Deep Q-Learning For The Traveling Salesman Problem

Behzad Karimi Deanta Kelly Ian Kessler Fatima Ododo

Montana State University
behzad.karimi@student.montana.edu
deanta.kelly@student.montana.edu
ian.kessler@student.montana.edu
fatima.ododo@student.montana.edu

M 508 Presentation
May 2, 2024

Introduction

- Project Focused On Implementation And Analysis Of Algorithm Described In *Learning Combinatorial Optimization Algorithms over Graphs*
- Focused On One Combinatorial Optimization (CO) Application: The Traveling Salesman Problem (TSP)
- Used Held-Karp Algorithm ($\in \mathcal{O}(2^n n^2)$) To Check Accuracy
- Acccording to the paper, classical algorithms "seldom exploit a common trait of real-world optimization problems: instances of the same type of problem are solved again and again on a regular basis, maintaining the same combinatorial structure, but differing mainly in their data."
- **Problem Statement**: Given a distribution $\mathbb{D}$ of graphs, can we learn better heuristics to solve TSP that generalize to unseen instances from $\mathbb{D}$?

Presentation Overview

1. The Traveling Salesman Problem
2. Q-Learning
3. Evaluating Q With *Structure2Vec* Neural Network
4. Data Analysis

The Traveling Salesman Problem

(On Whiteboard)

Q-Learning

```
Initialize experience replay memory M to capacity N
for episode e = 1 to L do
    Draw graph G from distribution D
    Initialize the state to empty S_i = ()
    for step t=1 to T do
        v_t = { random node v ∈ S̄_t,                 w.p. ϵ
              { argmax_{v∈S̄_t} Q̂(h(S_t), v; θ),      Otherwise
        Add v_t to partial solution: S_{t+1} := (S_t, v_t)
        if t ≥ n then
            Add tuple (S_{t−n}, v_{t−n}, R_{t−n}, S_t) to M
            Sample random batch from B ∼ M
            Update θ by SGD over (6) from B
        end if
    end for
end for
return θ
```

$$v_t = \begin{cases} \text{random node } v \in \bar{S}_t, & \text{w.p. } \epsilon \\ argmax_{v \in \bar{S}_t} \hat{Q}(h(S_t), v; \theta), & \text{Otherwise} \end{cases}$$

Structure2Vec: Describing Our Embedding

- Given partial solution S, we create a p-dimensional embedding for each node $v \in V$ and each layer i: $(\mu_S^{(i)})_v \in \mathbb{R}^{p \times 1}$
- Let $m = |V|$. Then, the i^{th} layer of our neural network is

$$\mu_S^{(i)} = [(\mu_S^{(i)})_0, (\mu_S^{(i)})_1, \ldots, (\mu_S^{(i)})_{m-1}] \in \mathbb{R}^{p \times m}$$

- To have our embedding depend on S, we use

$$x_S := [1\{v \in S\} : v \in V] \in \{0, 1\}^{1 \times m}$$

- Want our embedding to exploit the graph structure, so we define
  - The Neighbors of v to be $\mathcal{N}(v)$ ($= V \setminus \{v\}$ For Complete Graphs)
  - The Weight of Edge (v, u) to be $w(v, u)$

Structure2Vec: Calculating Hidden Layers

- Our Initial Layer: $\mu_S^{(0)} := 0^{p \times m}$
- From One Layer To The Next:

$$(\mu_S^{(i+1)})_v \leftarrow \text{relu}(\theta_1 x_S[v] + \theta_2 \sum_{u \in \mathcal{N}(v)} (\mu_S^{(i)})_u + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v,u)))$$

  Where $\theta_1 \in \mathbb{R}^p$, $\theta_2, \theta_3 \in \mathbb{R}^{p \times p}$, $\theta_4 \in \mathbb{R}^p$

- Using T hidden layers, we compute a sequence of hidden layers:

$$\mu_S^{(0)} \to \mu_S^{(1)} \to \cdots \to \mu_S^{(T)}$$

  Where $\mu_S^{(T)}$ is Final Hidden Layer

Output "Q" Layer

- Using final hidden layer, $\mu_S^{(T)}$, we use evaluation function:

$$\hat{Q}(S, v) = \theta_{5a}^{\intercal}\text{relu}(\theta_6 \sum_{u \in V} (\mu_S^{(T)})_u) + \theta_{5b}^{\intercal}\text{relu}(\theta_7 (\mu_S^{(T)})_v)$$

Where $\theta_{5a}, \theta_{5b} \in \mathbb{R}^p$, $\theta_6, \theta_7 \in \mathbb{R}^{p \times p}$

- Letting $\overline{S} := V \backslash S$, we update $S \leftarrow S + [v^*]$, where

$$v^* = \text{argmax}_{v \in \overline{S}} \hat{Q}(S, v)$$

- $\hat{Q}(S, v)$ depends on $\Theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_{5a}, \theta_{5b}, \theta_6, \theta_7]$
- The parameters Θ are learned using reinforcement learning.

