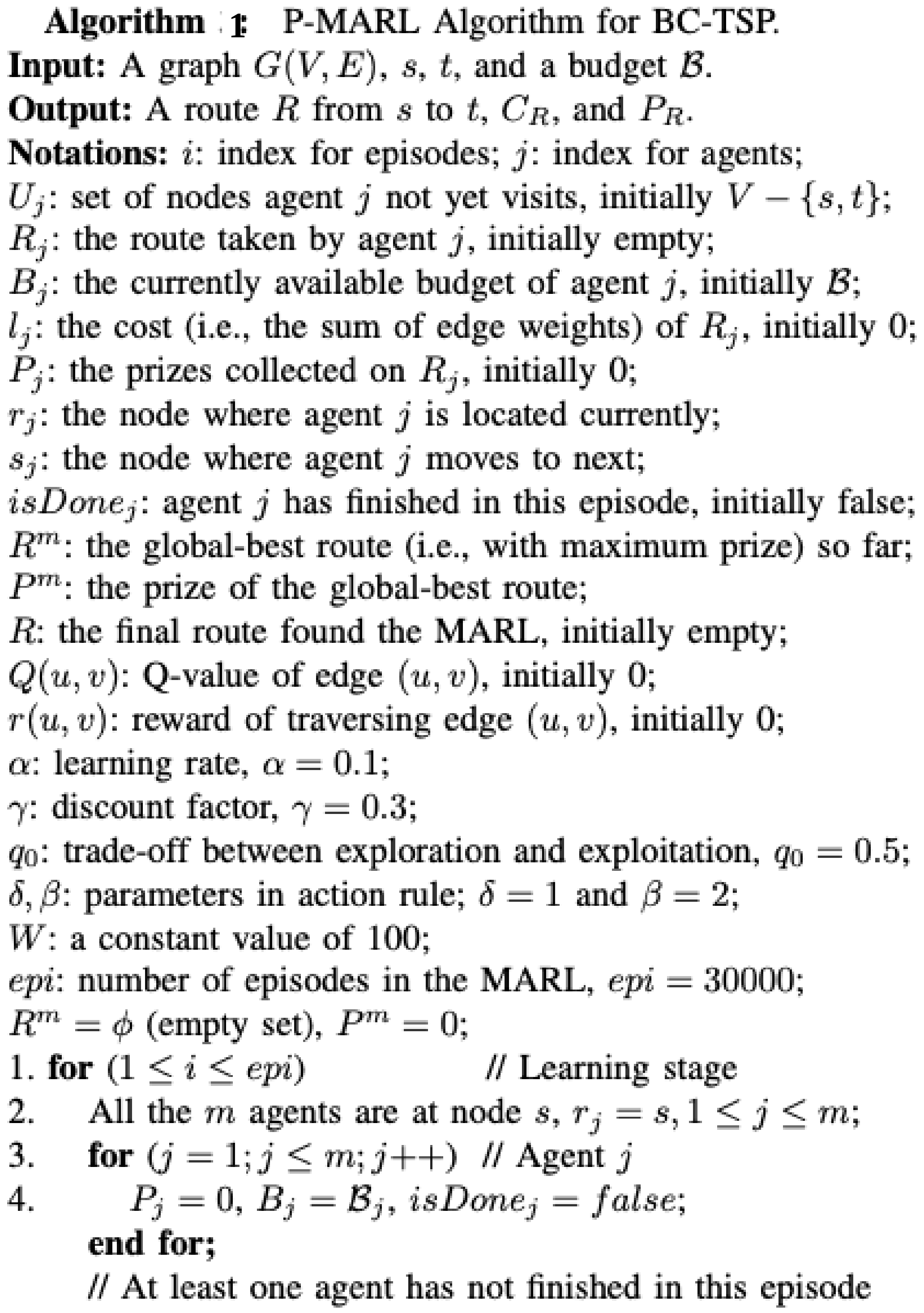
Termination. When $u \in U \cap F(s, B) = \phi$, which means the agent does not have an unvisited budget-feasible node to move to, the agent proceeds to destination t and terminates in this episode.

5.2 P-MARL Algorithm

In P-MARL, in Algorithm 1, each trial consists of a learning stage for the m agents to learn a commonly shared Q-table (lines 1-33), and an execution stage for the traveling salesman to find the prize-collecting route and to collect the prizes (lines 34-39). The learning stage takes place in a present number of episodes. Each episode consists of the below two steps.
In the first step (lines 3-24), 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. 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 8-13); 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 14-21). Here, the prizes at each node can be collected multiple times by different agents (as this is the learning stage).

In the second step (lines 25-32), the m agents compare with each other their prize-collecting route and check if there is a new *global-best route* found in this episode. Here, the global- best route is a route found so far with the maximum collected prizes since the beginning of the trial. If so, it updates the reward value and Q-value of the edges of this newly found global-best route (lines 29-30). In particular, in line 29, for any each edge (u, v) in the newly found global-best route, we increase its reward value of r(u, v) by $\frac{i*W}{P_{j^*}}$, where i is the P_{j^*} current episode number, W is a constant, and P_{j^*} is the prize of the global-best route. We call this reward value update the *multiplicative increase reward update*, and its rationale will be explained in section 5.3.

In the execution stage (lines 34-38), 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.



```
Algorithm 1: P-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;
$R^m$: the global-best route (i.e., with maximum prize) so far;
$P^m$: the prize of the global-best route;
$R$: the final route found the MARL, initially empty;
$Q(u, v)$: Q-value of edge $(u, v)$, initially 0;
$r(u, v)$: reward of traversing edge $(u, v)$, initially 0;
$\alpha$: learning rate, $\alpha = 0.1$;
$\gamma$: discount factor, $\gamma = 0.3$;
$q_0$: trade-off between exploration and exploitation, $q_0 = 0.5$;
$\delta, \beta$: parameters in action rule; $\delta = 1$ and $\beta = 2$;
$W$: a constant value of 100;
$epi$: number of episodes in the MARL, $epi = 30000$;
$R^m = \phi$ (empty set), $P^m = 0$;
1. for $(1 \leq i \leq epi)$                // Learning stage
2.     All the $m$ agents are at node $s$, $r_j = s, 1 \leq j \leq m$;
3.     for $(j = 1; j \leq m; j{+}{+})$  // 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 ($\exists\ j,\ 1 \leq j \leq m,\ isDone_j == false$)
6.     for ($j = 1; j \leq m; j{+}{+}$)  // 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.             $r_j = t$;
13.          end if;
14.          else
15.             Finds the next node $s_j$ following action rule;
16.             $R_j = R_j \cup \{s_j\}$;
17.             $l_j = l_j + w(r_j, s_j)$, $B_j = B_j - w(r_j, s_j)$;
18.             $P_j = P_j + p_{s_j}$;              // Collect prize
19.             $r_j = s_j$;                    // Move to node $s_j$;
20.             $U_j = U_j - \{s_j\}$;
21.          end else;
22.       end if;
23.    end for;
24. end while;
25. $j^* = \text{argmax}_{1 \leq j \leq m} P_j$;
26. if ($P_{j^*} > P^m$)                // Found a global-best route
27.    $P^m = P_{j^*}$, $R^m = R_{j^*}$;
28.    for (each edge $(u, v) \in R_{j^*}$)
29.       $r(u, v) = r(u, v) + \frac{i \cdot W}{P_{j^*}}$;     // Update reward value
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 if;
33. end for; // End of each episode in learning stage
    // Execution stage
34. $r = s$, $R = \{s\}$, $C_R = 0$, $P_R = 0$, $B = \mathcal{B}$;
35. while ($r! = t$)
36.   $u = \text{argmax}_b Q(r, b)$;
37.   $R = R \cup \{u\}, C_R = C_R + w(r, u), P_R = P_R + p_u$,
      $B = B - w(r, u)$;
38.   $r = u$;
39. end while;
40. RETURN  $R$, $C_R$, $P_R$, $B$.
```

