LEARNING FROM ROUTES: A COMPARATIVE STUDY OF PCTSP MODELS FOR ENHANCED DECISION-MAKING

Thedore Ho, Ryan Lau, Gary Kar & Frances C. Watson
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA
{teho, rclau, ckar, fwatson}@usc.edu

1 INTRODUCTION

The Prize Collecting Traveling Sales Problem (PCTSP) is a distinctive variant of the widely studied Traveling Salesman Problem (TSP) with additional layers of complexity and practical considerations. In the traditional TSP, given a list of cities and the distances between them, a salesperson aims to determine the shortest route to visit specified locations. The PCTSP attaches prizes to each location and requires the salesperson to collect a minimum prize value, adding to the intricacy of the problem. The salesperson must now strategically balance the cost of travel against the value of each location, potentially avoiding sites of lesser value if it leads to a more efficient route. This augmented complexity aligns the PCTSP more closely with real-world scenarios where the trade-off between travel costs and potential gains holds significant importance. The objective of this project is to develop a Q-learning algorithm that improves upon existing solutions to the PCTSP in terms of performance, efficiency, and scalability.

This project is pivotal due to the lack of known polynomial algorithms that can offer optimal solutions for all NP-Hard optimization problems, unless P equals NP. Current conventional algorithms frequently rely on handcrafted heuristics or commercial solvers, often resulting in near-optimal solutions constrained by certain factors or entailing non-deterministic methods. Conversely, Deep Learning (DL) showcases promise by acting as a potential breakthrough, utilizing its ability to learn as high-quality universal approximators. Furthermore, the scope of PCTSP extends to a diverse array of practical contexts. This problem was initially formulated by Balas (1989) as a model for scheduling the daily operations of a steel rolling mill, since then it has been widely studied to solve problems such as the radio tower planning for telecommunication, and the design of tourist routes. As such, the findings from this project could yield valuable insights and solutions beneficial across multiple industries.

2 RELATED WORK

- Ruiz et al. (2022) presented a multi-agent reinforcement learning solution that is very similar to ours however it is not able to outperform greedy heuristic algorithms.
- Chen et al. (2021) conducted an analysis of Q-Learning algorithms and demonstrated techniques such as decaying learning rate could be useful in applications such as the PCTSP.
- Gonzalez et al. (2023) designed an algorithm for the Budget-Constrained Traveling Salesman Problem, another variant of the TSP. Using Q-learning, it was able to outperform greedy algorithms in some cases.
- Climaco et al. (2020) designed a Branch-and-Cut algorithm and two Mixed Integer Programming heuristics that could solve instances of the PCTSP in a much shorter time than existing solution, but only for a small number of nodes.
- Liu et al. (2023) developed a Hybrid Q-Learning Network-Based Method for routing problems that could compute optimal solutions in shorter times for cases with a large amount of nodes.

- Bogyrbayeva et al. (2022) conducted a survey of reinforcement learning algorithms for NP-Hard vehicle routing problems and outlined improvements to traditional solutions.
- Bienstock et al. (1993) presented an approximation algorithm for the PCTSP using modified linear programming relaxation which can be extended to other prize-collecting problems.
- Xu et al. (2022) studied attention-based reinforcement learning models for multiple vehicle routing problems and developed a new model with context embedding for increased scalability.
- Chaves & Lorena (2005) combined multiple existing models based on clustering to solve the PCTSP and were able to obtain quicker results using a more aggressive search on a small number of nodes.
- Blauth & Nagele (2023) developed an approximation algorithm for the PCTSP which was able to improve the approximation guarantee with respect to the natural linear programming relaxation of the problem.

3 Background: Integer Linear Programming Optimization

We present an integer linear program by Bienstock et al (1993). Let $G = (V, E)$ be a complete and undirected graph, in which V and E are the sets of vertices and edges. For each edge $e_{(i,j)} \in E$, there is a cost c_e satisfying the triangle inequality, and for each vertex $i \in V$ there is a non-negative penalty w_i and a prize p_i.

Here the binary decision variables $x_{ij} = 1$ if the edge $(i, j) \in E$ is part of the tour or $x_{ij} = 0$ otherwise, and $y_i = 1$ if the vertex $i \in V$ is visited or $y_i = 0$ otherwise. For every subset of vertices S, we also need to define $\delta(S) = \{(i, j) \in E | i \in S, j \in V \setminus S\}$ for the subset of edges where only one end is in S but the other end in $V \setminus S$, and $\delta(i)$ as a set of all incidents edges to some vertex i and a root vertex u for every subset of vertices S.

