Reinforcement Learning as Probabilistic Inference Application to the Vehicle Routing Problem

Amine Kobeissi [1] **Danil Garmaev** [1] **Jeremy Kim** [1] **Michael Almanza** [1]

Abstract

In this project, we explore the use of reinforcement learning (RL) reformulated as a probabilistic inference problem that solves the Vehicle Routing Problem. Building on the framework presented in ”Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review” by S. Levine, we frame policy optimization as probabilistic inference, using the maximum entropy principle to enhance exploration and robustness. We then re-implemented and adapted the Maximum Entropy RL method and Soft Actor-Critic (SAC) algorithm, as proposed in the paper “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor” by Haarnoja et al. This research bridges the gap between probabilistic graphical models (PGMs) and RL, and extends this current framework to discrete cases, demonstrating the potential of inference-based RL techniques for complex route optimization problems.

1. Introduction

The Vehicle Routing Problem (VRP) is a cornerstone of optimization research, with real-world applications spanning logistics, transportation, and supply chain management. The challenge lies in efficiently routing vehicles to serve customers while minimizing costs such as distance, fuel, or time, often under complex constraints. While traditional optimization techniques, like heuristics or exact solvers, have achieved success, they struggle to scale to larger, more dynamic problem instances.

Reinforcement Learning (RL) offers a promising alternative by enabling intelligent agents to learn adaptive routing strategies through interaction with the environment. However, standard RL approaches often face difficulties with exploration and stability, particularly in high-dimensional or sparse-reward settings.

To address these challenges, we explore inference-based RL techniques, specifically the Soft Actor-Critic (SAC) algorithm and Maximum Entropy (ME) methods. By framing the VRP as a probabilistic inference problem, we integrate the maximum entropy principle to enhance exploration and policy robustness. This allows the agent to discover diverse, high-quality routing solutions while avoiding premature convergence.

While this framework was originally designed to solve problems in continuous spaces, we have implemented an adapted approach to evaluate the performance of this framework on discrete problems such as VRP instances of increasing complexity, with metrics such as total travel cost (distance) and exploration quality. Experiments are conducted on synthetic VRP environments and benchmark datasets like the Solomon Dataset to validate this approach.

Through experiments on synthetic datasets, we compare the performance of the methods with that of traditional RL techniques like Deep Q-Networks (DQN), providing a foundation for applying inference-based RL to more complex optimization problems.

1.1. Reinforcement Learning

In Reinforcement Learning (RL) problems, an intelligent agent learns to take actions in an environment to maximize rewards over time. A basic RL problem is modeled as a Markov Decision Process (MDP):

- A set of environment and agent states / state space $(s \in S)$
- A set of actions $(a \in A)$ of the agent
- State transition probability: transition probability at time t from s_t to s_{t+1} under action a_t $[s_{t+1} \sim p(s_{t+1}|s_t, a_t)]$
- Reward: given after transition from s_t under action a_t $[r_t = r(s_t, a_t)]$

[1]Département d’informatique et de recherche opérationnelle, Université de Montréal, Montréal, Québec, Canada. Correspondence to: Amine Kobeissi <amine.kobeissi@umontreal.ca>, Danil Garmaev <danil.garmaev@umontreal.ca>, Jeremy Kim <jeremy.kim@umontreal.ca>, Michael Almanza <michael.almanza@umontreal.ca>.

2. Methodology

2.1. Reward-conditioned graphical model

In its unconditioned form, the graphical model often produces random actions, leading to random trajectories. To make this process more structured and goal-directed, we bake the reward function into the model. This is achieved by introducing **optimality variables** $\mathcal{O}$, which are binary indicators that specify whether a trajectory is optimal. These variables allow us to constrain and guide the agent's behavior. Within the goals of the project, we focus on constructing a policy and framing the reward function in a specific way:

$$p(\mathcal{O}_t = 1|s_t, a_t) = \exp(r(s_t, a_t)$$

Reward function r is modeled as the log-probability of the optimal variable $\mathcal{O}_t$ being True.