5.3 Rationale of the Multiplicative Increase Reward Update

Line 29: $r(u, v) = r(u, v) + \frac{i * W}{P_{j^*}}$ is referred to as the multiplicative increase reward update. Its rationale is as follows. Whenever the current global-best route is found, it is preferred that it be embedded in the Q-table so that in the execution stage, the traveling salesman can always follow this global-best route to receive the maximum amount of prizes. To achieve this, we only need to increase the reward values of any edge (u, v) in this global best route in a way such that for node u, the reward $r(u, v)$ is the maximum among all the edges (u, v) ∈ E. Fortunately, the episode number i can be utilized to achieve this, as the episode number corresponding to the global-best route is the largest when $r(u, v)$ is updated, thus guaranteeing that $r(u, v)$ is the maximum among all the edges (u, v′) ∈ E.

5.4 Ant-Q

The main idea of the Ant-Q is to model the exploration-exploitation trade-off of RL for a group of agents (i.e., ants) to find the optimal solution for the traveling sales- man problem cooperatively. All the m ants at the starting node make independent and parallel moves following the below exploration-exploitation distribution function. Given its current node s, an ant moves to another node t following

$$\begin{cases} t = \operatorname{argmax}_{u \in U \cap \mathcal{F}(s,B)} \frac{[Q(s,u)]^{\delta} \times p_u}{[w(s,u)]^{\beta}}, & q > q_0, \\ p(s,t) = \frac{([Q(s,t)]^{\delta} \times p_u)/[w(s,t)]^{\beta}}{\sum_{u \in U \cap \mathcal{F}_{(s,B)}} ([Q(s,u)]^{\delta} \times p_u)/[w(s,u)]^{\beta}}, & q \le q_0. \end{cases} \tag{7}$$

That is, when q > q0, an ant chooses to exploit what it has learned by moving to a node u that maximizes the learned Q-value Q(s,u). Otherwise, it adopts a pseudo- random-proportional, which gives the probability with which an ant chooses a node t to move to. It then updates the same Q-table using only the discounted next-state evaluation:

$$Q(r_j, t) = (1-\alpha) \cdot Q(r_j, t) + \alpha \cdot \gamma \cdot \max_{z \in U_j \cap \mathcal{F}(t, B_j)} Q(t, z). \quad (8)$$

This is repeated until each ant has finished its tour and is back in the starting node. Finally, the edges belonging to the shortest tour j∗ are updated using

$$Q(u, v) \leftarrow (1-\alpha) \cdot Q(u, v) + \alpha \cdot [\Delta Q(u, v) + \gamma \cdot \max_b Q(v, b)], \quad (9)$$

where $\Delta Q(u, v) = \frac{W}{P_{j^*}}$ if (u, v) belongs to the best tour j^* found in this round and zero otherwise. The loop is repeated until a termination condition is met. In Ant-Q, the termination condition is verified after a fixed number of episodes or when no improvement is obtained for a fixed number of episodes.

5.5 Difference between P-MARL and Ant-Q

P-MARL is inspired by Ant-Q [15], a combined colony optimization and RL approach to solving combinatorial optimization problems. The main idea of the Ant-Q is to model the exploration-exploitation trade-off of RL for a group of agents (i.e., ants) to find the optimal solution for the traveling salesman problem cooperatively. There are three major differences between Ant-Q and P-MARL. First, each agent in P-MARL only update the Q-table at the end of an episode when a global-best route is found. Second, multiplicative increase reward update approach is used to update the Q-value in P-MARL. Lastly, in P-MARL, each agent will terminate at the destination node within assigned budget.

Theorem 1: In the training stage, if one of the m agents finds the optimal route for the BC-TSP, the salesman will receive the maximum prize in the execution stage.

Proof: This is due to our designed reward model, wherein the reward of an involved edge (u, v) is increased by an amount $\frac{i \cdot W}{P_{j*}}$, where i is the episode number, $W = 100$, and P_{j*} is the prizes of the global-best route j^*. As this increased amount of reward is proportional to i, which always increases, it guarantees that for any node u, if (u, v) belongs to the global-best route, then $r(u, v)$ must the largest among all edges $r(u, v')$. Thus, the Q-table "encodes" the global route immediately after it is found in the training stage. As a result, in the execution stage, the traveling salesman can follow the maximum Q-value from each node to traverse this global-optimal route.

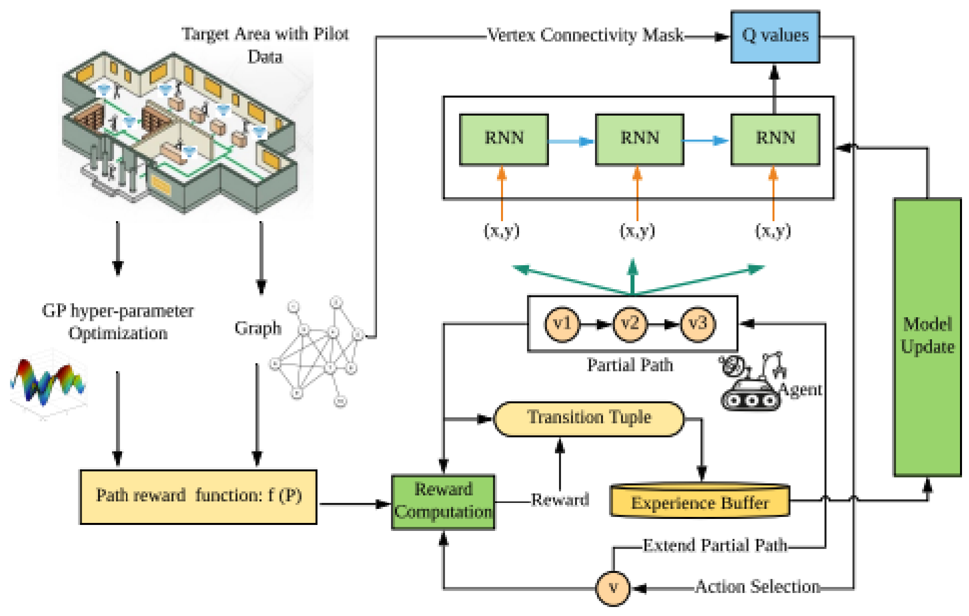
CHAPTER 6 – DEEP Q-LEARNING NETWORK

In this section, we present a solution that employs a Q-learning approach to simulate the setup as described in [41]. We first outline the solution architecture and then discuss the reward mechanism, the Q-learning network, and the learning algorithm. The Q-learning network has been implemented and utilized to evaluate the performance of P-MARL.