Hence the PCTSP can be formulated as follows:

$$
\begin{array}{ll}
\text{min} & \sum_{e \in E} c_e x_e + \sum_{i \in V} w_i (1 - y_i) \\
\text{subject to} & \sum_{e \in \delta(i)} x_e = 2y_i, \forall i \in V \\
& \sum_{i \in V} p_i y_i \geq p_{min} \\
& \sum_{e \in \delta(S)} x_e \geq 2y_i, \forall (S \subset V \setminus \{u\}, i \in S) \\
& y_i \in \{0, 1\}, \forall i \in V \\
& x_e \in \{0, 1\}, \forall e \in E.
\end{array}
$$

The objective function aims to minimize the sum of travel costs and penalties. The first constraint ensures that a feasible solution goes exactly once through each visited vertex. The second constraint ensures a minimum bound on the collected prize. The third constraint ensures the connectivity of the route in order to eliminate sub-tours by ensuring that if a vertex i is in the tour but not in S, then there must be at least 2 edges into or out of S. The last two constraints ensure all variables need to be either 0 or 1.

However, this standard textbook ILP formulation will have an exponential number of constraints to prevent sub-tours which is less than ideal for relatively larger instances with $|V| \geq 20$.

For the sake of simplicity, we will assume p_{min} to be non-negative and the starting vertex u to always be node 0.

4 Software Design

In this project, a single agent reinforcement learning Q-Learning algorithm was created to solve the PCTSP. Our Q-Learning algorithm was adapted from MARL, a multi agent reinforcement learning algorithm published from the California State Univeristy by Ruiz et al. (2022). A single agent RL algorithm was chosen because the authors of MARL found no improvement in performance with multiple agents compared to one. The details of the Q-Learning agent and enviroment can be found below. A link to the full codebase in Google Colab can be found in the appendix.

4.1 ENVIRONMENT SET-UP

Other authors have chosen to use real-world data sets such as the cities in the United States. However, this project uses a random generation technique for a more general analysis of the PCTSP on an unbiased environment. The environment consisted of 1000 cities. Each city had a random prize in the range (0, 100) and random 2D coordinates in the range (1, 100,000) for each dimension. The minimum prize the agent needed to collect was 8333 which was calculated proportional too the maximum possible prize for a city and the number of cities. The goal of the agent is to find the smallest path such that the sum of prizes of the cities in the path are greater than or equal to the prize quota.

4.2 Q-LEARNING FORMULAS

p_x = the prize of city x
$w(u, v)$ = weight (L2 norm) of edge (u, v).
$Q(s, a)$ = Q value of edge (s, a)
α = learning rate
$R(s, a)$ = Reward value of edge (s, a)
γ = Discount Factor
U = Set of unvisited cities
N_u = Number of unvisited cities

The initialization of the Q value and reward tables were borrowed from MARL algorithm with some adaptations.A positive reward was used as oppose to a penalty reward used by MARL because this was more consistent with conventional Q-Learning reward strategies. Furthermore, a 1/2 scaling factor in the initial Q values was used to make the initial Q values close to the reward values.

Initial Q value of edge (u,v): $\dfrac{p_u + p_v}{2 * w(u, v)}$

Reward value of edge (u,v): $\dfrac{p_u}{w(u, v)}$

Q-Learning Algorithm: $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(R(s, a) + \gamma max_{t \in U} Q(a, t))$

The exploitation method simply selects the city with the highest Q-Value as the next city and the exploration method selects the next city following the distribution weighted on the Q-Values of the cities. However, it was discovered that this did not yield enough exploration in our algorithm so we used a uniformly random selection for exploration and the weighted distribution for exploitation. Below are the probabilities p(s,t) of selecting the next city t given the current city s for exploration and exploitation respectively. The exploitation algorithm was adapted from the MARL exploration algorithm. Note that the exploitation is a uniform distribution which results in a random selection. The agent selects exploration vs exploitation using a predefined probability.

Exploitation: $p(s, t) = \dfrac{Q(s, t) * p_u / w(s, t)^2}{\sum_{u \in U} [Q(s, u) * p_u / w(s, u)^2]}$

