Cracking the TSP: A Deep Q-Learning Approach

Mark Holcroft
STOR-i, Lancaster University

December 2024

Abstract

Since "learning" to outcompete humans in areas such as chess and the Atari games, Deep Q-Learning (DQL) has been applied to a growing number of Operational Research problems with varying success. In this report, DQL was applied to the Travelling Salesman Problem (TSP), exploring various combinations of hyper-parameters including the learning rate (α), the discount factor (γ), and the exploration rate decay (ρ). Performance was evaluated across 10 TSP instances, demonstrating that all tested configurations improved upon random solutions. The DQL configuration with ($\gamma = 0.99, \alpha = 0.001, \rho = 0.9975$) achieved the best average performance across these instances. This optimised DQL model was then compared to a tuned Genetic Algorithm (GA). While both methods successfully enhanced solution quality across all instances, the GA consistently outperformed DQL by a significant margin.

Introduction

The Travelling Salesman Problem (TSP) is a classic optimisation problem, with the aim of finding a minimal distance route between a set of nodes. The problem suffers from the combinatorial explosion problem, and as such has traditionally been solved using metaheuristics, a collection of methods developed to solve such problems efficiently and within a small margin of error.

Recently, many areas of Operational Research (OR) have benefitted from the reinforcement learning method Deep Q-Learning (DQL). Although the TSP is widely-studied, there is little literature regarding DQL's application to the problem. The aim of this report is to investigate whether DQL can be a valid tool for solving the TSP, and to what degree tuning a selection of its hyper-parameters can improve its performance, comparing the results to that of the well-established Genetic Algorithm (GA) metaheuristic.

1 Deep Q-Learning

Deep Q-Learning is an advanced version of Q-Learning (QL), a reinforcement learning method where an agent learns an optimal policy by navigating a state space and identifying the best action (a) to take at each state (s). In QL, a Q-matrix stores Q-values (rewards) for each state-action pair, updated iteratively using the Bellman's equation:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)),$$

where $Q(s, a)$ is the currently stored Q-value for the (s, a) state-action pair, α is the learning rate, r is the immediate reward, γ is the discount factor applied to future rewards, and $\max_{a'} Q(s', a')$ is the maximum future reward.

For the Travelling Salesman Problem (TSP), the state space includes visited nodes, unvisited nodes, and the current node. Note that the reward in this case is denoted by a loss, as a greater distance is undesirable. The agent selects the next node (action) using an ϵ-greedy strategy to balance exploration of the state space and exploitation of good routes. However, as the number of nodes increases, the state space grows exponentially, making the Q-matrix too large to store and the algorithm impractically slow.

DQL addresses this by replacing the Q-matrix with a neural network to approximate Q-values, significantly reducing memory requirements and improving scalability for larger TSP instances. The steps of DQL are as follows:

1. Initialise the Neural Network
    - A first, online network for learning the Q-values
    - A second, target network for storing the Q-values, updated periodically
2. Experience Replay
    - Stores experiences in a replay memory
    - Samples from the replay memory to improve learning
3. Action Selection using ϵ-Greedy Algorithm
4. Training
    - Compute new Q-value and loss
    - Update the neural network weights
5. Update the Target Network
6. Decay ϵ, up to a Predetermined Minimum Value

The above algorithm is run iteratively until a sufficient amount of space has been explored and a good policy acquired.

1.1 The Genetic Algorithm

As there is limited literature surrounding DQL usage for the TSP, we will use the Genetic Algorithm (GA) as a benchmark against which its performance can be measured. According to Toaza and Esztergár-Kiss, 2023, the GA is the most common metaheuristic to feature in published papers. This, alongside its empirical good performance, makes it suitable for use as a benchmark. We have tuned our GA as follows:

- We choose a population size of 75, following recommendations by Jong and Alan, 1975 to use a population between 50 and 100.
- We use tournament selection to choose parent solutions.

- We use a Modified Order Crossover method, taking a segment from parent A and filling in the remaining genes in the order they appear in parent B.
- The best 75 solutions are kept after each generation.
- We select a mutation rate of $1/n$, where n is the number of nodes, as the mutation rate should be proportional to the size of the problem.
- We choose the Reverse Sequence Mutation (RSM) as our mutation operator as it empirically outperforms alternatives like the 2-swap operator.

