Journal of Artificial Intelligence Research 1 (1993) 1-15 Submitted 9-3; published x

Optimization of Travel Itineraries using Multi-Agent Deep Q-Networks (MADQN): The Qtrip Approach

Alessandro Barro ALESSANDRO1.BARRO@MAIL.POLIMI.IT
POLITECNICO DI MILANO,
PIAZZA LEONARDO DA VINCI 32, MILAN 20133 IT

Abstract

Deep Q-Networks (DQN) have revolutionized the reinforcement learning landscape by using neural networks to approximate Q-value functions, addressing tasks with large state and action spaces. One of the most challenging computational optimization problems in computer science, especially in the realm of travel, is the Traveling Salesman Problem (TSP). This study introduces the Qtrip algorithm, which leverages DQNs to optimize travel itineraries. Unlike traditional methods, Qtrip can accommodate an infinite range of user-specific preferences, offering highly personalized travel routes. By integrating a multi-agent system, we tap into the collective intelligence of Multi-Agent Deep Q-Networks (MADQNs). Aggregating insights from multiple agents during training, the Qtrip algorithm improves robustness and accuracy in itinerary optimization. Our initial results highlight the potential of MADQNs not just for reimagining TSP formulations but also for innovating adaptive, user-focused travel optimization techniques in modern computing.

1. Introduction

Reinforcement learning has undergone remarkable advancements in recent years, with Deep Q-Networks (DQN) standing out as a pivotal breakthrough. DQNs harness the capabilities of neural networks to approximate Q-value functions, addressing intricate tasks with vast state and action domains. A paramount challenge in computational optimization, especially in computer science applications tied to travel, remains the Traveling Salesman Problem (TSP). Specifically, this paper explains the MADQN methodology and also draws a comparison between this and Prize Collecting TSPs, elucidating their mutual emphasis on high-reward path finding.

Driven by the pursuit of more adaptable, user-focused solutions, we present the Qtrip algorithm. This innovative method employs DQNs to refine travel itineraries, extending the flexibility to encompass an extensive, theoretically infinite range of user preferences. In stark contrast to traditional techniques, this empowers users with bespoke travel plans tailored meticulously to their distinct preferences.

Pushing the boundaries of innovation, we weave in the essence of multi-agent systems, tapping into the collective strength of Multi-Agent Deep Q-Networks (MADQNs). By fostering collaboration among multiple agents, the Qtrip algorithm amasses a wealth of shared insights, promising heightened robustness and accuracy in itinerary optimization.

The core contribution of this paper is the unveiling of the Qtrip approach, a paradigm shift in adaptive, user-centric travel optimization. We unravel its conceptual underpinnings, operational mechanisms, and the preliminary results acquired. The evidence suggests that

MADQNs hold immense potential, not merely as an alternative to conventional TSP solutions, but as flag-bearers of next-gen methodologies in today's computational landscape.

2. Technical foundations of MADQNs in Qtrip

Multi-Agent Deep Q-Networks (MADQNs) extend the foundational principles of DQNs to contexts involving multiple agents interacting concurrently with the environment. This section elucidates the technical underpinnings of the MADQN algorithm, contrasting foundational elements from both Markov Decision Processes (MDPs) and MADQNs, thereby offering insights into the design choices made in this research.

2.1 State Representation

State representation in reinforcement learning crucially determines the agent's perception and comprehension of the environment. The challenge lies in devising state definitions that are both representative and unequivocal.

Let's define our set of cities as $\Omega = \Omega^{(1)}, \ldots, \Omega^{(C)}$ with a corresponding discrete time value $t \in \mathbb{N}$. For simplicity in notation, cities might often be referred to by their index c, circumventing the explicit use of $\Omega^{(c)}$.

Our state, denoted as $s \in \mathcal{S}$, is a one-hot encoded vector with the dimensionality of $\mathcal{S}$ given by:

$$\dim(\mathcal{S}) = \dim(\Omega) \times (C + 1) \tag{1}$$

In this structure, $s^{(c)^*}$ acts as a binary indicator reflecting the agent's presence in the c^{th} city:

$$s = (s^{(1)}, \ldots, s^{(c)^*}, \ldots, s^{(C)}, t) \tag{2}$$

This approach unequivocally pinpoints the agent's location at any instant, bolstering efficient learning and decision processes.

A pivotal aspect is the association of each state $s^{(c)}$ with its respective city $\Omega^{(c)}$, reducing computational strain posed by the variable nature of $\Omega^{(c)}$.