6.1 Solution Overview

The model employs a Recurrent Neural Network (RNN) to approximate Q-values, as future rewards depend on all visited vertices. For each input state, a Q-value is assigned to every vertex in the graph, even if it is not a direct neighbor of the last vertex of the path. These Q-values are then masked with the graph's connectivity to filter out non-reachable vertices. In each epoch, the agent starts from the initial vertex and selects actions according to an ε-greedy policy based on the Q-values. Rewards are calculated using f(P), and state transition tuples are added to the experience buffer. At each step, a batch of transition tuples is sampled from the buffer to update the model's parameters by minimizing the temporal difference in equation 6. Fig 3 shows the overall architecture of this solution.

Figure 3 – Architecture of RNN [41]



6.2 Reward Mechanism

Upon taking each action, the environment delivers an immediate reward signal and transitions to the next state. The reward of taking action $a \in A'(P_p)$ is defined in equation 4 where $f(P_p + [a])$ represents the total reward of the epoch after taking the action while $f(P_p)$ is the reward before taking the action a. In contrast to the reward mechanism outlined in [41], the agent cannot visit any infeasible nodes, thus no penalty reward is incurred.

$$r(P_p, a) = f(P_p + [a]) - f(P_p) \tag{4}$$

The process yields a transition tuple $< s,\ a,\ r,\ s',\ IsDone >$, where taking action a from state s results in the agent receiving reward r, the state transitioning to s', and IsDone indicating whether the action terminates the episode. This transition tuple is stored in an experience buffer, which serves as input for training the Q-network.

6.3 Q-learning Network

The Q-learning network predicts Q-values to help the agent make optimal decisions. We opt for an RNN-based neural network because of the sequential nature of the input. Given a path P_p, the RNN takes as input the 2D location coordinates corresponding to each vertex in P_p. The output of the final cell is a Q-value vector Q_m with a length equal to the number of vertices |V|. However, because the graph might not be fully connected and the predicted Q-values are only applicable to adjacent vertices, we introduce a masking technique as in equation 5 which we will only consider vectors that are neighbors of the current node.

$$Q_m[i] = \{ \begin{matrix} 1 & v_i \in N(v_k) \\ 0 & else \end{matrix} \qquad (5)$$

6.4 Learning Algorithm

Using the Q-network, the agent employs an ε-greedy policy to explore the solution space, incorporating the same constrained exploration and exploitation strategy as P-MARL mentioned in Section 5.1. The state transition tuples $< s,\ a,\ r,\ s',\ IsDone >$ explained in Section 6.2 are stored in an experience buffer M, and the network parameters are trained using this memory buffer. At each step, a batch of transition is sample from the memory buffer. The network is optimized in an iterative way by minimizing the temporal difference with a loss function defined as in equation 6.

$$L(\theta) = (Q(s_t, a_t) - \left(r_t + \ \gamma\, max_a Q(s_{t+1}, a)\right))^2 \qquad (6)$$

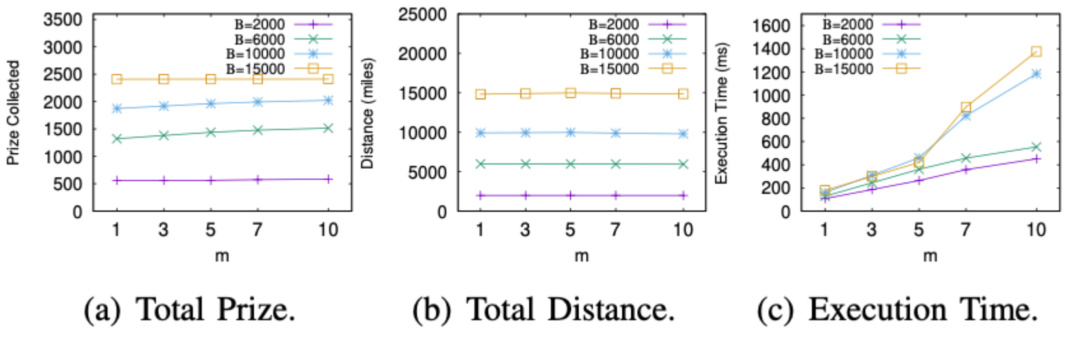
CHAPTER 7 - SIMULATION RESULTS

This section outlines the research findings. Firstly, we analyze the effect of varying the number of agents in P-MARL. Following that, we compare the performance of P-MARL with RNN using a dataset containing 48 cities in the U.S.

7.1 Impacts of Number of Agents in P-MARL.

We study the impact of the number of agents m on the P-MARL's performance. We vary m from 1, 3, 5, 7, to 10, and the budget B from 2,000, 6,000, 10,000, to 15,000. Fig. 4(a) shows that for each m, the higher the B, the larger the collected prize. A close look shows that when B equals to 6,000 and 10,000, the collected prizes are increased when increasing m. This is because the more agents work collaboratively, the better chance that they can find the global prize-collecting route. Fig. 4(b) shows the traveled distance of the P-MARL w.r.t. m and B. The higher the B, the more distances it can travel. Here, we observe that varying m does not seem to affect the traveled distance of the salesman, as the focus of the P- MARL algorithm is to collect as much prize as possible while staying within the budget. Finally, Fig. 4(c) shows for each B, with the increase of m, the execution time of the P-MARL algorithm increases. As more agents are learning the prize- collecting route in each episode and the P-MARL executes in a sequential manner for all the agents, it takes more time to find the global route. As there is a big jump in the execution time from m = 5 to 7 while the prizes and distance stay almost the same, it suggests that in our scenarios, 5 agents are the right number of agents for the prize-collecting learning process. We leave it as future work to investigate if there is any theoretical underpinning to explain the optimal number of agents with respect to the execution time of the learning process.

