Floyd-Warshall Reinforcement Learning: Combining Advantages from Model-Free and Model-Based RL

Vikas Dhiman[1], **Shurjo Banerjee**[1], **Jeffrey M. Siskind**[2] **and Jason J. Corso**[1]
The University of Michigan[1]
Purdue University[2]

Abstract

Reinforcement learning algorithms are oftentimes classified as either *model-free* or *model-based* based on whether a state-dynamics is learned. In the *model-free* case, algorithms such as Q-learning and policy gradients learn the optimal action value function that maximizes expected future reward for all state-action pairs perceived by an agent in an environment. The action value function depends upon both reward distribution and state dynamics but neither can be recovered from the learned action value function. That is why, *model-free* RL struggles with transferring learned behaviours to tasks where only the reward distribution can change, for example, goal-conditioned tasks. In contrast, *model-based* RL explicitly learns the state-dynamics function allowing for the transfer of an agent's learned behaviours to environments in which only the reward distribution changes. Oftentimes, this explicit modelling requires an additional planning step to predict state-dynamics making policy computation in such algorithms an expensive process. Inspired by both these paradigms and the Floyd-Warshall algorithm for path-planning on graphs, this work introduced Floyd-Warshall Reinforcement Learning (FWRL), a novel algorithm that combines advantages from both *model-based* and *model-free* approaches. The algorithm works by learning a goal-conditioned value function that transitions from model-based behavior to model-free behavior in well explored regions of an agent's state space. FWRL is shown to transfer knowledge about an environment when the reward location is dynamic compared to a model-based Q-learning baseline. FWRL is shown to meet the ground between model-free and model-based algorithms by being model-free in the more frequently visited regions while being model-based on less visited paths.

1 Introduction

Reinforcement learning (RL) is an exciting field of research as it allows for agents to learn complex, autonomous behaviors in a multitude of environments while requiring minimal supervision only in the form of reward signals. This is readily evidenced from RL's recent successes from goal-agnostic activites such as playing Atari games **?** from purely visual input and defeating world GO **?** and Starcraft champions (), to goal-driven ones such as recent applications in robotic navigation and manipulation. In the realm of goal-conditioned tasks, this work introduces Floyd-Warshall Reinforcement Learning (FWRL), a new algorithm that allows transferring learned behaviours to environments in which the underlying reward distribution is dynamic.

Algorithms that underly RL are often classified as being either *model-based* or *model-free*, the distinction being whether an environment state-transition function is learned explicitly or implicitly. In *model-based* RL, the dynamics that govern an environment's transitions is modelled as separate step for policy computation. At any point in an episode, agents use this model to predict future states and utilize this information to maximize possible reward. This formulation is known to be sample-efficient while normally not achieving high asymptomatic performance. Due to the requirement of planning steps to predict future states, policy computation can be an expensive process. In contrast, in *model-free* RL, algorithms such as policy gradients and Q-learning learn the optimal action value function by maximizing expected future reward for every state-action pair that the agent perceives. While highly sample-inefficient, agents trained under this paradigm have been shown to achieve high asymptomatic performance in a variety of different problem spheres.

Both paradigms of RL suffer different disadvantages in transferring learned behaviors to *reward-dynamic* settings i.e. environments in which the underlying reward distribution changes. While model-based RL allows for the separation of environment dynamics and reward, small errors in the modelling function lead to significant drops in performance. In *model-free* RL, on the other hand, the conflated representation of environment and reward makes any form of transfer difficult. These problems are exacerbated in multi-goal settings.

Inspired by the idea of combining advantages from both RL paradigms and the Floyd-Warshall algorithm in graph-based path planning, this work introduces Floyd-Warshall Reinforcement Learning, a new framework for combining model-based and model-free RL in goal-driven reward-dynamic settings. FWRL works by modeling a goal-conditioned action-value function where every state in the state space can be a valid goal. This allows FWRL to remember the paths even if they do not lead to the goal location during a particular episode. This motivation is similar to the Hindsight Experience Replay **?**, however, we use the parameteric representation for “hindsight experience” instead of the replay memory.

Ideas of combining both RL paradigms to improve agent performance are not new. In Temporal-Difference modelling **?** model a goal conditioned action-value function that is also conditioned on the number of time-steps. In contrast our proposed value function is independent of the time-steps. In Hindsight Experience Replay **?**, previous episode experience is used augment and bootstrap learning.