2.2. Graphical Model structure

The graphical model for this setup can be visualized as follows:

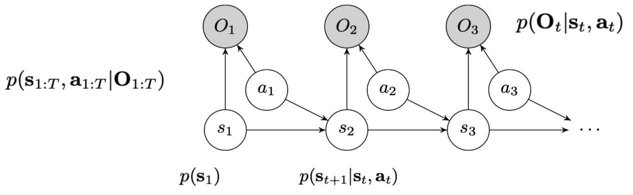
$p(\mathbf{O}_t|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}|\mathbf{O}_{1:T})$
O_1
O_2
O_3
a_1
a_2
a_3
s_1
s_2
s_3
$p(\mathbf{s}_1)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

Figure 1. Graphical model with optimality variables (observed).

This structure captures the dependencies between states (s_t), actions (a_t), and optimality variables ($\mathcal{O}_t$). It defines a probability distribution over trajectories. For this project, our primary focus is on inferring the policy given the rewards. This approach offers an alternative way to address the standard reinforcement learning (RL) problem.

2.3. Policy Construction

In this framework, the **policy** is represented as the conditional distribution $p(a_t|s_t, \mathcal{O}_{t:T})$.

This describes the action to take (a_t) given the current state (s_t) and conditioned on the trajectory being optimal in all future time steps ($\mathcal{O}_{t:T}$).

Since the graphical structure resembles a Hidden Markov Model (HMM), we can use backward-forward messaging techniques to infer the policy. Backward messages propagate information about optimality from the future to the past, while forward messages generate trajectories by sampling actions and propagating states over time.

2.3.1. Backward Messaging

The backward message encapsulates the probability that a trajectory from the current state-action pair (s_t, a_t) will remain optimal until the final time step T:

$$\beta_t(s_t, a_t) := p(\mathcal{O}_{t:T}|s_t, a_t)$$

Similarly, when considering only the state s_t, the backward message represents the probability of future optimality starting from that state:

$$\beta_t(s_t) := p(\mathcal{O}_{t:T}|s_t)$$

These definitions allow for a recursive formulation. The message at time t depends on the dynamics of the system, the reward structure, and the messages at future time steps.

$$\begin{aligned}\beta_t(s_t, a_t) &= p(\mathcal{O}_{t:T}|s_t, a_t)\\ &= \int_S p(\mathcal{O}_{t:T}, s_{t+1}|s_t, a_t)ds_{t+1}\\ &= \int_S p(\mathcal{O}_{t:T}|s_{t+1})p(s_{t+1}|s_t, a_t)p(\mathcal{O}_t|s_t, a_t)ds_{t+1}\\ &= p(\mathcal{O}_t|s_t, a_t)\mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[\beta_{t+1}(s_{t+1})\right]\end{aligned}$$

Here, $p(\mathcal{O}_t|s_t, a_t)$ reflects the reward (s_t, a_t), and the expectation integrates over all possible next states s_{t+1} based on the transition dynamics $p(s_{t+1}|s_t, a_t)$.

$$\begin{aligned}\beta_t(s_t) &= p(\mathcal{O}_{t:T}|s_t)\\ &= \int_A p(\mathcal{O}_{t:T}, a_t|s_t)da_t\\ &= \int_A p(\mathcal{O}_{t:T}|s_t, a_t)p(a_t|s_t)da_t\\ &= \mathbb{E}_{a_t\sim p(a_t|s_t)}\left[\beta_t(s_t, a_t)\right]\end{aligned}$$

This aggregates over all possible actions a_t, weighted by the prior policy $p(a_t|s_t)$, which can be assumed uniform without loss of generalization.