4-Fold Cross Validation

- Trained and Tested Model On 8 Random Euclidean Graphs

$$[G_i : i \in [0..7]]$$

- For each graph G_i, its vertex set $V_i \in \mathbb{Z} \times \mathbb{Z}$ with $|V_i| = 9$ and

$$V_i = \{(x_{ij}, y_{ij}) : -5 \leq x_{ij}, y_{ij} \leq 5\}$$

- The Error of Each Fold:
  Avg. Error on $\frac{8}{4} = 2$ Graphs Using Θ Trained on $8 - 2 = 6$ Other Graphs
- Error for Model: Avg. Error of all Folds

Approximation Ratio As Error

- For a full solution S for graph G, let
  $$\text{cost}_G(S) = \sum_{i=0}^{|S|-2} w(S[i], S[i+1]) + w(S[|S|-1], S[0]),$$
  (The Total Weight of Tour Generated from S)
- For graph G, let ...
  - $\hat{S}_\Theta$ be a full solution found using our model with weights Θ.
  - S^* be a full solution that minimizes cost_G.
- Then, Error is the Approximation Ratio

$$\rho = \frac{\text{cost}_G(\hat{S}_\Theta)}{\text{cost}_G(S^*)} \geq 1$$

- Optimal solutions S^* were found using the Held-Karp algorithm.

Tuning Hyperparameters

- Each Deep Q-Learning Model Defined By 7 Hyperparameters
  - p : Dimension of Each Node Embedding
  - T : Number of Hidden Layers
  - ϵ : Probability of Choosing Random $v \in \overline{S}$ to Append to Partial Solution S
  - n : Number of Steps Between States For n-Step Q-Learning
  - α : Learning Rate for Gradient Descent
  - β : Maximum Size of Batches for Mini-Batch Gradient Descent
  - γ : Discount Factor For Q-Learning
- Used 2 Options For Each Hyperparameter Among Models Used
- Error Was Calculated For $2^7 = 128$ Models, Each Created From Choosing Between 2 Values For Each of The 7 Hyperparameters

Avg. Error By Hyperparameter (1)

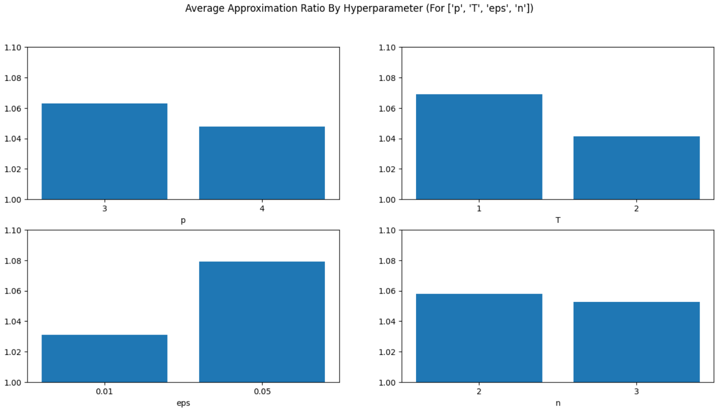

Figure: Average Approximation Ratio For p, T, ϵ, and n

Best Values are $p = 4$, $T = 2$, $\epsilon = 0.01$, and $n = 3$.

Avg. Error By Hyperparameter (2)

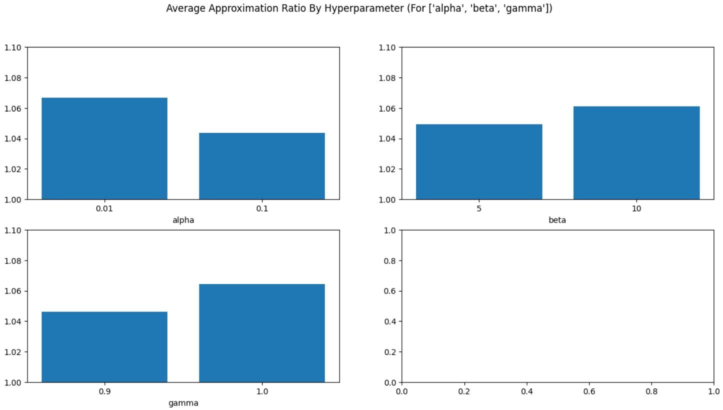

Figure: Average Approximation Ratio For α, β, and γ

Best Values are $\alpha = 0.1$, $\beta = 5$, and $\gamma = 0.9$.