We introduce the state-to-city mapping function Ψ:

$$\Psi^{(c)} : s^{(c)} \mapsto \Omega^{(c)} \tag{3}$$

This function becomes indispensable when doing every kind of computation involving the c^{th} city features (explained in the 'Reward calculation' section), which are much more manipulable when they are in vector form).

2.2 Action Formulation

In MADQNs, especially in multi-agent environments, the configuration of each agent's action space profoundly shapes the inter-agent dynamics. Within the Qtrip framework, agents symbolizing travelers have two choices: persist in their present city or transition to a different one within the dataset. For an agent currently positioned in city c, the suite of feasible actions, $\mathcal{A}^{(c)}$, is expressed as:

$$\mathcal{A}^{(c)} = \{a^{(1)}, a^{(2)}, \ldots, a^{(C)}\} \tag{4}$$

Here, each action $a^{(c)}$ denotes the choice to either stay in or move to the city c^*.

The efficiency of action selection, pivotal in both MADQNs and single-agent DQNs, heavily influences the learning dynamics and the network's eventual success. Among the myriad action-selection schemes, the epsilon-greedy strategy reigns as a prevalent choice.

Striking a balance between exploration (seeking new paths) and exploitation (leveraging known paths), the epsilon-greedy method dictates action selection probabilities. The probabilities P for selecting a stochastic action versus the greedy one are articulated as:

$$a_{t,i}^{(c)} = \begin{cases} \text{random}(\mathcal{A}^{(c)}) & P(a_{t,i}^{(c)}|s^{(c)}) = \epsilon \\ \arg\max_{a \in \mathcal{A}^{(c)}} Q(s^{(c)}, a_{t,i}^{(c)}) & P(a_{t,i}^{(c)^*}|s^{(c)}) = 1 - \epsilon \end{cases} \tag{5}$$

In this context, $a_{t,i}^{(c)^*}$ signifies the optimal action at time t, contingent upon the model parameters $Q^{\pi}(s^{(c)}, a_{t,i}^{(c)})$. These parameters elucidate the Q-value estimate for a specific state s and action a in city c, executed by agent i.

2.3 Reward Calculation

The reward mechanism within the Markov Decision Process (MDP) sets Reinforcement Learning apart from the PCTSP methods. Even though states are represented as one-hot vectors, their indices map to a specific object, $\Omega^{(c)}$, which possesses unique attributes. In the Qtrip context, these are nodes representing cities:

$$\Omega^{(c)} = \left\{ \begin{pmatrix} g_x^{(c)} \\ g_y^{(c)} \end{pmatrix}, \{\lambda_1^{(c)}, \ldots, \lambda_K^{(c)}\}, c \right\} \tag{6}$$

Here, for city $g \in \mathbb{R}^2$ provides geolocation, $c \in \mathbb{N}$ serves as an index, and $\lambda \in \mathbb{R}^K$ denotes a K-dimensional property vector. For simplicity, this structure of λ can be illustrated using a linear approach:

$$\lambda^{(c)} = (\text{history} : \lambda_1^{(c)}, \text{food} : \lambda_2^{(c)}, \ldots, \text{N} : 1, \text{E} : 0, \text{W} : 1, \text{S} : 0)^T \tag{7}$$

The weight W of city (c), contingent on tensor λ, is:

$$W(\Omega^{(c)}) = \frac{\sum_{j=1}^{K} \lambda_j^{(c)}}{\sum_{l=1}^{C} \sum_{m=1}^{K} \lambda_m^{(l)}} \tag{8}$$

This weight reflects the city's significance, derived by comparing its attributes with all other cities, and thus offers a dynamic reward mechanism influenced by myriad factors.

However, for effective modeling, the tensor λ must retain consistency across cities and datasets. A failure to uphold this could lead to model overfitting and hampered generalization. The normalization step fortifies the agents' capacity to consistently assign rewards across different datasets.

Our reward system embraces both the allure of a city and the journey costs. Two penalty constituents are introduced:

1. **Distance Penalty**: To deter extended travel between cities and prompt nearby exploration. Using cities $\Omega^{(c_1)}$ and $\Omega^{(c_2)}$, their distance is computed through the squared Haversine formula:

$$d(\Omega^{(c_1)}, \Omega^{(c_2)}) = \left(2R \arcsin\left(\sqrt{\text{Hav}(\Delta\text{X}) + \cos(g_x^{(c_1)})\cos(g_x^{(c_2)}) \times \text{Hav}(\Delta\text{Y})}\right)\right)^2 \tag{9}$$

With variables defined as in the original text.