The backward messages link directly to standard RL concepts:

$$\begin{aligned}V_t(s_t) &= \log \beta_t(s_t)\\ Q_t(s_t, a_t) &= \log \beta_t(s_t, a_t)\end{aligned}$$

where $V_t(s_t)$ is a value function, and $Q_t(s_t, a_t)$ is the state-action value function. This leads to the following relationships:

$$\begin{aligned}V_t(s_t) &= \log \int \exp(Q_t(s_t, a_t))da_t\\ Q_t(s_t, a_t) &= r(s_t, a_t)\\ &\quad + \log\mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[\exp(V_{t+1}(s_{t+1}))\right]\end{aligned}$$

The connection is similar to the relationship in standard RL, with the log-integral-exponential term acting as a "soft-max" function.

In stochastic systems, this approach tends to favor rare actions with high rewards, which can lead to overly optimistic decisions. While this is mathematically valid, it may result in poor exploration, especially in noisy environments, as these rare actions can dominate the policy.

2.3.2. Optimal Policy

The policy is defined as $p(a_t|s_t, \mathcal{O}_{1:T})$, representing the probability of selecting an action a_t given the state s_t, under the condition of optimality across all time steps. We derive it as follows:

$$\begin{aligned} p(a_t|s_t, \mathcal{O}_{1:T}) &= p(a_t|s_t, \mathcal{O}_{t:T}) \\ &\quad \text{(independence due to model structure)} \\ &= \frac{p(a_t, s_t|\mathcal{O}_{t:T})}{p(s_t|\mathcal{O}_{t:T})} \\ &= \frac{p(\mathcal{O}_{t:T}|a_t, s_t)p(a_t, s_t)}{p(\mathcal{O}_{t:T}|s_t)p(s_t)} \\ &= \frac{\beta_t(s_t, a_t)}{\beta_t(s_t)} \cdot p(a_t|s_t) \end{aligned}$$

Assuming $p(a_t|s_t)$ is uniform, the policy simplifies to:

$$\pi(a_t|s_t) = \frac{\beta_t(s_t, a_t)}{\beta_t(s_t)}$$

Using reinforcement learning notation, we can express this as:

$$\pi(a_t|s_t) = \exp\left(Q_t(s_t, a_t) - V_t(s_t)\right) = \exp\left(A_t(s_t, a_t)\right)$$

where $A_t(s_t, a_t)$ is the advantage function.

Thus, we arrive at an intuitive interpretation: actions with higher values are more likely to be chosen, while lower-value actions still have a chance due to randomness, serving as a form of natural tie-breaking. This approach resembles Boltzmann exploration, where probabilities are weighted by the exponential of the action values.

2.3.3. Forward Messaging

The forward message, denoted as $\alpha_t(s_t)$, represents the probability of reaching state s_t given the observations up to time $t-1$, $\mathcal{O}_{1:t-1}$. Formally, it is defined as:

$$\begin{aligned} &\alpha_t(s_t) = p(s_t|\mathcal{O}_{1:t-1}) \\ &= \int p(s_t, s_{t-1}, a_{t-1}|\mathcal{O}_{1:t-1}) ds_{t-1} da_{t-1} \\ &= \int p(s_t|s_{t-1}, a_{t-1}, \mathcal{O}_{1:t-1}) p(a_{t-1}|s_{t-1}, \mathcal{O}_{1:t-1}) \\ &\cdot p(s_{t-1}|\mathcal{O}_{1:t-1}) ds_{t-1} da_{t-1} \end{aligned}$$

The main components:

- Transition Probability: $p(s_t|s_{t-1}, a_{t-1})$ describes the probability of transitioning to state s_t given the previous state s_{t-1} and action a_{t-1}.
- Policy Term: $p(a_{t-1}|s_{t-1}, \mathcal{O}_{1:t-1})$, which defines the probability of taking action a_{t-1} in state s_{t-1}, given past observations.