2 Tuning DQL

For the report, we code DQL in Python using the torch package by Paszke et al., 2019. We then select a few parameters of DQL to tune. For this, we chose to use the same parameters used in Sprague, 2015 when famously assessing the performance of DQL on Atari, and modify the learning rate α, the ϵ-decay rate ρ, and the discount factor γ. We also utilise a key experience buffer - only high-quality actions associated with low losses are saved to this, and are sampled from more frequently. The values of the other parameters are as follows:

- The key experience sample ratio is set to a start of 0.1 and a growth of 1.001 per iteration, with a maximum of 0.8.
- The exploration rate (ϵ) start value was set to 1 to prioritise exploration during the early stages of training.
- Following a recommendation by Mnih et al., 2013 to have a batch size proportional to the problem size, the batch size was set to 32, a standard for small datasets.

We selected values of γ close to 1 (0.9 and 0.99) as it is important in the TSP to consider the entire route. For α, we selected very small values (0.001 and 0.0001) to allow for smoother convergence. Finally, we used ρ values of 0.99 and 0.9975, chosen to be close to 1 for sufficient exploration.

3 Results

3.1 Selecting the Optimal DQL Hyper-Parameter Configuration

The DQL was run on ten relatively small TSP instances ranging between 14 and 38 nodes, all selected from The Heidelberg TSP Database and The Waterloo TSP Database. Euclidean distances between pairs of cities was used for simplicity. This is justified as, according to Boyacı et al., 2021, there is a greater than 0.98 correlation between true distances and Euclidean distances. For each instance, we ran the DQL with each combination of parameters three times and took an average of the final solution values. We also ranked the final value obtained by each algorithm, using two metrics, their average rank and average Dist, where:

$$\text{Dist}_{ij} = \frac{\text{Cost achieved by DQL Configuration } i \text{ on instance } j}{\text{Longest route achieved for instance } j}.$$

	$\gamma_1, \alpha_1, \rho_1$	$\gamma_1, \alpha_1, \rho_2$	$\gamma_1, \alpha_2, \rho_1$	$\gamma_1, \alpha_2, \rho_2$	$\gamma_2, \alpha_1, \rho_1$	$\gamma_2, \alpha_1, \rho_2$	$\gamma_2, \alpha_2, \rho_1$	$\gamma_2, \alpha_2, \rho_2$
Uly_22	157.0	150.6	130.7	**104.3**	157.7	145.9	133.5	119.8
Bur_14	38.4	45.5	43.3	**37.3**	55.7	43.1	41.9	40.9
Dji_38	15209.7	**13155.7**	19416.0	18755.2	17091.1	14662.9	19636.8	19252.6
Lux_25	4313.4	3974.6	5782.3	5898.4	6210.1	**3446.0**	5877.9	5962.5
Qat_25	4573.6	**3964.3**	5840.8	5785.5	5842.8	4463.6	5907.3	6053.0
Rwa_25	11051.4	7994.9	10955.8	11826.1	14999.0	**7519.8**	10740.0	12093.1
Uly_16	116.9	131.4	110.0	113.4	120.1	105.8	**105.1**	127.1
Uru_25	9181.0	**8058.5**	11457.8	12937.8	10797.3	9156.7	11703.2	11279.8
Zim_30	26535.3	19361.9	28214.0	25289.7	23865.0	**18488.6**	26633.8	24955.7
Sah_29	**53064.5**	55253.4	86330.0	57673.8	60337.4	57749.1	66296.5	72473.4
Avg Rank	3.8	3.2	5.4	4.3	6	2.5	5.2	5.6
Avg Rank/8	3rd	2nd	6th	4th	8th	1st	5th	7th
Avg Dist	0.78	0.723	0.895	0.842	0.913	0.709	0.866	0.892
Avg Dist/8	3rd	2nd	7th	4th	8th	1st	5th	6th

Table 1: Average cost and corresponding rank and distance for each hyper-parameter combination applied to each TSP instance.

These values are shown in Table 1. The results give us that our optimal set of hyper-parameters is $(\gamma = 0.99, \alpha = 0.001, \rho = 0.9975)$, with a close second of $(\gamma = 0.9, \alpha = 0.001, \rho = 0.9975)$. We will hence use the former to compare against the classical GA meta-heuristic. A greater exploration decay is intuitive, as the algorithm will have the opportunity to explore a greater number of routes before selecting one, and a greater learning rate could be preferable as exploration decays quickly and there are only 2000 iterations over which the algorithm can learn.