Exploration: $p(s, t) = \dfrac{1}{N_u}$

4.3 HYPER-PARAMETERS

Hyper-parameters were primarily chosen through a process of trial and error. A decaying learning rate was used as it showed better performance in Chen et al. (2021). A learning rate a = .07 was used. Along with a discount factor of .2 and a probability of choosing exploration or exploitation of .3 and .7 respectively.

```python
def get_next_explore(self, cities):

  unvisited_idx = self.unvisited[:,0]
  self.next_node = int(random.choice(unvisited_idx))


def get_next_exploit(self, cities):

  sum_val = self.q_inval[[int(i) for i in self.unvisited[:,0]]].sum()

  prob_vals = self.q_inval/sum_val

  prob_vals = prob_vals.reshape((1,cities.shape[0]))
  cities_prob = np.insert(cities, 4, prob_vals, axis =1)

  unvisited_prob = [(int(i), cities_prob[int(i),4]) for i in self.unvisited[:,0]]

  unvisit_idx = [i[0] for i in unvisited_prob]
  sel_prob = [i[1] for i in unvisited_prob]

  self.next_node = random.choices(unvisit_idx, sel_prob)[0]
```

Figure 1: Exploitation and Exploration Algorithms

5 Experiments

To measure the performance, we compared it to several greedy algorithms that use varying heuristics to reach a solution. A Q-learning algorithm should outperform these greedy heuristics primarily because Q-Learning much more computationally expensive. Below are the baseline greedy algorithms used to measure the performance of the Q-learning algorithm.

Random Algorithm: Select the next city randomly.
Greedy Algorithm 1: Select the city with the highest prize as the next city in path.
Greedy Algorithm 2: Select the city with the highest prize to distance ratio as the next city in path.

Authors of the MARL algorithm were able to out preform Random Algorithm and Greedy Algorithm. However, they were only able to match, not out preform Greedy algorithm. While the authors argued that this performance demonstrates Q-learning is a promising solution to the PCTSP because it matched the best heuristic based-algorithms, the Q-Learning algorithm is much more computationally expensive than Greedy Algorithm 2. Therefore, increased performance is necessary to justify the use of Q-Learning to solve PCTSP.
We could have chosen to measure the performance of our algorithm with other RL algorithms directly as our baseline. However, this would require duplicating the environment used on those RL algorithms which could be very computationally expensive and the Q-Learning algorithm would only be viable on that environment. A more general approach was decided to use randomly generated environments and measure the performance against greedy heuristics as those are not environment dependent. This would also provide insight into whether RL algorithms are viable solutions to PCTSP compared to greedy algorithms.

6 Results and Discussion

6.1 Results

On average the Q-Learning algorithm outperformed the best greedy algorithm, Greedy Algorithm 2, by 5%. The Q-Learning algorithm outperformed the random algorithm by 96% and outperformed Greedy algorithm 1 by 92% percent. The performance percentage is calculated by the percent the Q-Learning algorithms reduced the path length of the original Greedy Algorithms.

Algorithm	Avg Path Length	Avg Q-Learning % Reduction
Random	8738029.483	96%
Greedy 1	4480759.486	92%
Greedy 2	368937.8836	5%
Q-Learning	351917.0073	0%

Table 1: Q-Learning Performance vs Greedy Algorithms, 10 Trials

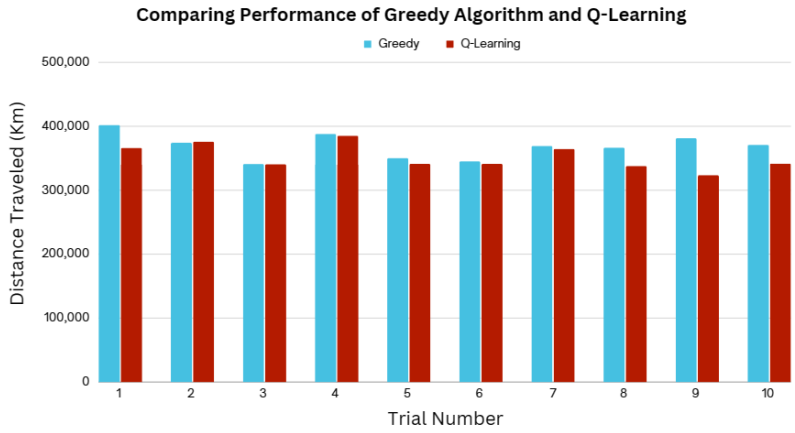

Figure 2: Greedy 2 vs Q-Learning

While on average, the Q-Learning outperforms the best greedy algorithm Greedy 2 by 5% this is not always the case. In some cases, the Q-Learning algorithm does worse than the Greedy 2 as noted in trial 2 in Figure 1. The standard deviation of the percent path reduction from Greedy 2 was 5%. The full data from the trials can be seen in the appendix at Table 2.