Using the structure of the graphical model, the policy term expands to:

$$p(a_{t-1}|s_{t-1}, \mathcal{O}_{1:t-1}) = \frac{p(\mathcal{O}_{t-1}|s_{t-1}, a_{t-1})\, p(a_{t-1}|s_{t-1})}{p(\mathcal{O}_{t-1}|s_{t-1})}$$

Linking Forward and Backward Messages: Once $\alpha_t(s_t)$ is computed, it can be combined with the backward message $\beta_t(s_t)$ to compute the posterior over states, $p(s_t|\mathcal{O}_{1:T})$, as follows:

$$\begin{aligned} p(s_t|\mathcal{O}_{1:T}) &= \frac{p(s_t, \mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})} \\ &= \frac{p(\mathcal{O}_{t:T}|s_t)\, p(s_t, \mathcal{O}_{1:t-1})}{p(\mathcal{O}_{1:T})} \\ &\propto \beta_t(s_t)\, \alpha_t(s_t) \end{aligned}$$

Thus, the posterior distribution over states is proportional to the product of the forward message $\alpha_t(s_t)$ and the backward message $\beta_t(s_t)$. This highlights the importance of both forward and backward passes in computing the overall likelihood of trajectories.

2.4. Improving Exploration with Variational Inference

Optimality variables often lead the agent to prioritize short-term rewards, limiting its ability to explore effectively. The resulting optimized trajectory distribution is:

$$p(\tau) = p(s_1|\mathcal{O}_{1:T} \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t, \mathcal{O}_{1:T}) p(a_t|s_t, \mathcal{O}_{1:T})$$

where the transition probabilities now depend on optimality. It means that the agent can control both the actions and the dynamics of the system, which is not desirable.

To address this, we use variational inference to redefine the trajectory distribution, reducing the direct dependence on optimality variables. The objective is to find a variational distribution $q(s_{1:T}, a_{1:T})$ that approximates $p(\mathcal{O}_{1:T}, s_{1:T}, a_{1:T})$, the structure in the following figure:

The trajectory policy becomes:

$$q(s_{1:T}, a_{1:T}) = p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) q(a_t|s_t)$$

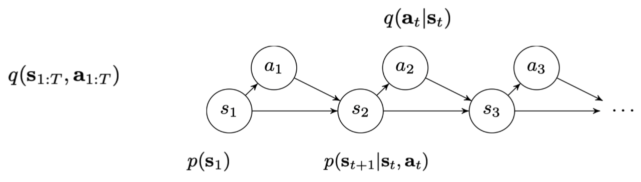
$q(\mathbf{a}_t|\mathbf{s}_t)$
$q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$
a_1
a_2
a_3
s_1
s_2
s_3
$\cdots$
$p(\mathbf{s}_1)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

Figure 2. Graphical model with optimality variables (observed).

The trick is that we now have a fixed transition probability. Using standard approach in variational inference, we have the following lower bound:

$$
\begin{aligned}
&\log p(\mathcal{O}_{1:T}) = \\
&= \log \int\int p(\mathcal{O}_{1:T}, s_{1:T}, a_{1:T}) \frac{q(s_{1:T}, a_{1:T})}{q(s_{1:T}, a_{1:T})} ds_{1:T} da_{1:T} \\
&= \log \mathbb{E}_{(s_{1:T}, a_{1:T}) \sim q(s_{1:T}, a_{1:T})} \left[\frac{p(\mathcal{O}_{1:T}, s_{1:T}, a_{1:T})}{q(s_{1:T}, a_{1:T})} \right] \\
&\text{by Jensen's inequality} \\
&\geq \mathbb{E}_{(s_{1:T}, a_{1:T}) \sim q(s_{1:T}, a_{1:T})} [\log p(\mathcal{O}_{1:T}, s_{1:T}, a_{1:T}) \\
&\quad - \log q(s_{1:T}, a_{1:T})] \\
&= \mathbb{E}_{(s_{1:T}, a_{1:T}) \sim q(s_{1:T}, a_{1:T})} \left[\sum_t r(s_t, a_t) - \log q(a_t|s_t) \right] \\
&= \sum_t \mathbb{E}_{(s_t, a_t) \sim q} \left[r(s_t, a_t) + H(q(a_t|s_t)) \right]
\end{aligned}
$$

Thee objective is now composed of two components: reward and entropy, but in terms of variational distribution. Together these terms balance exploration and exploitation, guiding the agent toward trajectories that are both high-reward and broadly exploratory.