2. **Stay Penalty**: To deter excessive dwelling within a city, ensuring expansive exploration:

$$p(\Omega^{(c)}, K) = D(\Omega^{(c)}) \cdot K \tag{10}$$

With the transition from state s to s' via action a, our reward function $R(s'|s, a)$ becomes:

$$R(s'|s, a) = \left[W(\Omega^{(c)}) - d(\Omega^{(c)}, \Omega^{(c^*)}) - p(\Omega^{(c)}, \theta)\right] \tag{11}$$

$$\forall i, j$$

Our reward framework is meticulously designed to mirror an agent's journey through the Qtrip environment. By accounting for city characteristics and movement dynamics, we offer a holistic strategy adept at navigating varied datasets, substantiating our reinforcement learning method's edge over traditional PCTSP techniques.

2.4 Parallelized Training Approach

Training models in environments with multiple agents can be computationally intense and time-consuming. Leveraging parallelism helps to efficiently gather experiences and optimize models. Notably, unlike the traditional PCTSP, this model is capable of learning to recognize patterns across varying datasets, providing a dynamic edge in addressing complex scenarios.

In a multi-agent domain, the Q-value for an agent isn't merely a result of its actions, but also of the actions of other agents within the environment. The multi-agent Deep Q-Network (MADQN) Bellman equation, which lays the foundation for the Q-value update rule, is given by:

$$Q^{\pi}(s, a_i) = \mathbb{E}_{a' \sim \pi} \sum_{i=1}^{A} [r(s, a_i, a_{-i}) + \gamma Q^{\pi}(s', a'_i, a'_{-i})] \tag{12}$$

Where the expectancy is taken over the conjoined policy π obtained from evaluating multiple agents $i = 1, \ldots, A$. Here, a_i represents the action taken by the i^{th} agent, while a_{-i} denotes the actions chosen by all other agents.

2.4.1 Proof: from the Bellman equation to the MADQN update rule

Expanding on the foundation laid by the Bellman equation, we delve into the derivation of the MADQN update rule. Initially, we focus on the Q-values ascertained by the i^{th} agent, subsequently integrating them collectively.

$$Q^{\pi}(s, a_i) = \mathbb{E}_{a' \sim \pi}[r(s, a_i, a_{-i}) + \gamma Q^{\pi}(s', a'_i, a'_{-i})] \tag{13}$$

The TD error for agent i is:

$$\delta = r + \gamma \max_{a'_i} Q^{\pi}(s', a'_i, a'_{-i}) - Q^{\pi}(s, a_i, a_{-i}). \tag{14}$$

Using the TD error, the Q-value for agent i is updated as:

$$Q_i^{\pi}(s, a_i, a_{-i}) \leftarrow Q_i^{\pi}(s, a_i, a_{-i}) + \alpha\delta. \tag{15}$$

Here, $\alpha \in \mathbb{R}$ is the learning rate. Substituting the value of δ from above, we get:

$$Q_i^{\pi}(s, a_i, a_{-i}) \leftarrow Q_i^{\pi}(s, a_i, a_{-i}) + \alpha \left[r + \gamma \max_{a'_i} Q_i^{\pi}(s', a'_i, a'_{-i}) - Q_i^{\pi}(s, a_i, a_{-i}) \right]. \tag{16}$$

By joining all actions, hence evaluating multiple agents $i = 1, \ldots, A$, we get

$$Q_i^{\pi}(s, a_i, a_{-i}) \leftarrow \sum_{i=1}^{A} Q_i^{\pi}(s, a_i, a_{-i}) + \alpha \big[r + \gamma \max_{a'_i} Q_i^{\pi}(s', a'_i, a'_{-i}) - Q_i^{\pi}(s, a_i, a_{-i}) \big]. \tag{17}$$

Which corresponds to the final MADQN update rule for Q-values. The convergence of this formula hinges on the nature of interactions between agents and the exploration strategies they employ. □

At first, data is collected by iterating each agent over episodes $E = (s, a, s')$ and days t in parallel threads. It sets up the training, clears previously visited cities (for previous episodes), and initializes the total time. For each day within the total time, the system extracts the current state, selects an action, and updates the state based on the selected action.