3.2 DQL vs GA Performance

To illustrate the improvements our tuned DQL is able to make, we run it on the Burma TSP instance and compare our results to those obtained using a random initial solution. The two are shown in Figure 1. DQL is clearly able to improve the random solution, with a more intuitive route and a much reduced cost. However, there remain significant improvements which can be made. Next, we plot DQL's average performance at every tenth iteration against that of GA on the 25-node Uruguay TSP instance. Each line represents an average of 20 runs of the respective algorithm, and the results are shown in Figure 2. We can see clear improvements on the initial solution for both algorithms, although the GA is more successful both in rapidly lowering the route length and achieving a lower final cost. An average random solution cost is approximately 14,270 for this problem (taking an average of 100,000 randomly generated solutions), and both methods comfortably outperform this within the first 500 iterations.

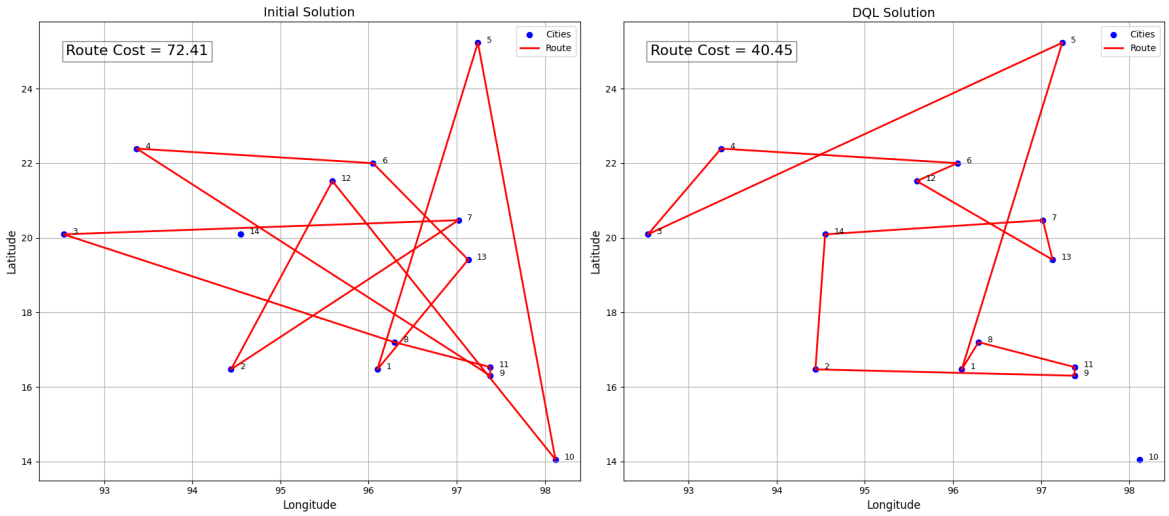

Figure 1: TSP Solution Using our DQL vs Using a Random Initial Start for the Burma TSP Instance.