Figure 4 – MARL with varying number of Agents m



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

7.2 Comparing P-MARL with RNN.

We compare P-MARL with the RNN under various budget constraints, as shown in Fig. 5, 6, and 7. In Fig 5, the total prizes collected are depicted, revealing that P-MARL consistently outperforms RNN, with the performance disparities being particularly pronounced at smaller budgets. In Fig. 6, it is observed that, at smaller budgets, both algorithms cover similar distances due to budget exhaustion. However, at larger budgets, P-MARL incurs less distance cost compared to RNN. Fig. 7 showcases that the training time of P-MARL remains consistently low, whereas the training time of RNN increases with higher budgets. This efficiency in training time underscores P-MARL’s superiority over RNN. These findings collectively demonstrate that the P-MARL algorithm is not only more efficient in terms of distance but also more effective in terms of prize collection compared to the DRL approach. This is due to DRL’s focus on high-dimensional and continuous input spaces, the feature extraction process, and training deep neural networks with a large number of parameters with complex optimization algorithms such as gradient descent and its variants [5]). As such, it is a computationally intensive process that either takes time to converge to an optimal solution or, many times, it only produces an approximate solution with a large optimality gap [6].

Figure 5 – Comparing reward between P-MARL and RNN in execution stage

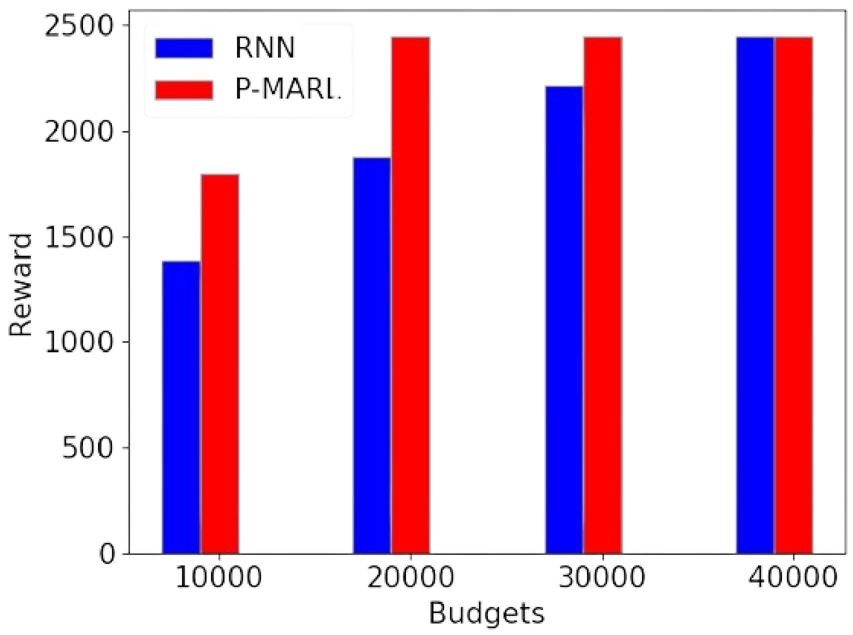


Figure 6 – Comparing distance between P-MARL and RNN in execution stage

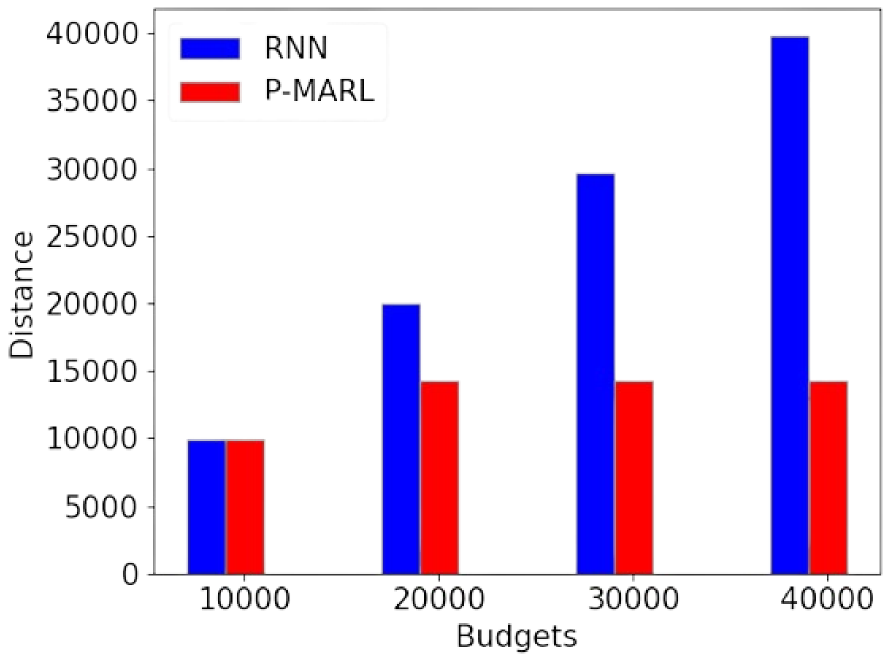


Figure 7 – Comparing execution time between P-MARL and RNN in execution stage

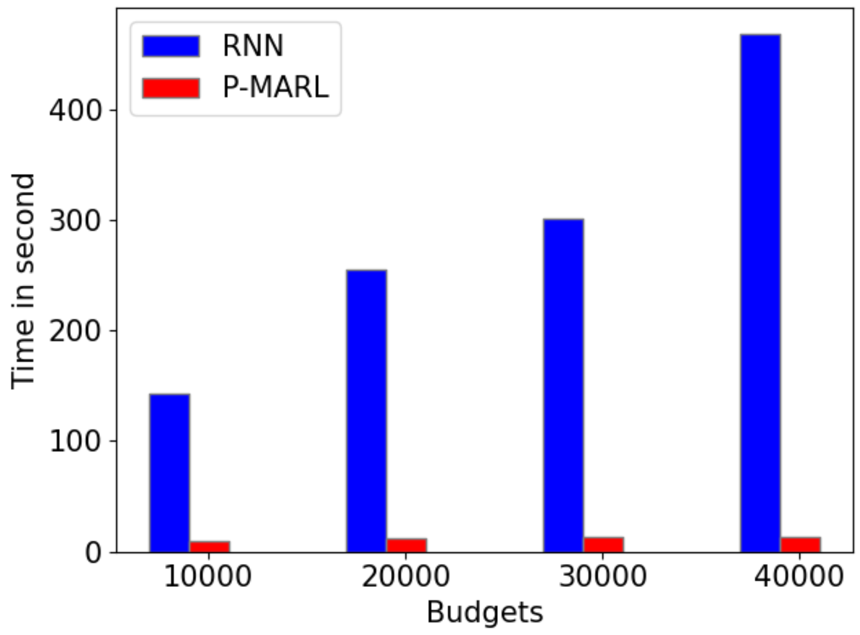


Fig. 8, 9, 10, and 11 show the performance comparison of P- MARL and RNN in the training stage. The rewards attained by RNN and P-MARL are scrutinized over 5000 training episodes across various budget constraints. The rewards (i.e., the collected prizes) are presented as a running average derived from 100 episodes, with a 95% confidence interval. Fig. 8 highlights P-MARL's early capacity to discern positional signals within the initial 200 episodes under a budget of 20,000. Fig. 9, 10, and 11 reveal P-MARL consistently achieving maximum prize collection under budgets of 20,000, 30,000, and 40,000, respectively, while maintaining stable performance. Conversely, the RNN model demonstrates a lack of effective learning from the environment and performs notably poorly. These findings underscore the efficiency and consistent performance of the P-MARL algorithm in comparison to the RNN model throughout the training process.

