RL-IoT：Towards IoT Interoperability via Reinforcement Learning

Jiajun Han, jh4316

Contents

1. Problem Description

- **Motivation:**

In information systems, the presence of IoT devices is exponentially growing and most of them are custom devices: they rely on proprietary protocols, often closed or poorly documented. Here we want to interact with such devices, by learning their protocols in an autonomous manner.

- **Goal:**

In the referenced paper, the author proposes RL-IoT, a system that explores how to automatically interact with possibly unknown IoT devices. Leveraging reinforcement learning to recover the semantics of protocol messages and to take control of the device to reach a given goal, while minimizing the number of interactions. Assume we know only a database of possible IoT protocol messages, whose semantics are however unknown. RL-IoT exchanges messages with the target IoT device, learning those commands that are useful to reach the given goal. A Yeelight smart bulb is in the case study.

2. Reinforcement Learning Algorithms

The paper includes 4 algorithms belonging to two categories:

• **Temporal-Difference (TD) learning**: The agent updates V (s) after every time step as: Common Phishing methods:

$$V(s_t) \leftarrow V(s_t) + a[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \qquad (1)$$

The parameter α is the learning rate, how much V (s_t) should change when updated. γ is a discount factor that weights the importance of the destination state $V(s_{t+1})$. SARSA and Q-learning are popular TD algorithms .The former is an on-policy algorithm, whereas the latter is an off-policy method.

2. Reinforcement Learning Algorithms

- **TD(λ) learning:**

The agent takes n time steps before updating $V(s_t)$. As such, TD(λ) algorithms must memorize visited states to update them later. The parameter λ controls how the n future states influence $V(s_t)$ (i.e., like a decay parameter). And this paper use the he most common TD(λ) algorithms are direct extensions of traditional TD learning methods: **SARSA (λ) and Q(λ)**.

As for the **ϵ-greedy algorithm**, we should choose an improved of the ϵ-greedy method which is a decayed-ϵ-greedy method. We are supposed to set ϵ in a way that allows a high exploration during the initial search.

3. IoT Reinforcement Learning framework

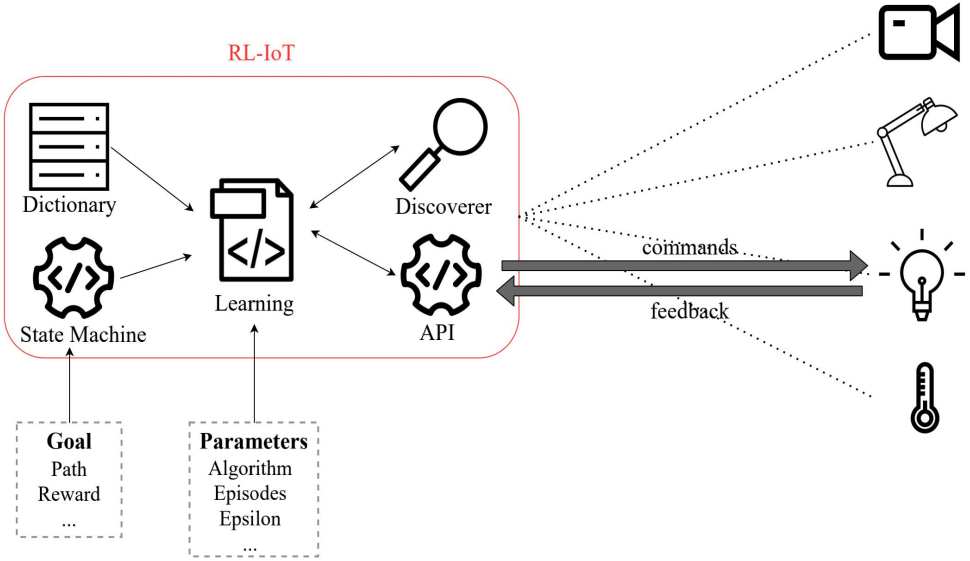


Figure. : RL-IoT framework overview

RL-IoT receives a **Goal** as input that the RL Module should learn how to achieve. The Goal represents a sequence of settings the device should follow. We assume to receive a feedback to decide **Reward** from the feedback channel directly offered by the IoT protocol. (Result message is generated upon receiving COMMAND messages.)