After data collection, the model is trained using aggregated experiences. It is looped over the experiences, pushing them to a replay buffer and subsequently training the Q-network using those experiences extracted from this buffer.

Then training is parallelized among multiple agents. The experiences are gathered in batches, with partial training optionally executed after each batch. At the end of this accumulation, the system completes training with all experiences.

Finally, the Deep Q-network is trained and its weights are updated. A replay buffer, which stores experiences to allow for more efficient and randomized training, is utilized. The function samples experiences from the replay buffer and processes states. Using the Q-values of the initial states and the target Q-values from next states, it computes the Q-values for the actions taken. The Q-network is then trained using these values.

In our approach, agents operate independently from each other. This design choice is not just arbitrary, but stems from the potential advantages that independent exploration brings to the table. Having independent agents can significantly enhance the exploration of the state space, especially when leveraging exploration techniques such as epsilon-greedy. Given their individualized randomness, each agent might chart distinct trajectories in the state space, consequently providing a mosaic of experiences. Such diversity can be instrumental when training the Q-network, offering a wide panorama of transitions and scenarios.

However, like any approach, this too comes with its set of trade-offs. We outline some of the primary advantages and potential pitfalls of this method:

- Diverse Exploration: Independent agents naturally offer varied exploration strategies. This is invaluable in environments characterized by vast state spaces or where multiple optimal solutions are feasible.
- Redundancy: On the flip side, independence can also breed redundancy. It's possible for multiple agents to traverse similar regions of the state space or replicate the same errors. This redundancy, however, can be counteracted with sufficiently large agent populations or by introducing nuanced variations in each agent's behavior.
- Coordinated Exploration: In contrast, cooperative agents, when designed well, can synchronize their exploration. This can potentially lead to quicker convergence towards optimal or near-optimal solutions. Yet, this comes with its own pitfalls. There's a risk of these agents homogenously leaning towards certain strategies and getting entrenched in suboptimal solutions.
- Complexity: Crafting cooperative strategies usually comes with its baggage - the complexity increases both in algorithm design and in computational requirements.

Given the challenge at hand unraveling optimal itineraries, our inclination towards independent agents seems apt. It allows the system to wade through a myriad of potential itineraries and strategies, ensuring that the exploration isn't prematurely skewed towards any specific route.

2.5 Exploration Strategies and Decay Mechanisms

The trade-off between exploration (trying new actions) and exploitation (using known actions to maximize rewards) is critical. Balancing this trade-off is often achieved by adjusting the exploration rate, often denoted as ϵ, throughout the learning process. Over time, the exploration rate is typically decayed to promote more exploitation as the Q-values become more accurate. The chosen strategy for this decay plays a pivotal role in the agent's learning efficiency and effectiveness.
Among various decay strategies, logarithmic decay stands out due to its nature of reducing ϵ rapidly initially, and then decelerating over time. Mathematically, it can be represented as:

$$\epsilon_t = \frac{c}{\log(t+d)} \tag{18}$$

Where t is the current episode or time step, and c and d are constants chosen to adjust the rate and initial value of ϵ. This strategy offers a rapid initial exploration, ensuring a broad sweep of the state-action space, followed by a progressively deeper exploitation as the agent refines its policy.
Another prominent method, entropy decay, focuses on the uncertainty of the agent's policy. By measuring the entropy of the policy, we gain insights into its randomness. A higher entropy signifies more randomness, indicating exploration, while lower entropy leans towards exploitation. The aim is to decay this entropy over time to transition from exploration to exploitation.

The entropy H of a policy $\pi(a|s)$ can be calculated as:

$$H(\pi) = -\sum_a \pi(a|s) \log \pi(a|s) \tag{19}$$

Entropy-based strategies are adept at promoting structured exploration, taking advantage of the inherent uncertainty in the policy.