Figure 8 – Comparing P-MARL and RNN in training stage with budget = 10,000

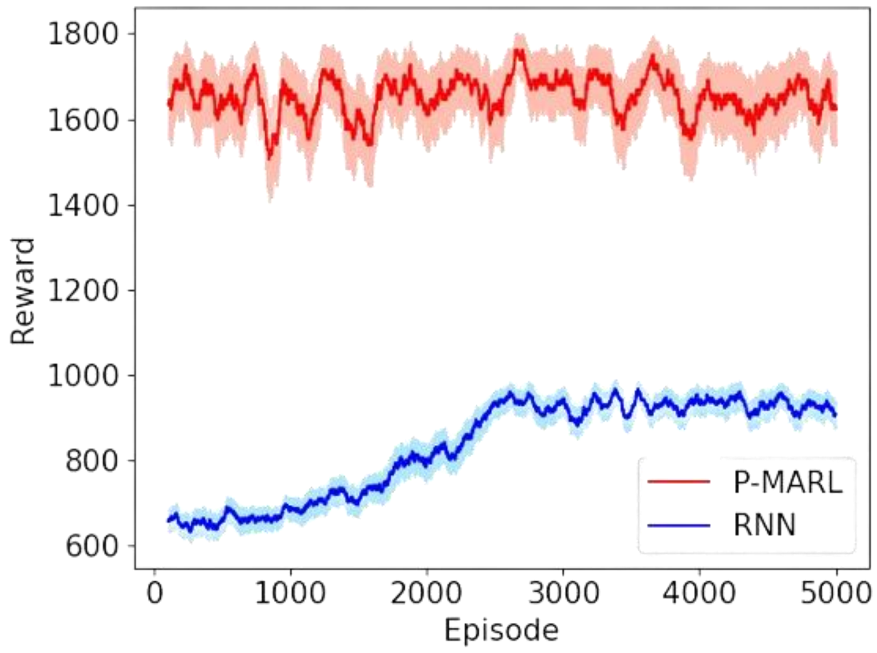


Figure 9 – Comparing P-MARL and RNN in training stage with budget = 20,000

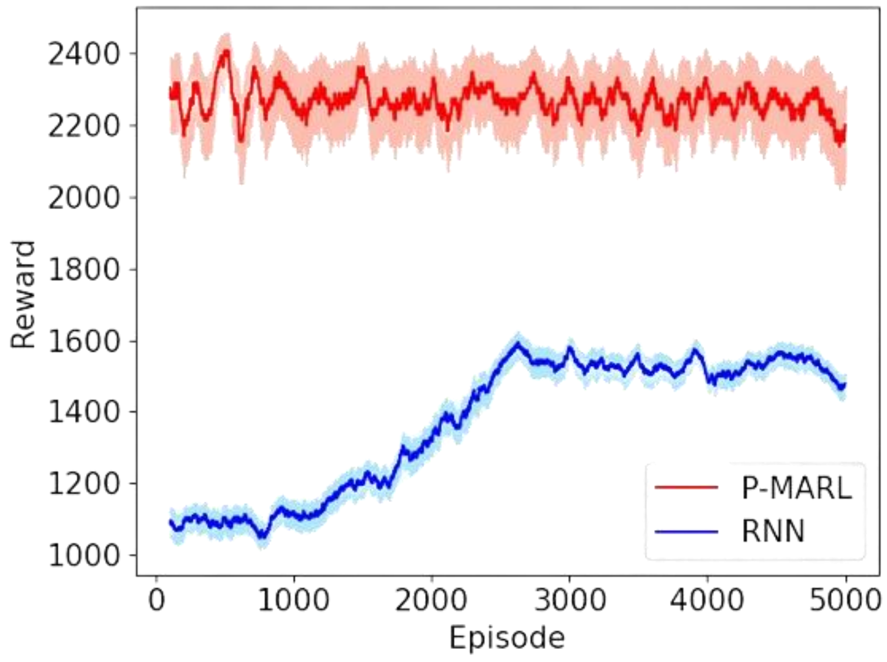


Figure 10 – Comparing P-MARL and RNN in training stage with budget = 30,000

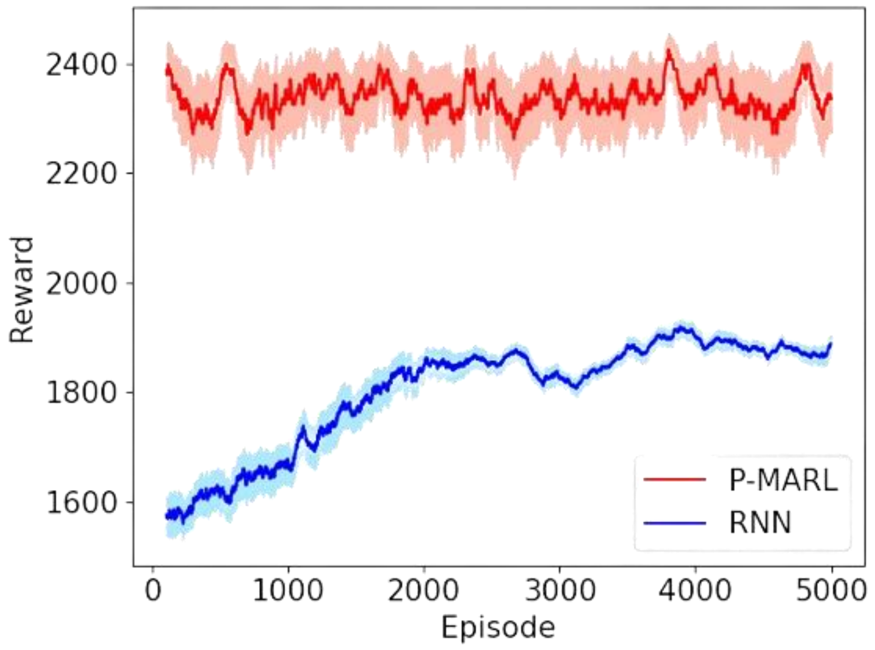
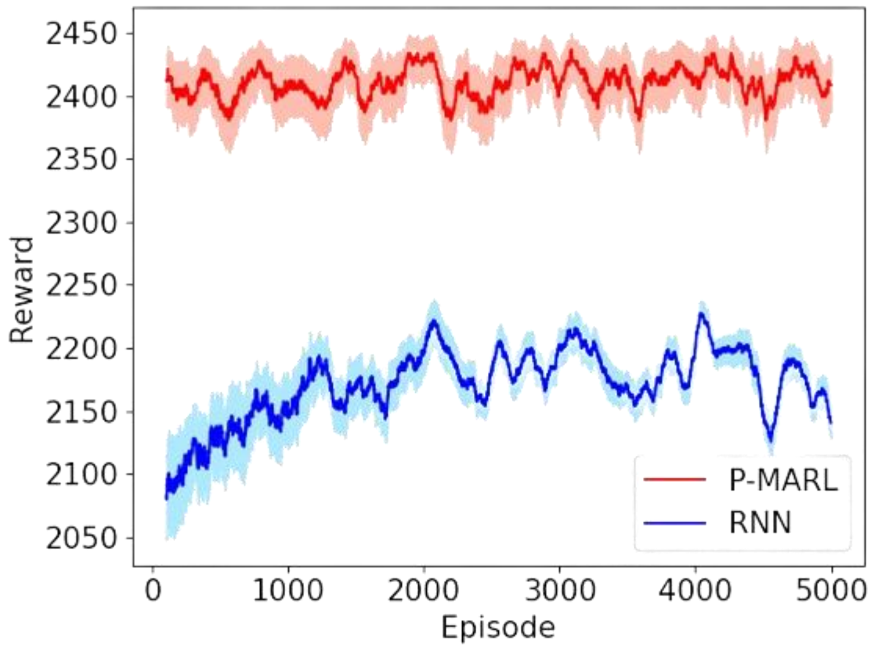