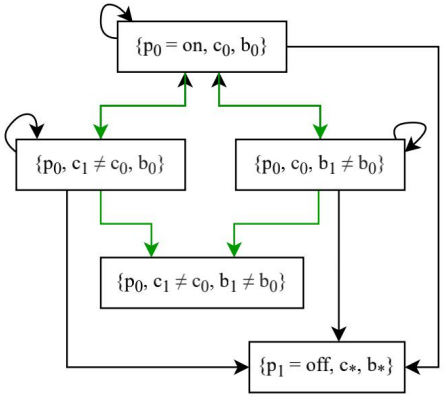
It leverages an internal **Message Dictionary** containing a list of IoT protocol messages that can be used to interact with devices. This dictionary can contain a mix of messages from different IoT protocols, vendors, versions, etc.

The **Learning module** builds and updates the internal State Machine. It explores which of the several messages in the Dictionary can be used to change the state of the IoT device towards the given Goal.

The Learning module interacts with two other modules. The **Discoverer module** is responsible for scanning the local network in the search for IoT devices. And the **Socket API module** abstracts all the mechanisms to communicate with the target IoT device.

4. Environment Definition

We define the **state-machine** of a protocol as a graph containing nodes for states and edges for commands that let the device move from one state to another. A collection of ordered states linked by commands is a **path**. Commands stored in the Message Dictionary could change the IoT device settings, i.e., the current state. Note that we assume that the sets of states S and actions A are finite sets, and thus the Q value function $Q(s, a)$ can be represented as a matrix.



5. The Use Case Study: The Yeelight bulb

This paper use a Yeelight smart bulb as a case study to demonstrate the feasibility of the approach. The protocol offers 37 commands, and only about half of them work with the selected smart bulb, with multiple commands that could generate the same action. For instance, one could set a color via a *set_rgb*, *set_scene*, or *adjust_prop* message.

The smart bulb connects to the network using Wi-Fi. It periodically broadcasts its presence using **Advertisement Message**. It is thus easy for the Discoverer module to find them in the LAN. Once the system identifies the device IP address, it starts interacting with it by sending **Command Message** from the Dictionary and receiving **Request Message** from the bulb.

5. The Use Case Study: The Yeelight bulb

Yeelight offers control protocols running on top of both HTTP and raw TCP sockets.
The latter relies on simplistic JSON messages that carry commands. This figure presents one of the simplest JSON messages to set the color of a smart bulb. The device responds to well formatted commands with a result message.

Command:

```
{"id": 1,
 "method": "set_rgb",
 "params": [255, "sudden", 0]}\r\n
```

Answer:

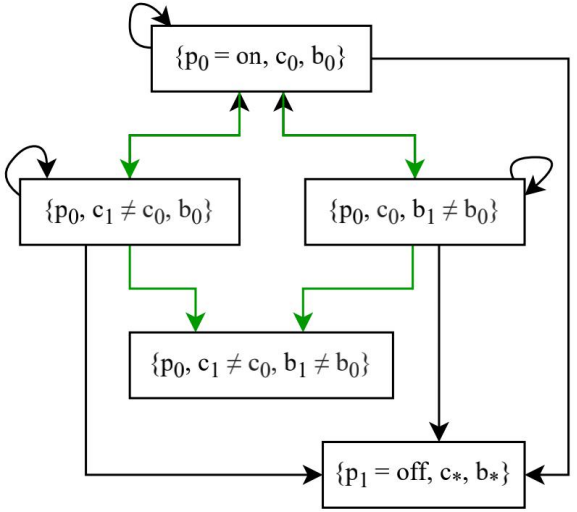
```
{"id": 1, "result": ["ok"]}\r\n
```

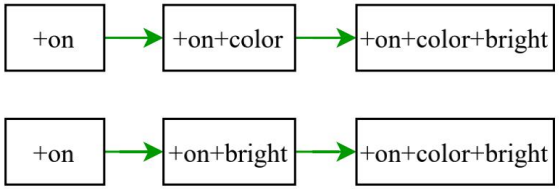
The commands can have some parameters to set, which belongs to finite sets, for some commands the number of admissible values can be huge ((i.e., the color of the smart bulb has 16777216 possible values.). And the combinations of commands result into a large nunmber of distinct combinations. To reduce the action space, this paper considers the action as only one command, with its parameters randomly chosen in valid ranges.

5. The Use Case Study: Definition of goals

For testing RL-IoT, the paper build and study two scenarios with different state-machines of increasing complexity. In the first scenario, given a switched-on bulb, our **Goal 1** is to learn how to change the color and the brightness of the bulb, in whatever order.