2.6 Policy Obtaining

Determining an optimal policy is at the heart of the reinforcement learning framework. In essence, a policy dictates the mapping from states to actions with the goal of maximizing the cumulative reward over time as an agent interacts with its environment.
For every state s and corresponding action a, the Q-value, denoted by $Q(s, a)$ or $Q^{\pi}(s, a)$ when associated with policy π, encapsulates the anticipated cumulative reward from taking action a in state s and thereafter following the optimal policy. The best action a^* for a given state s can be deduced by:

$$a^* = \arg\max_{a} Q^{\pi}(s, a) \tag{20}$$

This essentially picks the action which yields the highest Q-value for the given state.
In complex multi-agent settings, interactions among agents can profoundly shape the resulting policy. For any agent i, given its action a_i and the simultaneous actions of other agents a_{-i}, the optimal joint action at a time-step t is described by:

$$a_{t,i}, a_{t^*,-i} = \arg\max_{a_{t,i}, a_{t,-i}} Q_i^{\pi}(s, a_{t,i}, a_{t,-i}) \tag{21}$$

The multifaceted nature of multi-agent scenarios must be underscored. Here, an individual agent's decisions can have nuanced effects on the rewards and choices of other agents.

For a more tailored and user-centric policy, reward reshaping can be employed. This technique modifies the original Q-values based on user-specific preferences. Given the initial Q-values, we can reshape them using the following:

$$Q^*(s, a_{t,i}, a_{t,-i}) = Q_i^{\pi}(s, a_{t,i}, a_{t,-i}) \cdot \theta \tag{22}$$

Where θ denotes user preferences for each city. By influencing the resultant Q-values, θ steers the agent's policy, ensuring it better mirrors user desires and objectives.

3. Experiments and Results

3.1 Approach and Dataset Description

We conducted a total of three experiments. For each experiment, we generated $t = 10$ itineraries for both the training and validation datasets. For the test set, we considered $t = 3, 5$ and 7 trips (for each experiment respectively). This experimental setup was designed to comprehensively evaluate the model's capacity to effectively learn and construct diverse itineraries tailored to varying day trip durations. Specifically, we were keen to observe whether the model could appropriately prioritize cities based on their reward values. An essential aspect of our evaluation was to determine if the model could discern and possibly exclude cities that fell outside the desired reward boundaries. Such an ability would ensure that the model consistently recommends high-quality itineraries, even for shorter day trips.

The dataset utilized in this study is sourced from three distinct geographical regions: Japan (training set), Italy (validation set), and France (test set). This choice was motivated by the diverse cultural and geographical landscapes these countries offer, allowing us to test the adaptability and generalization capabilities of our model across varied environments. Each dataset comprises the following attributes for each city:

- **Name**: Denotes the city name.
- **Latitude and Longitude**: Geographical coordinates pinpointing a city.
- **Attributes**: Provides a dictionary-like structure, assigning ratings for aspects like culture, food, nature, among potential others.
- **Population**: The total population of the city.
- **ID**: A unique index for the model.

3.2 Model and Training Details

Our Multi-Agent Deep Q-Network (MADQN) was implemented with a multi-layer architecture, details of which are provided in the supplementary section. For training our model, we employed the Adam optimizer with a learning rate of 0.01 (details in supplementary material), a batch size of 64, and trained for 2000 epochs. The chosen parameters highlight the network architecture's simplicity, making it easily reproducible. Furthermore, these parameters are commonly used in reinforcement learning scenarios.

3.3 Evaluation Metrics

The primary evaluation metric utilized is the absolute difference between the target Q-values (calculated through the Q-network training) and the Q-values resulting from the obtained policy π^*. This is further defined as Λ^Q, the Thresholded Q-value Precision (TQP).

$$\Lambda^Q(Q_i^{\text{target}}, Q_i^{\pi^*}, \theta) = \left(\sum_{t=1}^{T} \mathcal{X} \left[\sum_{i=1}^{N} |Q_i^{\text{target}} - Q_i^{\pi^*}| \leq \theta \right] \right) Z^{-1} \tag{23}$$

Here $\mathcal{X}$ is the indicator function, θ is the threshold parameter (arbitrary), Z is a normalizing constant (typically the total number of Q-values among all agents, and for all time steps $1, \ldots, T$). This function essentially works as a Q-value weighted average, counting the examples where the difference between Q_i^{target} (obtained in q-network training) and $Q_i^{\pi^*}$ (given from following the optimal policy π^*) exceeds an arbitrary constant θ. The counted examples will be normalized on the total number or examples Z.

The Q-MSE loss was chosen for this task for its capability to measure the square of differences between our estimated Q-values and the ground truth.

$$J(Q_i^{\text{target}}, Q_i^{\pi^*}) = \sum_{t=1}^{T} \sum_{i=1}^{N} \left(Q_i^{\text{target}} - Q_i^{\pi^*} \right)^2 \tag{24}$$

It's essential to recognize that an effective policy (or itinerary) cannot be solely evaluated through metrics. Human judgment plays a critical role in the model's assessment process.