Figure 11 – Comparing P-MARL and RNN in training stage with budget = 40,000



7.3 RNN convergence

To improve the RNN model's performance, we increased the training episodes to 50,000 and observed the results. Fig 12, 13, 14, and 15 illustrate the learning process of the RNN. However, the model still lacks convergence, as it reaches its maximum potential with a budget of only 40,000, while the nearest neighbor algorithm indicates that the maximum reward can be achieved with a slightly higher budget of around 13,000. Additionally, we noticed that the RNN stops learning after 25,000 episodes for different budgets. This behavior is attributed to the linear epsilon policy setup, suggesting that it may not be suitable for this use case.

Figure 12 – RNN average reward per episode with budget = 10,000

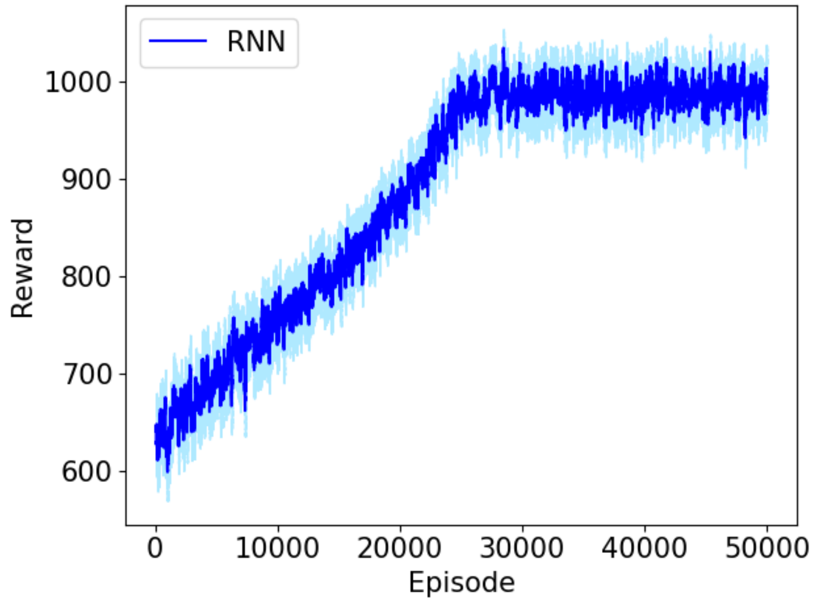


Figure 13 – RNN average reward per episode with budget = 20,000

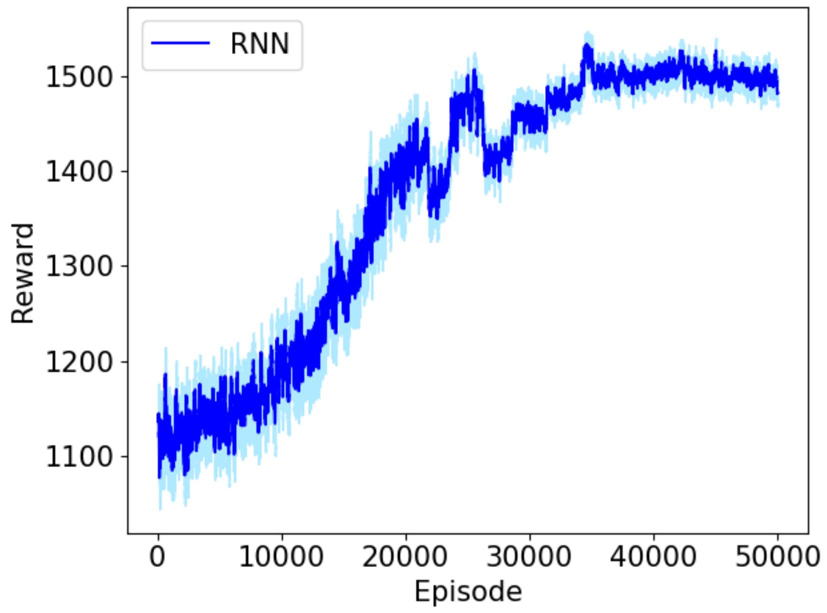


Figure 14 – RNN average reward per episode with budget = 30,000

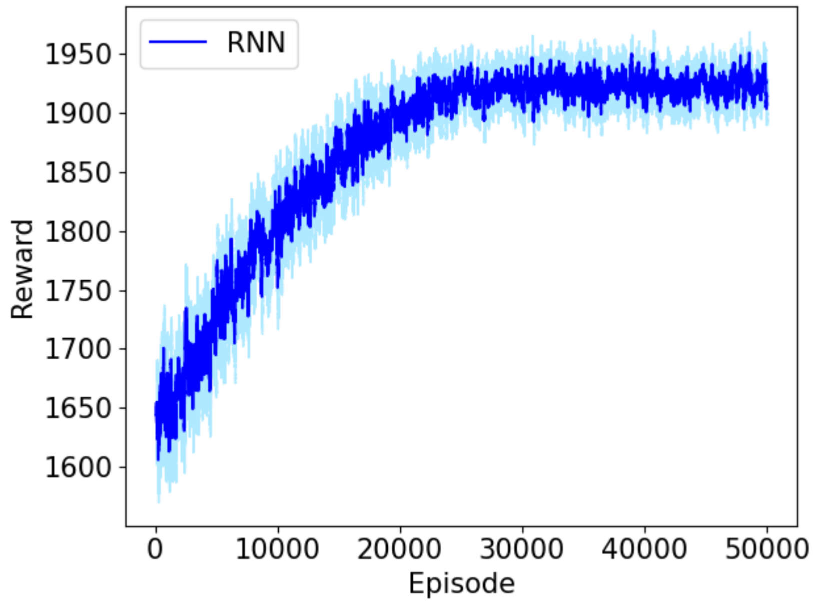
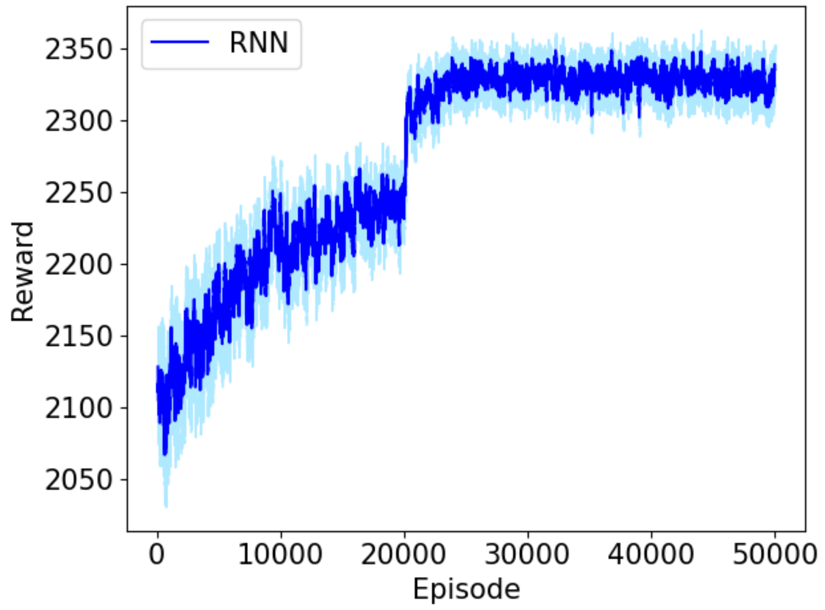


Figure 15 – RNN average reward per episode with budget = 40,000



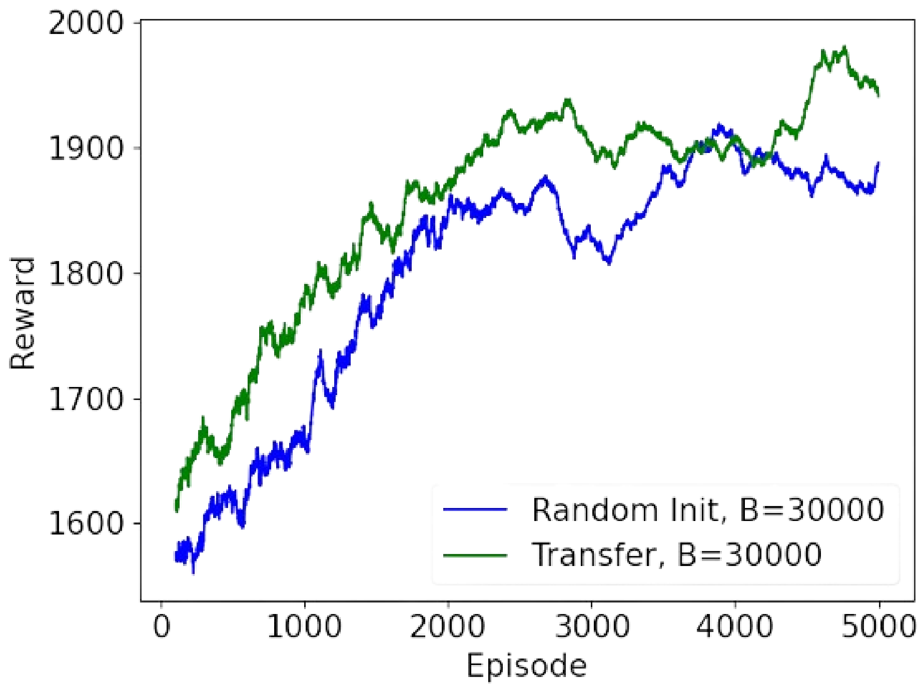
CHAPTER 8 – TRANSFER LEARNING

In real-world scenarios, the budget constraint B is typically tied to the battery capacity. Additionally, the starting vertices may vary. A key question arises: can we adjust the trained models to accommodate these changing constraints? This concept is known as transfer learning, where the Q-network parameters can be initialized either randomly or from pre-trained models. This section conducts experiments to show that the trained models can indeed adapt when one of the constraints changes.