We perform a transition from one state to another when one or more of these attributes are modified. This allows us to represent cases in which an attribute is continuous and/or has a high number of admissible values.

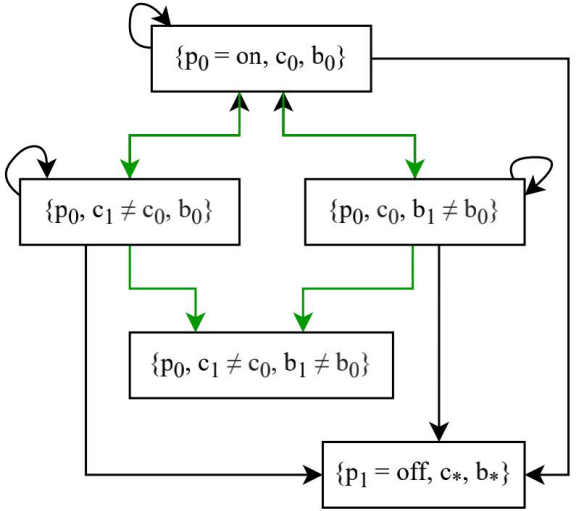


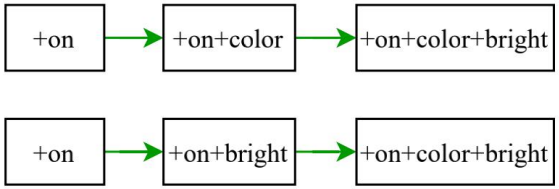


5. The Use Case Study: Definition of goals

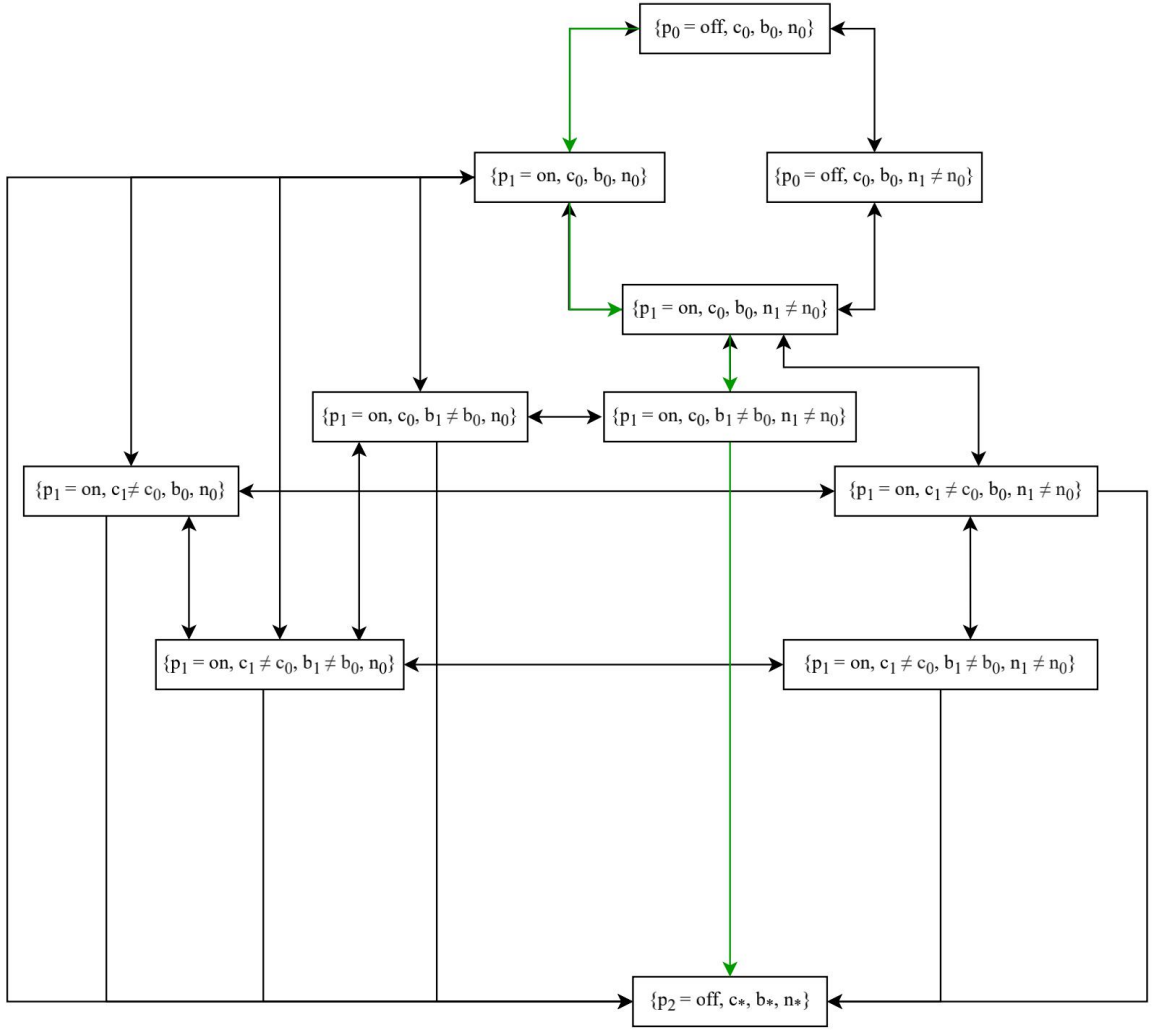
Here, the paper assigns the rewards for **Goal 1** as follows: (i) each new issued command has a small additional negative reward (-1), since we want to reach the goal in as few steps as possible; (ii) we give higher negative reward (-10) when the command produces an error or the state does not change; (iii) we assign no reward when we reach the final state without completing the path *{p1 = off, c∗, b∗}*; (iv) we give large positive reward (+205) when we reach the desired final state *{p0, c1 = c0, b1 = b0}*.