Good metrics don't always correlate with desirable itineraries, especially considering the subjective nature of travel experiences. This will be deepend later in the 'Discussion of the obtained results' section.

3.4 Results

3.4.1 1^{st} experiment: no user preferences

In our inaugural experiment, we intentionally omitted user preferences from the model's input parameters and set the test policy length as 7. This design choice was made to study the model's intrinsic behavior when presented with neutral scenarios. By doing so, we aimed to gauge the model's baseline performance and its ability to make decisions without any user-biased influence. Let's take a look at the outputs

The itinerary in Japan (training set):
- Day 1: Tokyo
- Day 2: Tokyo
- Day 3: Tokyo
- Day 4: Yokohama
- Day 5: Shizuoka
- Day 6: Nagoya
- Day 7: Nagoya
- Day 8: Kyoto
- Day 9: Osaka
- Day 10: Osaka

The itinerary in Italy (validation set):
- Day 1: Rome
- Day 2: Rome
- Day 3: Rome
- Day 4: Rome
- Day 5: Florence
- Day 6: Florence
- Day 7: Bologna
- Day 8: Venice
- Day 9: Venice
- Day 10: Venice

The itinerary in France (1^{st} test set):
- Day 1: Paris
- Day 2: Paris
- Day 3: Paris
- Day 4: Tours
- Day 5: Nîmes
- Day 6: Chamonix
- Day 7: Chamonix

3.4.2 2^{nd} experiment: user preferences

For our subsequent experiment, we delved into the realm of user preferences. Specifically, we assigned a score of 5—the highest possible rating—to city tags representing 'relax' and 'seaside'. This decision was driven by our interest in understanding how strong user inclinations towards relaxation and coastal experiences influence the model's decision-making. It's essential to emphasize that these user preference settings were solely used to shape the policy construction within the test set. This approach allowed us to assess the model's adaptability and responsiveness to explicit user preferences. The 2^{nd} test itinerary lenght was set to 5.

The itinerary in Japan (training set):
- Day 1: Tokyo
- Day 2: Tokyo
- Day 3: Tokyo
- Day 4: Tokyo
- Day 5: Shizuoka
- Day 6: Shizuoka
- Day 7: Nagoya
- Day 8: Nagoya
- Day 9: Kyoto
- Day 10: Osaka

The itinerary in Italy (validation set):
- Day 1: Rome
- Day 2: Rome
- Day 3: Rome
- Day 4: Florence
- Day 5: Florence
- Day 6: Bologna
- Day 7: Bologna
- Day 8: Padua
- Day 9: Venice
- Day 10: Venice

The itinerary in France (2^{nd} test set):
- Day 1: Nice
- Day 2: Nice
- Day 3: Nice
- Day 4: Cannes
- Day 5: Cannes

3.4.3 3^{rd} experiment: no user preference, short trip

In our final experiment, we sought to evaluate the model's capabilities under tight time constraints, absent any user preferences. Specifically, we chose a brief itinerary duration of

$t = 3$ to simulate scenarios where travelers have limited time, such as a short layover or a brief city stopover. By doing so, we aimed to understand how effectively the model can prioritize and recommend destinations when time is of the essence, ensuring travelers get the most out of their short visits.

The itinerary in Japan (training set):
- Day 1: Tokyo
- Day 2: Tokyo
- Day 3: Tokyo
- Day 4: Yokohama
- Day 5: Shizuoka
- Day 6: Nagoya
- Day 7: Nagoya
- Day 8: Kyoto
- Day 9: Nara
- Day 10: Osaka

The itinerary in Italy (validation set):
- Day 1: Rome
- Day 2: Rome
- Day 3: Rome
- Day 4: Rome
- Day 5: Florence
- Day 6: Padua
- Day 7: Venice
- Day 8: Venice
- Day 9: Milan
- Day 10: Milan

The itinerary in France (3^{rd} test set):
- Day 1: Paris
- Day 2: Paris
- Day 3: Paris

We provide the loss graphs of the training processes (X='Episodes', Y='Loss') below. The displayed graphs exhibit a pronounced alignment, underscoring the model's consistent stability throughout the episodic iterations. Initially, the loss function J displays a somewhat stochastic behavior, exploring a variety of routes. However, as the iterations progress, it converges to values approaching 0 This convergence is indicative of the model's focus on the optimal itineraries highlighted earlier. Such behavior emphasizes the model's potent ability to discern and learn a vast array of optimal policies. The range and versatility of these policies, albeit within certain boundaries and inherent subjectivity, are further echoed by the variance exhibited by J on the Y axis.