Best Hyperparameters?

- Considering Hyperparameters Independently:

  According to bar graphs, the best hyperparameters are:

  $p = 4$, $T = 2$, $\epsilon = 0.01$, $n = 3$, $\alpha = 0.1$, $\beta = 5$, and $\gamma = 0.9$
- Considering Hyperparameters As A Set:
  - Using the model with the above assignment of hyperparameters, the average approximation ratio was $\rho = 1$.
  - Of the 128 models used in cross-validation, 39 of them had an approximation ratio of $\rho = 1$.
- We will consider the above assignment of hyperparameters to give us our "best" model.

Q-Learning With “Best” Model Over 3 Episodes (1st Sample, 9 Vertices Each)

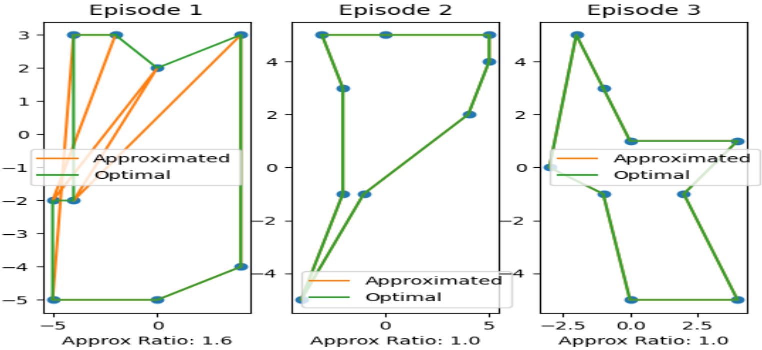

Figure: Q-Learning Over 3 Episodes (1st Sample, 9 Vertices Each)

Q-Learning With "Best" Model Over 3 Episodes (2nd Sample, 14 Vertices Each)

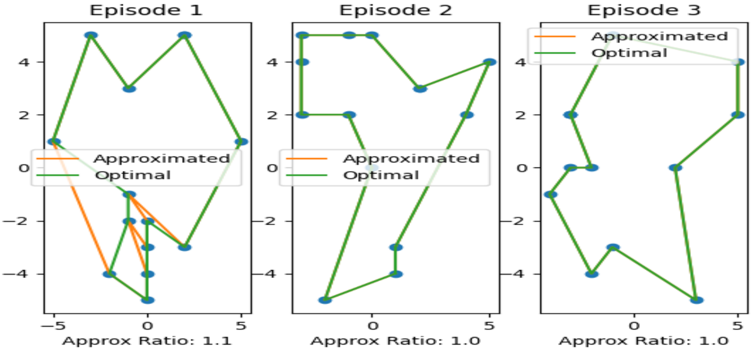

Figure: Q-Learning Over 3 Episodes (2nd Sample 14 Vertices Each)

Q-Learning With “Best” Model Over 3 Episodes (3rd Sample, 17 Vertices Each)

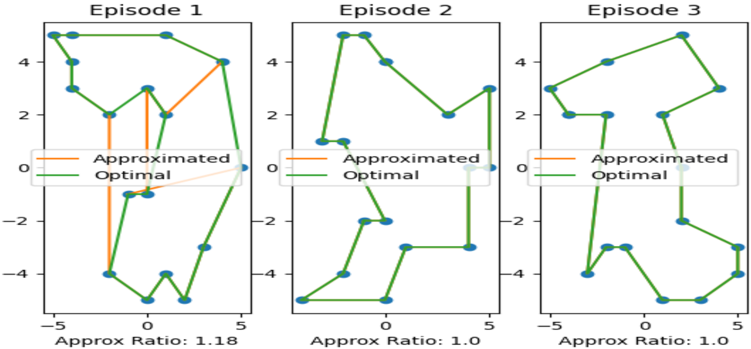

Figure: Q-Learning Over 3 Episodes (3rd Sample, 17 Vertices Each)

References

Focus Of Project:
Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, Le Song
Learning Combinatorial Optimization Algorithms over Graphs
Neural Processing Information Systems 5 April, 2017

For The Held-Karp Algorithm:
Feidiao Yang, Tiancheng Jin, Tie-Yan Liu, Xiaoming Sun, Jialing Zhang
Boosting Dynamic Programming with Neural Networks for Solving NP-hard Problems
Proceedings of Machine Learning Research 95:726-739, 2018.