Hence, with these assigned rewards, the optimal paths will reach a total reward of 203 (205 minus 2 steps).





Goal 2 is more complex since we want to learn how to move through a specific sequence of states. This paper assigns a large positive reward (+222) at the final state if we pass through the desired states in the right order. If we arrive to the same final state, but in a different sequence of the same intermediate states, we assign a positive, but smaller reward (+200). Negative rewards are similar to Goal 1. Here the optimal path is unique, with an optimal length of 4 time steps, generating the maximum total reward of 218 (i.e., 222 minus 4 steps).



6. Evaluation and Results

1. Performan Metrics
2. Learning Capacity
3. Parameter Tuning and Algorithms Comparison

5.1 Performance Metrics and Notation

We compute the total reward R(E) obtained during episode E:

$$R(E) = \sum_{t=1}^{T(E)} r_t(E) \text{ for } E \in \{1, ..., N_E\}$$

And compute the cumulative reward C(n_a) from the beginning of the learning process over the number of actions performed n_a:

$$C(n_a) = \sum_{E=1}^{E_{n_a}} \sum_{t=1}^{T_{n_a}(E)} r_t(E) \text{ for } n_a \in \{1, ..., N_a\}$$

Formal notation for evaluation metrics and parameters of the RL algorithms.

E	episode
N_E	total number of episodes
$R(E)$	total reward obtained in episode E
$T(E)$	total number of time steps t in episode E
N_a	total number of actions performed
$C(n_a)$	cumulative reward obtained after n_a actions
$Q(s, a)$	action value function or Q value function
ϵ	exploration-exploitation trade-off
α	learning rate
γ	discount factor
λ	trace decay

5.2 Learning Capacity

For **Goal 1**, Q-learning initially cannot reach the desired state. Missing the large positive rewards, it accumulates a negative reward on average. After few episodes, `R(E)` grows to the maximum value that could be observed (203 here).

The results are qualitatively similar for Goal 2, but with slower learning, given the higher complexity of the goal. However, also in this case the learning phase is still able to discover paths with positive reward after around 20 episodes.

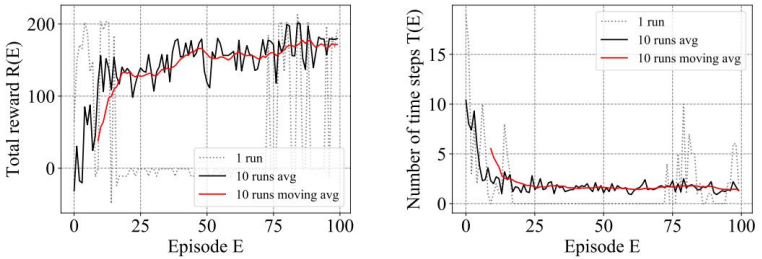


Fig. 6: Q-learning performance while learning Goal 1. $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.55$.