Experimentally, FWRL is shown to outperform both model-based, model-free and above combintations thereof in both a tabular and neural-network based setting. FWRL is found to outperform the next most significant baselines by as much x%.

In summary, this work introduces Floyd-Warshall Reinforcement Learning outperforming several strong baselines on a varierty of multi-goal reward-dynamic environments. The experimentation suite and all code is made available.

2 Related work

2.1 Goal-conditioned value functions

The idea of goal-conditioned value function is not new but has got attention because of revival of reinforcement learning based on deep neural networks. We build upon the goal-conditioned value functions of **?**. **?** proposed an architecture and a matrix factorization based algorithm for faster learning of UVFA (Universal value function approximators). UVFA focused on fast estimation of goal-conditioned value functions using sparse matrix factorization but not on bridging the gap between model-based and model-free algorithms.

? introduced the idea of Hindsight experience replay (HER) two learn about the model from previous episodes even when the goal location has changed or not been achieved. Our method can be seen as parametric approximation of “Hindsight memory”, that can help compress information from previous episodes instead of maintaining the entire history of replay memory.

? propose temporal difference models that estimate goal directed Q function for a specific kind of reward function, in particular the distance from the goal and in contrast with limited temporal horizon.

2.2 Combining model-based and model-free methods

2.3 Navigation with mapping

(1) CMP from Saurabh Gupta: is metric, might not working in continuous spaces. (2) Semi-parameteric Topological mapping: is not end to end. (3) Neural Map: Is actually not mapping

2.4 Model free DRL

does not generalize to multi-goal environments.

2.5 Model based DRL

Needs more exploration. Find the paper that shows that Model based DRL can actually compete with Model free DRL as long as it models uncertainty.

2.6 Multi-goal navigation based papers

Mirowski 2017, 2018: No one shot map learning, does not generalizes to new maps.

3 Background

We present a short review of the background material that our work depends upon.

3.1 Dijkstra

Dijkstra (**?**) is a shortest path finding algorithm from a given vertex in the graph. Consider a weighted graph $G = (\mathcal{S}, E)$, with $\mathcal{S}$ as the vertices and E as the edges. Dijkstra algorithms works by maintaining a data-structure $D : \mathcal{S} \to \mathbb{R}$, that represents the shortest path length from the source. The data structure D is initialized with zero at start location $D[s_0] \leftarrow 0$ and a high value else where $D[i] \leftarrow \infty \, \forall i \in \mathcal{S}$. The algorithm then sequentially updates D by

$$D[j] \leftarrow \min\{D[j], D[i] + r_{(i,j)}\} \, \forall (i,j) \in E, \qquad (1)$$

where $r_{(i,j)}$ is the edge-weight for directed edge $(i,j) \in E$. The shortest path $(s_0, s_1, \dots)$ starting from vertex $s_t \in \mathcal{S}$ can be read from D following nearest neighbor towards minimum D towards the goal $s_{t+1} = \arg\min_{i \in \text{Nbr}(s_t)} D[i]$ where $\text{Nbr}(s_t) = \{i | (i, s_t) \in E\}$ denotes the neighborhood of s_t. With a carefully chosen data-structure and traversal order, the Dijkstra Algorithm can be made to run in $O(|\mathcal{S}| \log |\mathcal{S}|)$.

3.2 Q-Learning

Q-learning (**?**) is a reinforcement learning (RL) algorithm that allows agent to explores the environment and simultaneously compute the maximum reward path.

An RL problem is formalized as an Markov Decision Process (MDP). A MDP is defined by a four tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the system dynamics and $R : \mathcal{S} \to \mathbb{R}$ is the reward yielded on a execution of an action. The objective of a typical RL problem is to maximize the expected cumulative reward over time, called the returns $R_t = \sum_{t'=t}^{T} r_{t'}$.