8.1 Transfer learning with different budget

Fig. 16 illustrates the impact of transfer learning when the budget varies. Initially, the base model is trained using a budget of 40,000. Subsequently, the budget is adjusted, and the model is fine-tuned based on the original base model. A comparison is made between the learning curves of a model initialized randomly and the fine-tuned base model. The results indicate that the fine-tuned base model converges more rapidly than the randomly initialized model. This rapid convergence and superior performance of the fine-tuned base model highlight the effectiveness of transfer learning.

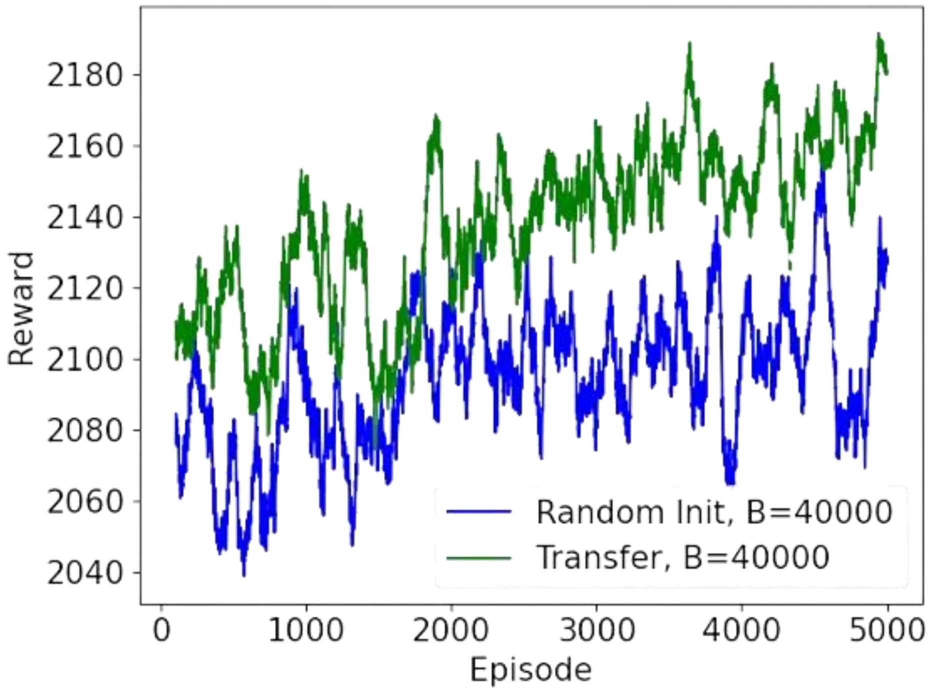
Figure 16 - Transfer learning with different budget



8.2 Transfer learning with different starting city

Fig 17 depicts the outcomes when the starting vertex is altered. Initially, the base model is trained with a starting vertex of $v_s = 0$ and a budget of 40,000. The learning curves of a randomly initialized model and the fine-tuned base model are compared. The findings distinctly indicate the beneficial impact of transfer learning. The fine-tuned base model demonstrates quicker convergence compared to the randomly initialized model. Furthermore, the fine-tuned base model achieves superior results by collecting more prizes from the environment.