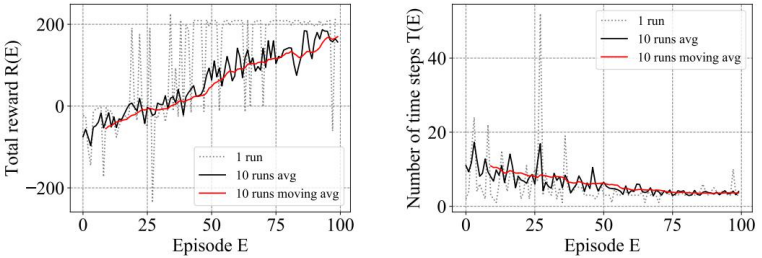


Fig. 7: Q-learning performance while learning Goal 2. $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.55$.

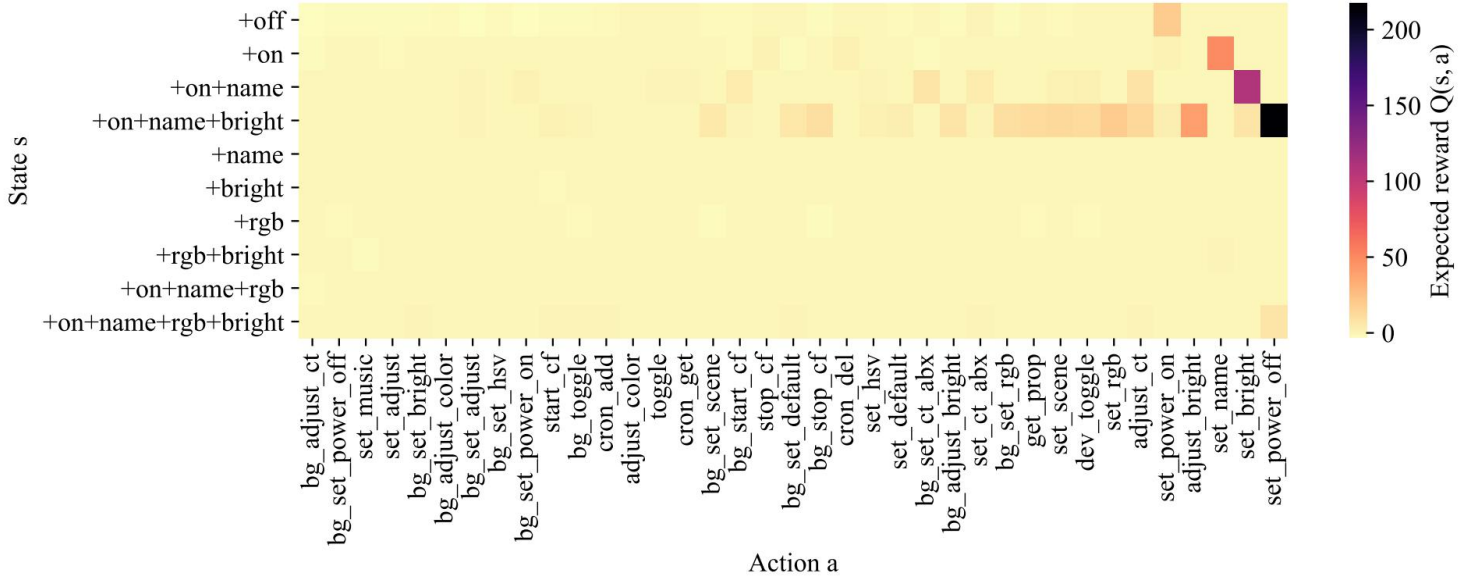
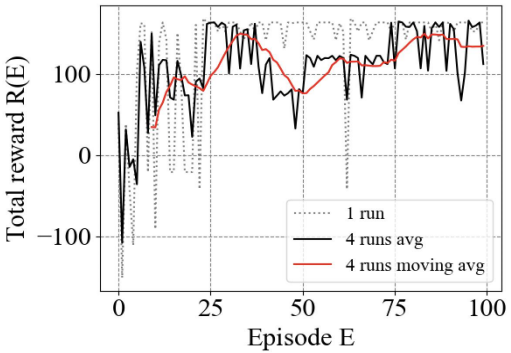
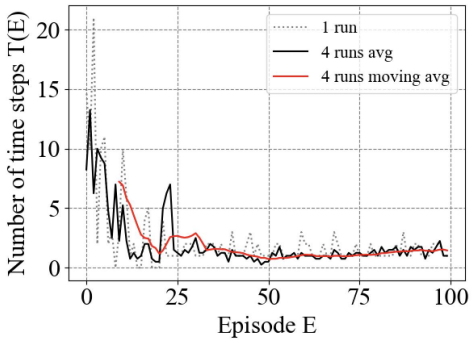