2.5. Implementation

The code for the implementation can be found in the following GitHub Repository: https://github.com/AKobeissi/pgm-reinforcement-learning.

To address the Vehicle Routing Problem (VRP), we developed a synthetic grid-based environment, where we implemented the Soft Actor-Critic (SAC), and Maximum Entropy RL (MaxEntRL) algorithms proposed in the literature from scratch using PyTorch and NumPy. A custom environment was designed to simulate a 2D grid where an agent starts at a central depot and must visit randomly placed customers while minimizing travel distance.

2.5.1. Environment

In this implementation, the core of the VRP environment is encapsulated within the VRPEnvironment class. This class is responsible for initializing the problem parameters, managing the state transitions based on agent actions, computing the rewards that guide the learning process, and setting up synthetic data for experiments.

The class is initialized with the number of customers to include in the problem, the grid size, representing the area where the agent interacts, as well as the depot and customer locations. The grid size for our experiments is a 50 x 50 2D grid. The initialization has the option to include data for experiments such as the depot location, number of customers, customer locations, vehicle capacity, customer demand, as well as time based data points for tests, such as due time, ready time, and service time. All of the data is generated using numpy's random data generator. There is also the option to initialize the environment with the Solomon Benchmark datasets if provided. If this option is selected, the customer data is parsed from the Solomon data, scaled to the grid size of our environment, and the depot is defined as the first customer in the dataset.

The states of the environment represent the context of the current route at a given time to inform decisions. It consists of a tuple containing the current vehicle position (in grid coordinates), the customer locations, visit status of customers, the current time, remaining vehicle capacity (load), as well as customer data such as demands, ready times and due times. Providing the state with such data ensures that the agent has access to relevant information for decision making. Note that here the state includes normalized current position, customer coordinates, visit status, current time, remaining vehicle capacity, demands, ready times, and due times. Normalization ensures values are on a consistent scale for effective learning.

In terms of the actions, as we are in a discrete setting, the actions are represented as indices corresponding to which customer (node) to visit next. The action space size is equal to the number of possible destinations, and invalid actions are masked out using a binary mask in the state.

The reward is simply calculated as the negative travel distance (calculated with the Manhattan Distance), as is standard in the VRP (Mohammadreza, N., et al. 2018). In our implementation, we also experimented by adding a bonus of 1000 if the agent manages to visit all customers.

Both the action and rewards are used in the ”step” function of the environment, which implements the transition dynamics and calculates the rewards based on how the agent acts. This function then updates the states and calculates the travel distance, visit status, and arrival times.

After all states are computed, the ”reset” function is called at the end of each episode to reinitialize the state variables Finally, the VRPEnvironment class is wrapped by a Gym environment, helping us leverage learning libraries such as stable_baselines3 for implementations of other benchmark models.

2.5.2. Models

MaxEntRL

To address the VRP, we re-implemented two reinforcement learning algorithms from scratch for a discrete problem, as they were originally developed to solve problems on continuous spaces.

First, the Maximum Entropy Reinforcement Leanring (MaxEntRL) extends the traditional RL method by incorporaitng an entropy term in it's objective function. Theoretically, adding the entropy term encourages policies to maintain randomness, thus promoting exploration and preventing premature convergence to suboptimal policies. The objective function for the MaxEntRL is defined as

$$J(\pi) = \mathbf{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t))) \right]$$

where $\pi(a \mid s)$ is the policy, $r(s_t, a_t)$ is the reward function, γ is the discount factor, α the temperature, and $\mathcal{H}(\pi(\cdot \mid s_t))$ is the entropy of the policy. In this implementation, the MaxEntRL class encapsulates the MaxEnt RL model. The neural network architecture contains separate policy and value networks, each consisting of fully connected layers with ReLU activations. The policy network consists of fully connected layers with 256 hidden units and ReLU activations. The output layer uses a softmax activation to produce probabilities over discrete actions. The value network estimates the state-value function, it also uses fully connected layers with ReLU activations and outputs a single state-value estimate. The actions correspond to routing decisions, such as selecting the next point to visit.