Q-learning works by maintaining an action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which is defined as the expected return $Q_\pi(s_t, a_t) = \mathbb{E}_\pi[R_t]$ from a given state-action pair. The Q-learning algorithm works by updating the Q-function using Bellman equation for every transition from s to s' on action a yielding reward r,

$$Q^*(s,a) = \mathbb{E}_\pi \left[r + \max_{a'} Q^*(s', a') \middle| s, a \right] \qquad (2)$$

3.3 Floyd-Warshall

Floyd-Warshall algorithm (**?**) is a shortest path finding algorithm from any vertex in a graph to any other vertex in the graph. Similar to Dijkstra, Floyd-Warshall algorithm finds the shortest path by keeping maintaining a shortest distance data-structure $D : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$. between any two pair of vertices $i, j \in \mathcal{S}$. The data-structure D is initialized with edges weights $D[i,j] \leftarrow r_{(i,j)} \, \forall (i,j) \in E$ and the uninitialized edges are assigned a high value $D[i,j] \leftarrow \infty \, \forall i,j \in \mathcal{S}$. The algorithm works by sequentially observing all the nodes in the graph and updating D as with the shortest path known so far:

$$D[i,j] \leftarrow \min\{D[i,j], D[i,k] + D[k,j]\} \quad \forall i,j,k \in \mathcal{S}\,. \qquad (3)$$

The update equation in the algorithm depends upon triangular inequality for shortest paths distances ($D[i,j] \leq D[i,k] + D[k,j]$) and hence works only in the absence of negative cycles in the graph. Fortunately, many practical problems can be formulated such that there are no negative cycles in the graph. Although, Floyd-Warshall algorithm runs in $O(|\mathcal{S}|^3)$ and is suitable for dense graphs, there exists extensions of the algorithm like Johnson's algorithm (**?**) that run in $O(|\mathcal{S}|^2 \log |\mathcal{S}| + |\mathcal{S}||E|)$ and work on the same principle.

Note, the similarity between the Eq. (**??**) and Eq. (**??**). Q-learning can be thought of as the generalization of Dijkstra with the introduction of the action space on the in the graph traversal problem. In MDP instead of choosing the next state (or vertex in graph traversal) to go to, we can only chose an action and the next state gets chosen by the system dynamics. However, RL is a much more challenging problem than path planning on graphs because there is no complete freedom on the order of traversal over state space and there exists a trade off between exploration and exploitation. With these parallels in mind, we extend the Floyd-Warshall algorithm to work on an MDP and call it Floyd-Warshall Reinforcement learning.

4 Problem definition

4.1 Environment Setup

Consider an agent interacting with an environment, ε. At every time step, t, the agent observes a state, $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the observation state space. The agent can traverse the state space by taking actions, $a \in \mathcal{A}$, where $\mathcal{A}$ is a fixed action space. A goal state, $g \in \mathcal{S}$, is specified to the agent where the goal state is a specific observation in the state space. R_{goal} is the reward recieved by the agent for finding the goal state and constitutes the largest reward in the environment. For every time step t, the agent takes an action a_t, observes a state $s_t \in \mathcal{S}$ and receives a reward $r_t \in [-R_{\text{goal}}, R_{\text{goal}}]$. Episodes are of a fixed number of time steps, T. For every episode, a randomized goal state is provided to the agent as input. If the goal state is observed by the agent during the course of an episode, the agent is randomly reinitialized within the environment while the goal state remains unchanged.

As is typical in RL domains, the agent's objective is to find the sequence of actions to take that maximizes the total reward from episode to episode. Since the environment itself is static, this is best achieved via the agent first discovering the goal location and then traversing the shortest path to it from every subsequent *spawn* state during the course of an episode. The agent is best suited by algorithms that emphasize the transfer *environment structure* from episode to episode.

4.2 Why is this problem important?

Many real world problems can be formulated in this context.

Consider a traveling postman problem who has moved into a new city. The postman has to explore the city and find the buildings that match the given address. The next time the postman gets the same address, they can use their experience to find out the building. Even when a new address is provided (in the next episode), the postman can use experience to find the new episode more quickly.

In robotics, tasks like picking and placing the object at a desired location can be formulated as goal-directed navigation.

4.3 Why is the problem hard?

Model-free Reinforcement learning methods assume that the rewards are being sampled from the a static reward function. In a problem where the goal location changes, hence the reward function also changes, it becomes hard to transfer the learned value-function or action-value function to the changed location. One alternative is to concatenate the goal location the state, making the new state space $[s_t, g]^\top \in \mathcal{S}^2$ larger. This method is wasteful in computation and more importantly in sample complexity.