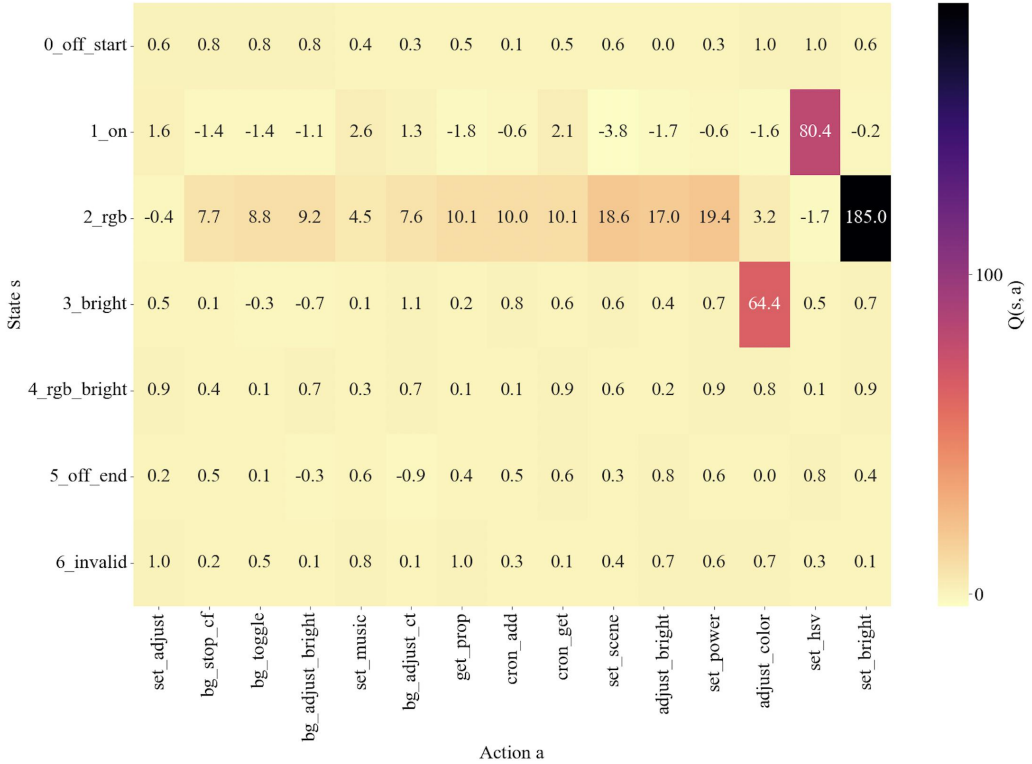
Fig. 8: Example of the final action-value Q matrix for Goal 2. Darker colors show commands (columns) that result in higher expected rewards for the states (rows). $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.55$.

The Q matrix above for Goal 2 is obtained after 100 episodes. Rows represent states, with the first 4 rows being the desired optimal path in the second scenario. Columns represent all available commands (actions). The darker is the color, the higher is the chance to select that command in that state.
Interestingly, RL-IoT has also identified alternative valid commands to move to the desired state, as shown by moderately darker shades for some commands.

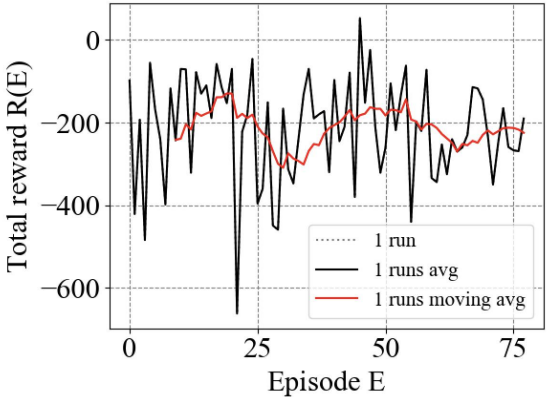
My experiment result: Goal 1 (Q-learning)