Furthermore, in the MaxEntRL implementation, action selection incorporates invalid action masking to ensure that the agent only considers feasible routing decisions. After the policy network outputs a probability distribution over actions, a binary mask is applied to zero out probabilities for invalid actions, such as revisiting already-serviced customers or selecting non-existent nodes. This mask is dynamically generated based on the state, using the visit status of customers to identify valid actions. The masked probabilities are then re-normalized to form a valid probability distribution, ensuring the sum equals 1. Finally, an action is sampled from this re-normalized distribution, allowing the agent to make decisions that comply with VRP constraints while continuously encouraging exploration through random sampling.

Soft-Actor Critic

We also implemented a discrete version of the Soft Actor-Critic (SAC) algorithm for solving the Vehicle Routing Problem (VRP). While the original SAC was designed for continuous action spaces, we adapted it for discrete space by replacing the Gaussian policy with a softmax distribution over actions. Similar to basic maximum entropy reinforcement learning, the core objective of SAC is to maximize both the expected return and the entropy of the policy. For discrete action spaces, the entropy term is computed using the discrete entropy formula, $\mathcal{H}(\pi(\cdot|s_t)) = -\sum_a \pi(a|s_t) \log \pi(a|s_t)$. Unlike the simple MaxEntRL, the SAC algorithm optimizes three loss functions, corresponding to the critic networks, actor networks, and the temperature parameter for entropy.

First, for the critic netowkrs, it consists of two separate Q-networks, Q_1 and Q_2, which estimate the expected return for all state-action pairs. The critic loss, $\mathcal{L}_Q$, is a function that measures the discrepancy between the predicted Q-values and the target Q-values derived from the Bellman equation, incorporating both reward and entropy terms.

$$\mathcal{L}_Q = \mathbf{E}_{(s,a,r,s')\sim D} \Big[\big(Q(s,a) - \big(r + \gamma \min_{i=1,2} Q_{\text{target},i}(s', a') - \alpha \log \pi(a' \mid s')\big)\big)^2 \Big]$$

In our implementation, for each sampled transition, actions are sampled from the next state's policy, and the corresponding log probabilities are computed. The minimum Q-value from the target critics is used to form the target Q-value, incorporating the entropy term to encourage policy entropy.

Next, the actor loss, $\mathcal{L}_\pi$, encourages the policy to select actions that maximize the expected Q-values while having a high entropy. This is because it combines the entropy term with the negative of the Q-value. Minimizing this encourages the policy to select actins that yield higher expected rewards while maintaining entropy, which helps balance exploration and exploitation.

$$\mathcal{L}_\pi = \mathbf{E}_{s\sim D}[\alpha \log \pi(a|s) - Q(s,a)]$$

Finally, there is the temperature loss, $\mathcal{L}_\alpha$, which is a term adjusted to make sure that the policy has an entropy which aligns with the predefined target entropy.

$$\mathcal{L}_\alpha = \mathbf{E}_{s\sim D}[-\alpha(\log \pi(a|s) + \mathcal{H}_{\text{target}}]$$

In our implementation, the temperature loss is minimized using its dedicated Adam optimizer, allowing α to dynamically change without manual tuning.

The training procedure for the SAC agent is an iterative process involving five main steps. First, given the current state, the agent samples an action from the policy distribution and selects the action with the highest probability for execution in the VRP environment. Second, the selected action is executed, giving a transition tuple $(s, a, r, s', done)$, which is stored in the experience replay buffer. Third, a batch is sampled from the replay buffer to perform network updates. Fourth, the network updates are performed, the α is adjusted based on the current policy's entropy to maintain the desired exploration and exploitation trade-off, we minimize the critic loss to improve the Q-value estimates, and we minimize the actor loss to optimize the policy for higher rewards and maintained entropy. Fifth, the target network updates using a soft update rule for stability, $\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}}$, where τ is a small constant for the update rate. Finally, repeat steps 1 to 5 until convergence.