5 Method

We present a model-free reinforcement learning method that easily transfers when goal location is dynamic. We accomplish this by maintaining a path based expected reward function from any state to any goal state. We call this algorithm Floyd-Warshall Reinforcement Learning, because of its similarity to Floyd-Warshall algorithm : a shortest-path planning algorithm on graphs. We define Floyd-Warshall value function as

$$F_\pi(i, l, j) = \mathbb{E}_\pi \left[\sum_{t=0}^{t=k} r_t \middle| s_0 = i, a_0 = l, s_k = j \right]. \quad (4)$$

When the policy is optimal, the Floyd-Warshall function should satisfy the constraint

$$F^*(s_i, a_i, s_j) = \max_{s_k} \left[F^*(s_i, a_i, s_k) + \max_{a_k} F^*(s_k, a_k, s_j) \right]. \quad (5)$$

We summarize the algorithm in Alg **??**.

Algorithm 1: Floyd-Warshall Reinforcement Learning (Tabular setting)

```
Let $r_g \leftarrow 10$;
/* By default all states are
   unreachable                                  */
Initialize $F(s_i, a_i, s_j; \theta_F) \leftarrow -\infty$ ;
Initialize $Q(s_i, a_i) \leftarrow 1$ ;
Initialize $s_g = \phi$ ;
Set $t \leftarrow 0$;
Observe $z_t$ ;
$s_t = \Phi_o(z_t; \theta_E)$ ;
for $t \leftarrow 1$ to $T$ do
    /* See Function ??                          */
    Take action $a_{t-1} \sim$ Egreedy($\pi^*(s_{t-1}, s_g, Q, F)$) ;
    Observe $z_t, r_t$ ;
    $s_t = \Phi_o(z_t; \theta_E)$ ;
    if $r_t >= r_g$ then
        /* Reached the goal                     */
        $s_g \leftarrow s_t$ ;
        /* Respawning does not need
           update of value functions            */
        continue;
    $Q(s_{t-1}, a_{t-1}) \leftarrow r_t + \max_a Q(s_t, a)$ ;
    $F(s_{t-1}, a_{t-1}, s_t) \leftarrow r_t$ ;
    for $s_k \in \mathcal{S}, a_k \in \mathcal{A}, s_l \in \mathcal{S}$ do
        $F(s_k, a_k, s_l) \leftarrow$
          $\max\{F(s_k, a_k, s_l), F(s_k, a_k, s_t) +$
          $\max_{a_p \in \mathcal{A}} F(s_t, a_p, s_l)\}$ ;
Result: $\pi^*(s_k, s_g, Q, F)$
```

6 Experiments

Experiments are conducted in a grid-world like setup as displayed in figure **??**. The agent can occupy any one of the white blank squares. The agent's observations is the numbered location of each square i.e. each squares x,y coordinate with the origin being the top left corner of the environment i.e. $s_t = (x, y)$. Agents can act by moving in the four different cardinal directions, $\{up, down, left, right\}$. Grid-world is chosen since due to it's simplicity it easily highlights differences between baseline models and FWRL. Several different grid-world setups are investigated to test the abilities of FWRL.

6.1 Four room grid world

Four room grid world is a grid world with four rooms connected to each other as shown in Figure **??**. This example is chosen due to it's intentional difficulty for random exploration based agents. Since the exit points, are narrow, random agents tend to get stuck in individual rooms.

6.2 Four room windy world

In four room windy world, the previous setup is augmented with *wind*. In cells that wind, shown by arrows, the agent gets pushed around by the wind with 0.25 probability in the direction of the arrow irrespective of the action taken. Concieved by , the setup increases the dependence of the dynamics model upon environment specifics.

6.3 Random Grid Worlds

While the prevoius experiments were chosen for the ability to study a specific property of these algorithms, random grid-world maps are created and tested upon.

6.4 Metrics

The metrics used to quantify and compare agent performance across both baseline methods and FWRL are described here.

1. **Reward**
   As in typical in reinforcement learning, the reward earned by the agent is treated as a metric of success. Since the environments used are finite MDPs, Q-Learning is known to learn the optimal policy given enough exploration time. Of interest is thus the amount of time taken for reward to climb.