Figure 17 - Transfer learning with starting vertices



CHAPTER 9 - CONCLUSION

We introduce a problem called budget-constrained Traveling Salesman Problem (BC-TSP), which arises in various robotic applications where robots are tasked with completing missions using limited battery power. Examples include robotic sensor networks, electric cars in ride-sharing, and automated warehouses. We compare our Multi-Agent Reinforcement Learning (MARL) approach with a Deep Reinforcement Learning (DRL)-based approach. The results indicate that the MARL approach outperforms DRL in terms of convergence and time efficiency for training and execution. However, since MARL relies on Q-learning as its core architecture, the algorithm needs to be recalculated to suit different instances. Therefore, we investigate transfer learning using the trained DRL model as a base model. The results demonstrate that fine-tuning the base model leads to quicker convergence than training a new model from scratch, while also achieving better performance. In future work, we plan to implement a multi-agent approach with DRL. Additionally, we intend to explore other DRL models such as actor-critic and determine the most effective model. Finally, we aim to expand the experimental environment to include more nodes and compare the performance of P-MARL and RNN.

CHAPTER 10 - REFERENCES

[1] Haversine formula. https://en.wikipedia.org/wiki/Haversine formula.

[2] Miller–tucker–zemlin (mtz) subtour elimination constraint. https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html.

[3] Traveling salesman tour of us capital cities. https://www.math.uwaterloo.ca/tsp/data/usa/index.html.

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

5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017.

[6] I.Bello,H.Pham,Q.V.Le,M.Norouzi,andS.Bengio.Neuralcombina- torial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

[7] 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.

[8] B. Chandra Mohan and R. Baskaran. A survey: Ant colony optimiza- tion based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 39(4):4618–4627, 2012.

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

[10] S. Dara and P. Tumma. Feature extraction by using deep learning: A survey. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1795–1801.

[11] Mario Di Francesco, Sajal K. Das, and Giuseppe Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. 8(1), 2011.

[12] 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.

[13] MarcoDorigoandLucaMariaGambardella.Astudyofsomeproperties of ant-q. In Hans-Michael et al. Voigt, editor, *Parallel Problem Solving from Nature — PPSN IV*, 1996.

[14] I. Drori et al. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 19–24, 2020.

[15] L. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *ICML*, 1995.

[16] L. C. Garaffa, M. Basso, A. A. Konzen, and E. P. de Freitas. Rein- forcement learning for mobile robotics exploration: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3796– 3810, 2023.

[17] Y. Gu, F. Ren, Y. Ji, and J. Li. The evolution of sink mobility man- agement in wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):507–524, 2016.

[18] S. Guo, C. Wang, and Y. Yang. Joint mobile data gathering and energy provisioning in wireless rechargeable sensor networks. *IEEE Transactions on Mobile Computing*, 13(12):2836–2852, 2014.

[19] A.GuezH.VanHasseltandD.Silver.Deepreinforcementlearningwith double q-learning. In *30th AAAI Conference on Artificial Intelligence*, 2016.

[20] J. Hua, L. Zeng, G. Li, and Z. Ju. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21(4), 2021.

[21] 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.

[22] D. Kim, L. Xue, D. Li, Y. Zhu, W. Wang, and A. O. Tokuta. On theoretical trajectory planning of multiple drones to minimize latency in search-and-reconnaissance operations. *IEEE Transactions on Mobile Computing*, 16(11):3156–3166, 2017.

[23] J.Kober,J.A.Bagnell,andJ.Peters.Reinforcementlearninginrobotics: A survey. 32(11), 2013.

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

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

[26] M. Ma, Y. Yang, and M. Zhao. Tour planning for mobile data- gathering mechanisms in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 62(4):1472–1483, 2013.

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

[28] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio et al., editor, *Advances in Neural Information Processing Systems,* volume 31. Curran Associates, Inc., 2018.

[29] S. Niu, Y. Liu, J. Wang, and H. Song. A decade survey of transfer

learning (2010–2020). *IEEE Transactions on Artificial Intelligence*, 1(2):151–166, 2020.

[30] 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)*.

[31] J. Ruiz, C. Gonzalez, Y. Chen, and B. Tang. Prize-collecting traveling salesman problem: a reinforcement learning approach. In *Proc. of IEEE ICC*, 2023.

[32] H.Salarian,K.W.Chin,andF.Naghdy.Anenergy-efficientmobile-sink path selection strategy for wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 63(5):2407–2419, 2014.

[33] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20:61–80, 2009.

[34] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2016.

[35] PadminiR.Sokkappa.Thecost-constrainedtravelingsalesmanproblem, 1991. Ph.D. Thesis, Stanford University.

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

[37] Ming Tan. *Multi-agent reinforcement learning: independent vs. coop- erative agents*, page 487–494. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[38] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. Orienteering problem: A survey of recent variants, solution approaches and applica- tions. *European Journal of Operational Research*, 255(2):315 – 332, 2016.

[39] C. Wang, S. Guo, and Y. Yang. An optimization framework for mobile data collection in energy-harvesting wireless sensor networks. *IEEE Transactions on Mobile Computing*, 15(12):2969–2986, 2016.

[40] Y.-C. Wang and K-C. Chen. Efficient path planning for a mobile sink to reliably gather data from sensors with diverse sensing rates and limited buffers. *IEEE Transactions on Mobile Computing*, 18(7):1527–1540, 2019.

[41] Y. Wei and R. Zheng. Informative path planning for mobile sensing with reinforcement learning. In *IEEE INFOCOM 2020*, 2020.

[42] Y. Wei and R. Zheng. A reinforcement learning framework for efficient informative sensing. *IEEE Transactions on Mobile Computing*, 21(7):2306–2317, 2022.

[43] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

[44] L. Xue, D. Kim, Y. Zhu, D. Li, W. Wang, and A. O. Tokuta. Multiple heterogeneous data ferry trajectory planning in wireless sensor networks. In *IEEE INFOCOM 2014*, pages 2274–2282.

[45] Y. Yang and M. Zhao. Optimization-based distributed algorithms for mobile data gathering in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(10):1464–1477, 2012.

[46] 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.

APPENDIX A

https://github.com/Vincentmak1994/Vincentmak1994-Multi-agent-reinforcement-learning-in-TSP