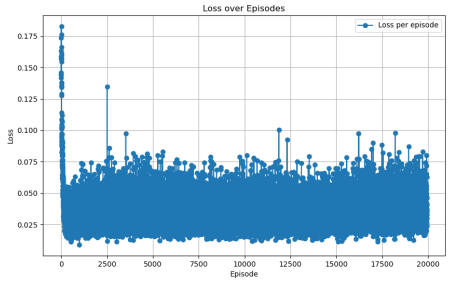

Figure 1: Losses on the 1^{st} experiment

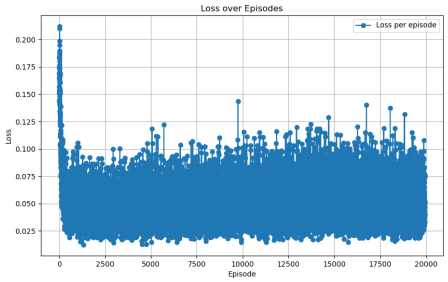

Figure 2: Losses on the 2^{nd} experiment

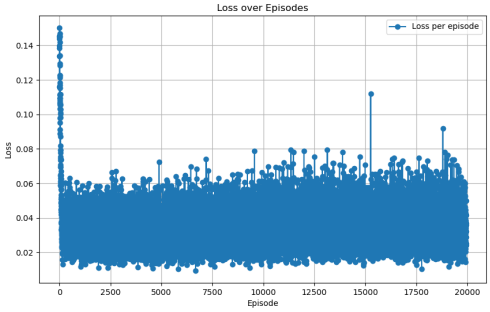

Figure 3: Losses on the 3^{rd} experiment

The obtained accuracy Λ^Q after three runs with changes to the model's parameters in the optic of the aforementioned experiments is shown below.

#	train	validation	test
1	0.91	0.90	0.85
2	0.87	0.86	0.82
3	0.84	0.84	0.79

$$\Lambda^Q_{\text{train, validation}} \approx 87\% \tag{25}$$

$$\Lambda^Q_{\text{test}} \approx 82\% \tag{26}$$

The slight accuracy drop in the test set (approximately 5%) indicates the robustness of our model. It showcases that the model is neither overfitting the training data nor is it too generalized to capture the nuances in the validation data.

From these experiments, our MADQN not only converges to an optimal solution but also demonstrates adaptability across diverse datasets. Given the versatility and efficiency of the approach, there's evident potential for real-world applications, especially in dynamic environments.

4. Discussion of the obtained results

Reinforcement learning offers a vast and varied landscape of methodologies and models. In this milieu, our proposed model stands out due to its adaptive and learning-centric approach. Unlike traditional combinatorial optimization challenges, such as the Prize Collecting Traveling Salesman Problem (PCTSP), our approach synergizes the classic goal of reward maximization with the adaptability intrinsic to data-driven models.

A direct comparison between Qtrip and the PCTSP system, while tempting, is not entirely justifiable given their disparate foundations. At its core, Qtrip is a reinforcement learning model, thriving on iterative learning. This iterative nature enables Qtrip to perpetually hone its decision-making, drawing from accumulated data and iterative feedback. The PCTSP, on the other hand, is not grounded in reinforcement learning, leading it to potentially emphasize different methodologies or operate under unique constraints.

It's noteworthy that while there are similarities in how both models handle reward collection, their operational strategies diverge significantly. PCTSP, for instance, provides a deterministic, one-off solution. In contrast, the strength of our model lies in its iterative learning and adaptability, as evidenced by its nuanced weighting of cities to curate itineraries that resonate with their significance and relevance. A case in point is the model's adeptness at emphasizing pivotal cities during shorter trips.

Interestingly, the observed 5% decrement in test set accuracy is not a liability. Instead, it underscores the model's robustness. Such a minimal disparity between training and testing performances suggests a model that's adeptly calibrated and steers clear of overfitting pitfalls.

In summation, the value of our model is anchored in its adaptability, its prowess at learning, and its broader practical ramifications. Its successful generalization across a spectrum of datasets intimates its potential competence with even larger and intricate datasets. This success not only validates its present applicability but also illuminates potential trajectories for future exploration and research.