2. **Latency-1:>1**
   First defined in **?**, Latency-1:>1is the ratio of the amount of time taken to hit the goal for the first time to the average amount of time taken to hit goals subsequently. It is the ratio of the exploration time over the exploitation time.

$$\text{Latency-1:>1} = \frac{(N-1)\tau_1}{\tau_N - \tau_1} \qquad \text{if } N >= 2$$

3. **Distance-Inefficiency**
   Described in **?**, the distance-inefficiency is the ratio of the distances travelled by the agent during an episode to the sum of the shortest path to the goal at every point of initialization.

$$\text{Dist-ineff.} = \frac{\sum_{i=1}^{N-1} \sum_{t=\tau_i+1}^{\tau_{i+1}-1} \|x_{t+1} - x_t\|}{\sum_{i=1}^{N-1} p(x_{\tau_i+1}, x_g)} \quad , \quad \text{if } N >= 1$$

where $p(x_{\tau_i+1}, x_g)$ denotes the approximate shortest path distance between spawn location x_{τ_i+1} and goal location x_g.

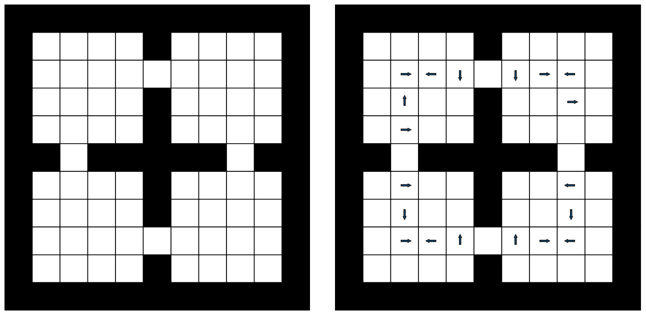

Figure 1: Left: Four room grid world. Right: Four room windy grid world with wind direction shown by arrows. The windy pushes the agent in the direction of wind with 0.25 probability irrespective of the action taken.

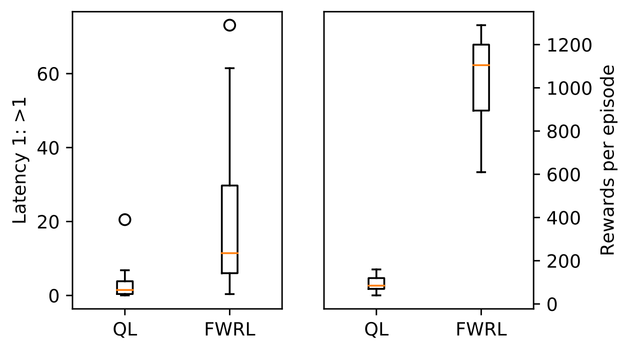

Figure 2: Results on grid world. FWRL beats Q-Learning consistently. Higher is better in both the metrics (higher is better).

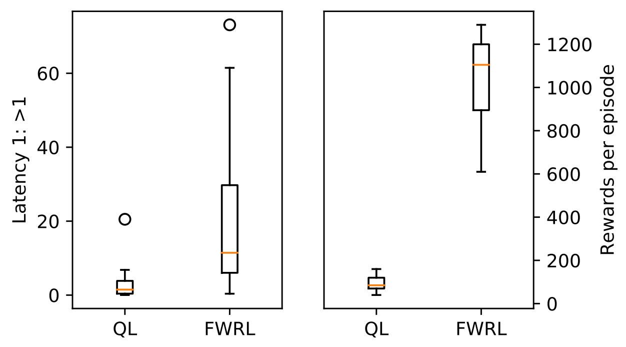

Figure 3: Results on windy world. FWRL beats Q-Learning consistently. Higher is better in both the metrics (higher is better).

7 Results

We evaluate Q-learning and Floyd-Warshall Reinforcement Learning (FWRL) on two metrics in two different environments. The two metrics we use are Latency Ratio and average reward per episode. The Latency ratio metric was introduced in **?**, which is defined as the ratio of time taken to reach the goal for the first to time to the average time taken to hit the goal thereafter. The Latency ratio thus measures the ratio of exploration time for first time finding the goal to the average exploitation time to reach the goal. Hence, higher latency ratio is better. Fig **??** and Fig **??** show the results.

7.1 Conclusion

Floyd-Warshall Reinforcement Learning (FWRL) allows us to learn a goal conditioned action-value function which is invariant to the change in goal location as compared to the action-value function used in typical Q-learning. This allows FWRL to transfer learned behaviors about the environment when the goal location changes. Many tasks like navigation, robotic pick and place are examples of goal-conditioned tasks that can benefit from this framework.

- Grid world: Set up a random goal static maze scenario, compare with normal Q-learning.
- Lava world: Set up a Lava world like **?** and test on it.
- : Set up a random goal static maze scenario, compare with normal Q-learning.