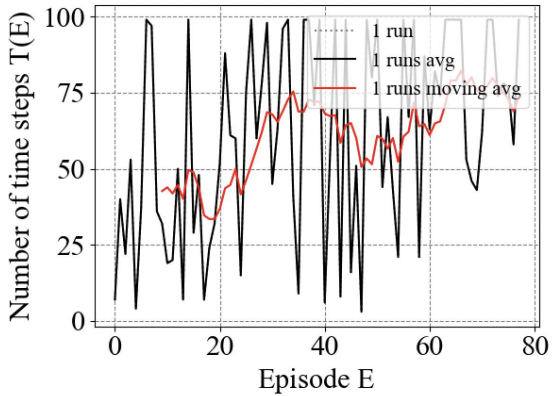




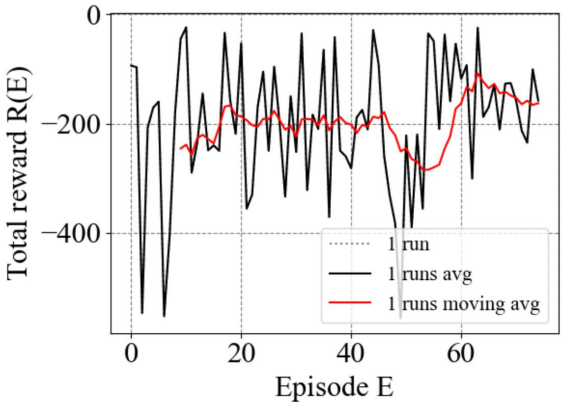


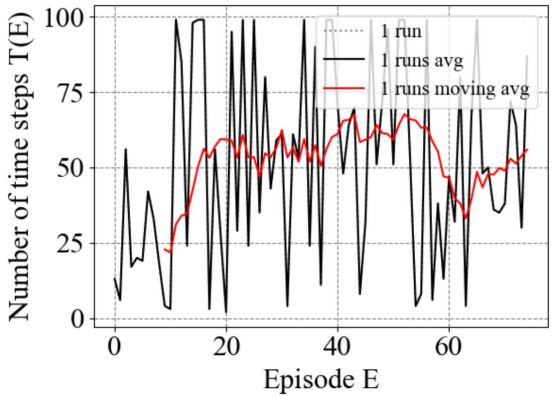
My experiment result: Goal 2 (Q-learning)





Qlearning(ϵ= 0.2, α = 0.2 and γ = 0.55)





Q(λ)(ϵ= 0.2, α = 0.1 and γ = 0.55)

My experiment result: Another Example

In addition to the two goal mentioned in the paper, I tested another example which I named **Goal 3,** as what the left figure shows.

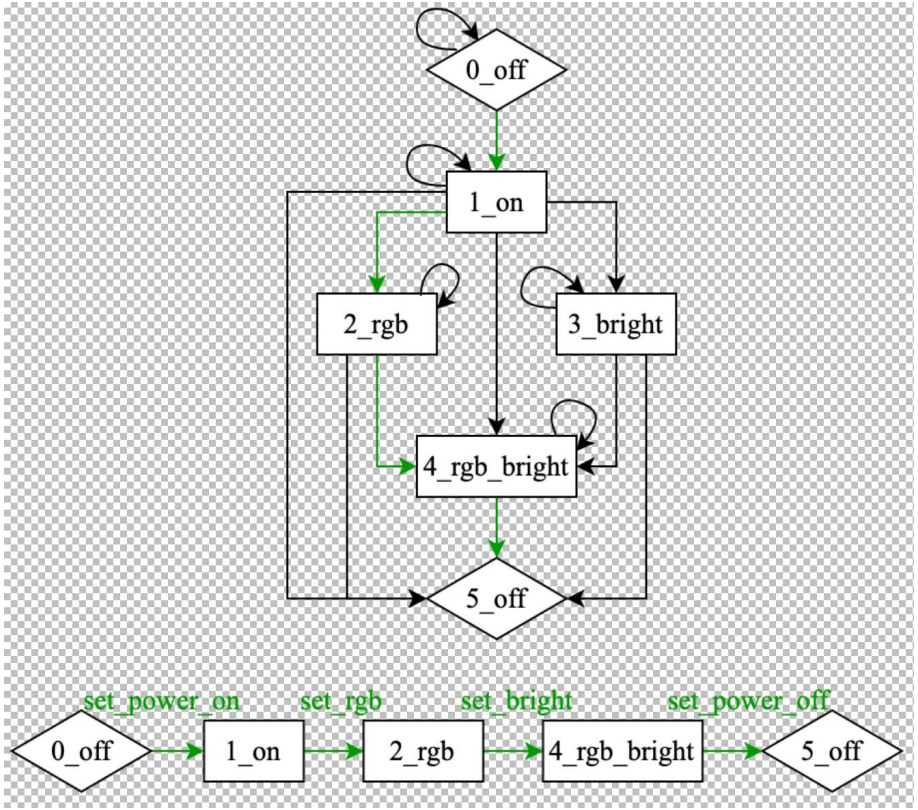
I assigned a large positive reward (+215) at the final state if it pass through the desired states in the right order (0-1-2-4-5).