6.2 Discussion

It is not very notable to reduce Random and Greedy 1 algorithms path lengths as authors of the MARL algorithm were already able to beat those algorithms. However, it is notable that the Q-learning Agent outperformed Greedy Algorithm 2, as the authors of MARL were not able to outperform it. While in the 10 trials a performance increase of 5% was achieved. However, considering the standard deviation of path reduction was 5%, it is important to note that these trials could be an outlier and more trials may be needed to be more certain of the Q-Learning performance. In Trial 8, the percent path reduction is 15% which is far more significant than any of the other trials. This shows that the Q-Learning algorithm has the potential to greatly reduce the cost of Greedy 2 depending on the environment. Overall, the results show that a Q-Learning algorithm can be a viable solution to the PCTSP and shows that a Q-Learning algorithm can not only match the performance of greedy heuristics but outperform them.

7 Conclusion and Future Work

In conclusion, this paper successfully illustrates the superiority of the Q-Learning algorithm over traditional greedy heuristics in solving the PCTSP. While Q-Learning emerges as an optimal solution, it's important to acknowledge its significantly longer run-time compared to its greedy counterparts. Future research should explore this time discrepancy, assessing whether a enhanced performance justifies the increased computational cost. Moreover, the Q-Learning algorithm's performance showed some inconsistency, indicating a need for more extensive testing to reliably gauge its effectiveness. A comparative study with more advanced Reinforcement Learning (RL) strategies, such as the Ant Colony Optimization technique by Shi et al. (2008), would also be insightful. Overall, this study provides compelling evidence that even simple single-agent RL algorithms have the potential to outshine greedy heuristics in complex tasks like the PCTSP.

REFERENCES

Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19:612–636, 1989.

Daniel Bienstock, Michel X. Goemens, David Simchi-Levi, and David Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

Jannis Blauth and Martin Nagele. An improved approximation guarantee for prize-collecting tsp. *ARIV*, 52, 2023.

Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Learning to solve vehicle routing problems: A survey. *ARIV*, 2022.

Antonio Augusto Chaves and Luiz A. N. Lorena. Hybrid algorithms with detection of promising areas for the prize collecting travelling salesman problem. *IEEE Xplore*, 57, 2005.

Yifei Chen, Lambert Schomaker, and Marco Wiering. An investigation into the effect of the learning rate on overestimation bias of connectionist q-learning. *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, pp. 107–118, 2021.

Glaubos Climaco, Luidi Simonetti, and Isabel Rosetti. A branch-and-cut and mip-based heuristics for the prize- collecting travelling salesman problem. *RAIRO - Operations Research*, 2020.

Jessica Gonzalez, Zari Magnaye, Christopher Gonzalez, Yutian Chen, and Bin Tang. Budget-constrained traveling salesman problem: a reinforcement learning approach. 2023.

Yubin Liu, Qiming Ye, Jose Escribano-Macias, Yuxiang Feng, Eduardo Candela, and Panagiotis Angeloudis. Route planning for last-mile deliveries using mobile parcel lockers: A hybrid q-learning network approach. *Transportation Research Part E: Logistics and Transportation Review*, 177, 2023.

Justin Ruiz, Christopher Gonzalez, Yutian Chen, and Bin Tang. Prize-collecting traveling salesman problem: a reinforcement learning approach. 2022.

Xiaohu Shi, Liupu Wang, and You Zhou. An ant colony optimization method for prize-collecting traveling salesman problem with time windows. *Fourth International Conference on Natural Computation, Jinan, China*, 59:480–484, 2008.

Yunqiu Xu, Meng Fang, Ling Chen, Gangyan Xu, and Chengqi Zhang. Reinforcement learning with multiple relational attention for solving vehicle routing problems. *IEEE Xplore*, 52, 2022.

A APPENDIX

A.1 CODE

Codebase: https://colab.research.google.com/drive/1AFUVKmN6FXBuAYzM9P-FH9-nTPmiVCAC?usp=sharing

A.2 FULL DATA

Random	Greedy 1	Greedy 2	Q-Learning
9130077.00419814	3927709.10086674	401772.960627626	366288.38371392
8316585.52216581	4442378.78128604	374312.095055003	376097.605781432
8835136.50418411	4517321.51167021	341078.208425922	340798.644016066
9197010.33648771	4496876.91697809	388181.7274115	385288.463587806
9124736.36533417	4615006.82946925	350453.263024442	341344.509001446
8317945.3591969	4611790.40152656	345301.290240983	341344.509001446
8733806.92516164	4533199.02399837	369407.865674995	364664.477009728
8376693.16831867	4464462.00545889	366690.418179368	337988.589642611
8552463.01460457	4771507.58507088	381345.544841873	323675.373418224
8795840.62609566	4427342.7036793	370835.46203921	341679.517997641

Table 2: Comparison of Total Path Lengths Accross Algorithms Per Trial