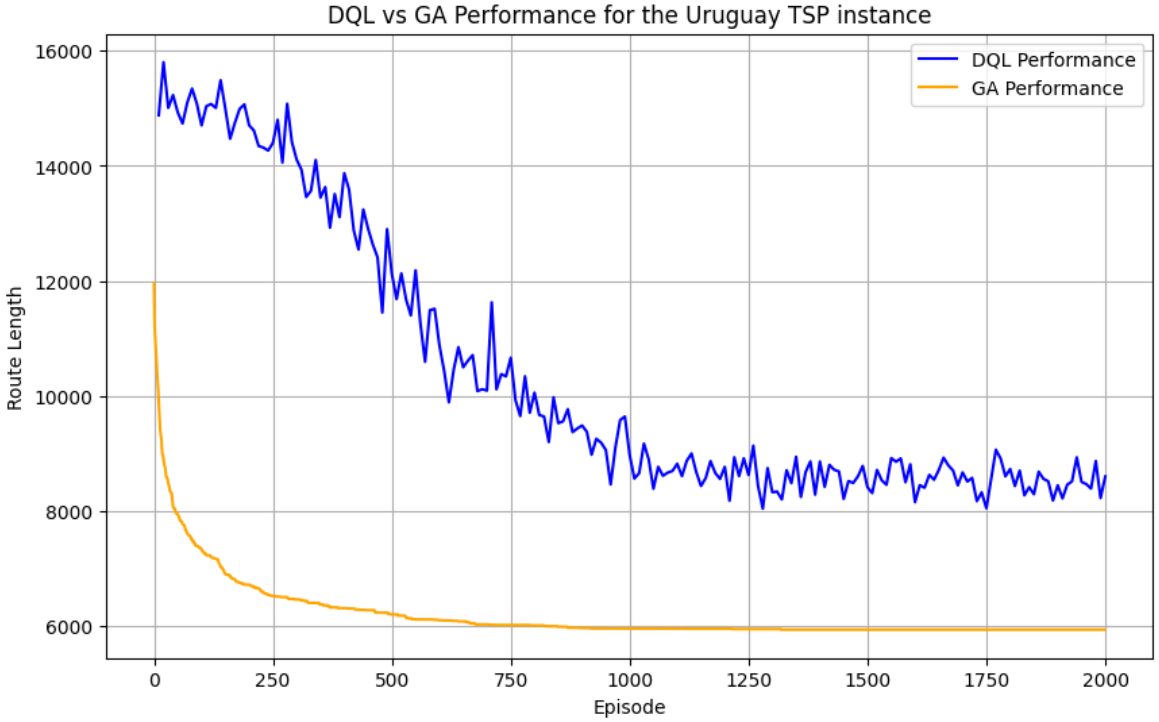

Figure 2: Average Performance of DQL vs GA over 2000 Iterations on the Uruguay TSP Instance.

4 Conclusion

To conclude, the report found that DQL can be successfully applied to a range of small TSP instances, yielding solutions of improved quality. The performance of DQL was highly responsive to adjustments to hyper-parameter selection, namely the learning rate (α), discount factor (γ), and exploration decay rate (ρ), although susceptible to poor performance under sub-optimal hyper-parameter configurations.

Despite its successes, DQL was consistently dominated both in speed and final solution value by the GA over all tested instances. This is unsurprising, with GA's dominance in the metaheuristic field being attributable to its empirical good-performance on such problems, but it does raise doubt regarding to what degree DQL would be useful for the classic TSP.

This report offers several avenues for improving DQL's performance on the TSP. Future work could involve testing more combinations of hyper-parameters, such as the learning rate (α), discount factor (γ), and exploration decay rate (ρ). Additionally, tuning other aspects of the algorithm - such as the initial exploration rate, experience replay ratio and its decay, batch size, and the neural network architecture itself (e.g., number of hidden layers) - could yield further enhancements.

While the results presented here are primarily of academic interest given the dominance of metaheuristics like GA, DQL's adaptability and ease of application justifies further exploration in other areas of OR. An area for future study could be in investigating whether DQL can achieve better outcomes in problems where metaheuristics have empirically performed poorly. With targeted improvements, DQL holds promise as a flexible and innovative tool for solving complex optimization problems.

References

Boyacı, B., Dang, T. H., & Letchford, A. N. (2021). Vehicle Routing on Road Networks: How Good is Euclidean Approximation? *Computers Operations Research*, *129*, 105197. https://doi.org/10.1016/j.cor.2020.105197

Jong, D., & Alan, K. (1975). *Analysis of the Behavior of a Class of Genetic Adaptive Systems* [Doctoral dissertation, University of Michigan]. https://deepblue.lib.umich.edu/handle/2027.42/4507

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv*, 1–9. https://doi.org/https://arxiv.org/abs/1312.5602

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*, 8024–8035.

Sprague, N. (2015). Parameter Selection for the Deep Q-Learning Algorithm. *Proceedings of the Deep RL Workshop at NIPS*. https://api.semanticscholar.org/CorpusID:29185036

Toaza, B., & Esztergár-Kiss, D. (2023). A Review of Metaheuristic Algorithms for Solving tsp-based Scheduling Optimization Problems. *Applied Soft Computing*, *148*, 110908. https://www.sciencedirect.com/science/article/pii/S1568494623009262