3. Results

We ran our implementations of Max Entropy RL and Soft Actor-Critic in VRP environments of increasing complexity, including a simple synthetic environment, a more complex synthetic environment, and the Solomon C101 benchmark. The quantities of interest in our experiments consist of the average distance and the average reward per episode, as well as the average of the last 10 final distances and rewards for each method, including our comparison methods, DQN, Nearest Neighbor, and OR-Tools. It should be noted that the Nearest Neighbor and OR-tools methods are not RL methods, so they don't have any reward values or final 10 episode averages, so they only have average distance.

3.1. Basic Environment

The initial experiments for these methods were run on a simple synthetic VRP environment. In this simple case, the agent seeks to minimize the travel distance while visiting as many customers as possible. The environment consists of a 50×50 grid with 10 customers initialized at random positions. The rewards are the negative distances from one customer to the next with a base reward for visiting a customer. In this way, our agent prioritizes customers in close proximity to its current location.

METHOD	AVG. REWARD	AVG. DISTANCE
MAXENT	-1181.04	232.7
SAC	-784.78	175.5
DQN	-110.12	8.59
NEAREST NEIGHBOR	N/A	127.48
OR-TOOLS	N/A	123.49

Table 1. Comparison of average distance and average reward on simple environment

METHOD	FINAL REWARD	FINAL DISTANCE
MAXENT	-1170.01	243.5
SAC	-552.22	140.9
DQN	-106.08	6.08

Table 2. Comparison of final distance and final reward on last 10 episodes of simple environment

We can see in both tables 1 and 2, MaxEnt and SAC are outperformed by every other model. The best performing model is DQN, which performs suspiciously well with an average distance of only 8. This could be due to a bug in the code or a lucky initialization, but it doesn't seem possible to visit 10 customers on a 50×50 grid in so few steps. MaxEnt and SAC do seem to obtain reasonable rewards, although they are both outperformed by Nearest Neighbor and OR-Tools. This indicates we have room left to improve our methods as it should outperform the Non-RL methods.

Also, the average of the last 10 episodes of SAC and DQN are an improvement over the average, which could indicate the RL algorithms are improving their output over time. With MaxEnt, the rewards are lower for the final 10 episode average, but it actually result in a larger distance. This could indicate a disconnect between our reward function and the quantity we actually want to minimize which is the distance. This may be due to our reward function being a sum of several objectives, notably the negative Manhattan distance and the customer reward here.

3.2. Complex Synthetic Environment

The more complicated synthetic VRP environment, which resembles the Solomon benchmark dataset, consists of a 50×50 grid, 100 randomly placed customers, each with a randomly generated demand between 1 and 30 units, and a vehicle capacity of 500 units. The reward function now incorporates a heavy penalty if the vehicle's visited customers demand exceeds its total capacity. Time related constraints were added such as the ready times, due times and service times. These extra constraints simulate real VRP environments and benchmarks like the Solomon benchmark data.

METHOD	AVG. REWARD	AVG. DISTANCE
MAXENT	-1499.76	323.4
SAC	-1620.64	328.0
DQN	-63503.46	2404.285
NEAREST NEIGHBOR	N/A	192.0
OR-TOOLS	N/A	151.0

Table 3. Comparison of average distance and average reward on complex environment

METHOD	FINAL REWARD	FINAL DISTANCE
MAXENT	-1334.2	285.4
SAC	-1530.2	318.4
DQN	-77033.2	2782.9

Table 4. Comparison of final distance and final reward on last 10 episodes of complex environment

We can see in table 3 that DQN performs quite poorly on our more complex environment. Its rewards are an order of magnitude worse than the other methods. This combined with the odd simple environment results could indicate an error with DQN. It also has worse rewards and distances for the average of the last 10 episodes. This could maybe be due to our incorporation of the capacity limit into the reward function, so the reward now depends on negative distance, customer reward (which is independent of demand), and the capacity constraint.