5. Challenges

While our model exhibits numerous strengths, it is essential to delineate the challenges and intricacies intrinsic to its design and operation. One of the primary challenges is the delicate balancing act required in fine-tuning the reward parameters. An itinerary that scores well metrically doesn't necessarily translate to a practically good itinerary. This disconnect can stem from the nuanced nature of travel, where subjective preferences often clash with objective metrics.

Being a model-based approach, Qtrip also introduces a level of indeterminacy, setting it apart from deterministic solutions like PCTSP. Such indeterminacy means that Qtrip can, at times, exhibit biases towards specific itineraries, necessitating re-training. This is a double-edged sword: while it allows for adaptability, it can also lead to inconsistent results, especially if the model becomes overly attuned to certain patterns in the training data.

Another challenge is the model's treatment of distance and stay penalties. Unlike traditional constraint-based approaches, our model treats these as penalties rather than fixed constraints. This flexibility can sometimes lead to the model "miscalculating" or even over-

looking these penalties. A case in point is the observed behavior in the second experiment. During validation, the policy, instead of taking a more nuanced path, made a direct switch from Venice to Milan. Such behaviors, while efficient on paper, might not always align with practical or desired travel sequences.

These challenges underscore the need for continuous refinement and validation, ensuring that the model's decisions resonate with real-world travel dynamics and user preferences.

6. Conclusions

In the realm of itinerary optimization, traditionally anchored by algorithms such as the Traveling Salesman Problem (TSP), our exploration has led to the evolution of the Qtrip algorithm, marking a pivotal transition. With the integration of Deep Q-Networks (DQN), Qtrip doesn't merely optimize itineraries but brings in the essential aspect of personalization, honoring user-defined preferences. This nuance revolutionizes the space, offering a stark divergence from conventional methods and carving out avenues for truly tailored travel experiences.

Grounded in reinforcement learning, Qtrip showcases the expansive potential of these techniques. Our exploration juxtaposing Multi-Agent Deep Q-Networks (MADQNs) with the Prize Collecting TSP has been enlightening. Both approaches might share the objective of maximum reward accrual, yet Qtrip, powered by MADQNs, excels in adaptability, proving its mettle particularly in volatile, ever-evolving scenarios. Beyond mere itinerary optimization, Qtrip taps into pattern discernment and continual learning, championing a novel approach.

However, like any pioneering effort, our work faced challenges, especially when navigating the complexities of multi-agent dynamics and integrating diverse user preferences. Yet, these challenges also pave the way for future enhancements, offering opportunities to refine the Qtrip mechanism further.

In summation, this endeavor not only enriches the repository of travel optimization techniques but advocates fervently for adaptive, user-centric solutions. As we navigate the future trajectory of travel planning, Qtrip stands as a beacon, heralding an era where technology synergizes with personal inclinations, curating journeys that echo individual dreams and yearnings.

7. Acknowledgments

We would like to extend my gratitude to my professor, Anand Avati, who introduced us to the captivating world of machine learning during a course at Stanford University. His teachings laid the groundwork for the methodologies employed in this research. Last but not least, we are grateful to the broader scientific community whose pioneering research in the domains of Reinforcement Learning and Optimization laid the foundation for our work.

8. References

References

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529-533.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Lowe, R., Wu, Y., Tamar, A., et al. (2017). *Multi-agent actor-critic for mixed cooperative-competitive environments*. In Advances in neural information processing systems (pp. 6379-6390).

Busoniu, L., Babuska, R., & De Schutter, B. (2008). *A comprehensive survey of multiagent reinforcement learning*. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38(2), 156-172.

Barro, A. (2023). *Reinforcement Learning*. In CS229: Machine Learning, Avati A. (Lecturer), Stanford University. Unpublished lecture notes.

Watkins, C. J., & Dayan, P. (1992). *Q-learning*. Machine learning, 8(3-4), 279-292.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1989). *Neuronlike adaptive elements that can solve difficult learning control problems*. IEEE transactions on systems, man, and cybernetics, 5, 834-846.

Silver, D., Huang, A., Maddison, C. J., et al. (2016). *Mastering the game of Go with deep neural networks and tree search*. Nature, 529(7587), 484-489.

Foerster, J., Assael, I. A., de Freitas, N., & Whiteson, S. (2016). *Learning to communicate with deep multi-agent reinforcement learning*. In Advances in Neural Information Processing Systems (pp. 2137-2145).