Otherwise, we still have Nearest Neighbor and OR-Tools outperforming MaxEnt and SAC, although all 4 methods seem to give reasonable results. This once again indicates our implemented methods are underachieving as we should be able to outperform these non-RL methods. The rewards and distances do seem better behaved here as we don't have an example of a lower reward yielding a larger distance.

3.3. Solomon Benchmark

In our application to real data, we tested our models on the Solomon benchmark. The Solomon C101 benchmark represents a real-world VRP scenario with tight time windows and capacity constraints. It consists of 100 customers with specific locations, demands, and strict service time windows. The benchmark is particularly challenging due to its clustered customer distribution and interdependent time-window constraints, making it a standard test for VRP algorithm effectiveness and efficiency.

Unfortunately, we encountered bugs which prevented us from obtaining valid results on the Solomon benchmark. It was quite a large bug which was only discovered when fixing a minor bug after the presentation date. When switching the distance from Euclidean to Manhattan distance as we were working in a discrete grid setting, we discovered a serious oversight with how our distances were being calculated and subsequently also had to change the reward function. It was difficult just to get the code to run on the synthetic environments after the fixes and we were unable to run it on the Solomon benchmark dataset as the code took around 1 minute per episode for the SAC method, so it was unrealistic to get a reasonable amount of episodes given our limited compute. To put it into perpective, many papers simulate millions of episodes, whereas even on our complex environment, we kept the episode count down to 1000 in order to have a our code terminate in a reasonable amount of time.

4. Conclusion

While our method of reformulating the maximization of a reward function as an inference problem in a graphical model did not outperform traditional RL algorithms, it still nonetheless has a potential to do so with a sufficient computing power to simulate more episodes. The method is already used to devise more effective and powerful forward RL algorithms, to developing probabilistic algorithms for modeling and reasoning about observed goal-driven behavior.

Recently, an intersection of maximum entropy RL and latent variable models have been explored, where the graphical model for control as inference is augmented with additional variables for modeling time-correlated stochasticity for exploration or higher-level control through learned latent action spaces.

One area that could be explored more is the connection between maximum entropy reinforcement learning and robust control. Training a policy to achieve high expected reward under the highest possible amount of injected noise (highest entropy) should result in it being robust to unexpected perturbations at test time. A detailed theoretical exploration of this phenomenon could be applied more broadly to a range of challenging problems involving domain shift, unexpected perturbations, and model errors.

Also, the exploration of the relationship between probabilistic inference and control can demystify the design of reward functions, which often are assumed to be merely an extrinsic and unchanging signal by traditional RL algorithms. Reinterpretation of rewards as log probability of some discrete event variable it can lead to more interpretable, more effective, and easier to specify reward functions and could lead to substantially more practical reinforcement learning methods in the future.

The probabilistic inference-based RL can be incorporated in a wide range of real-world challenges beyond VRP such as Bidding Strategy Optimization, and decision-making in Intensive Care Units (ICUs) that balance short-term patient survival with long-term recovery outcomes.

References

Levine, S. (2018). *Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review.*

Abdolmaleki, A., et al. (2018). *Maximum a posteriori policy optimisation.* In International Conference on Learning Representa- tions (ICLR).

Mohammadreza, N., et al. (2018) *Reinforcement Learning for Solving the Vehicle Routing Problem.*

Haarnoja et al. (2018) *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.*

Liu, M., Liu, J., Hu, Z., Ge, Y. and Nie, X. (2022). *Bid optimization using maximum entropy reinforcement learning.*

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). *Maximum entropy inverse reinforce- ment learning.* In International Conference on Artificial Intelligence

(AAAI).

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT Press.