If it arrives to the same final state, but in a different sequence of the same intermediate states (that is only 0-1-3-4-5), I assign a positive, but smaller reward (+200).

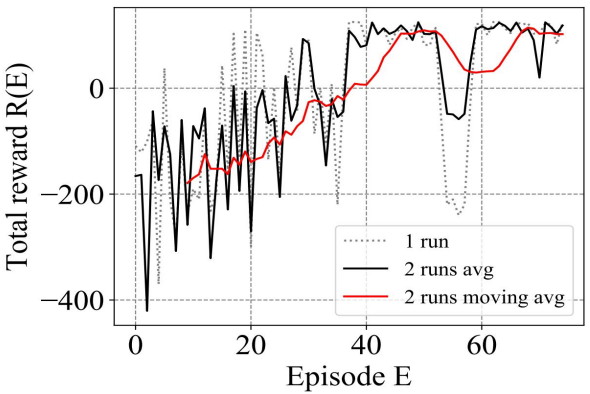
I assign no reward when it reaches the final state without setting both color and brightness.

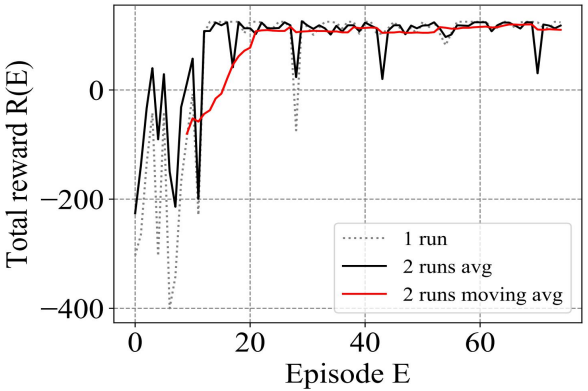
Negative rewards are similar to Goal 1.

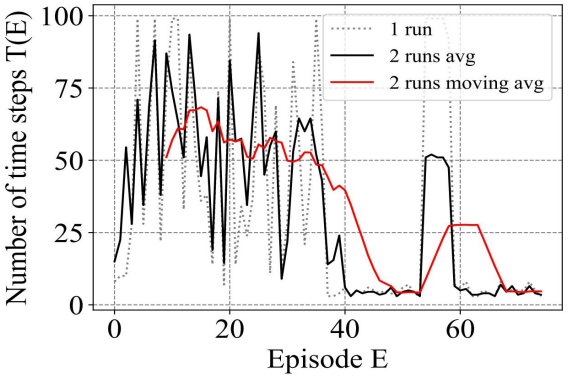
Here the optimal path is unique, with an optimal length of 4 time steps, generating the maximum total reward of 211 (i.e., 215 minus 4 steps).



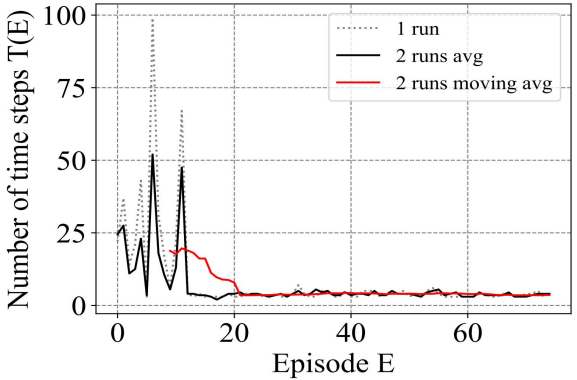
My experiment result: Another Example





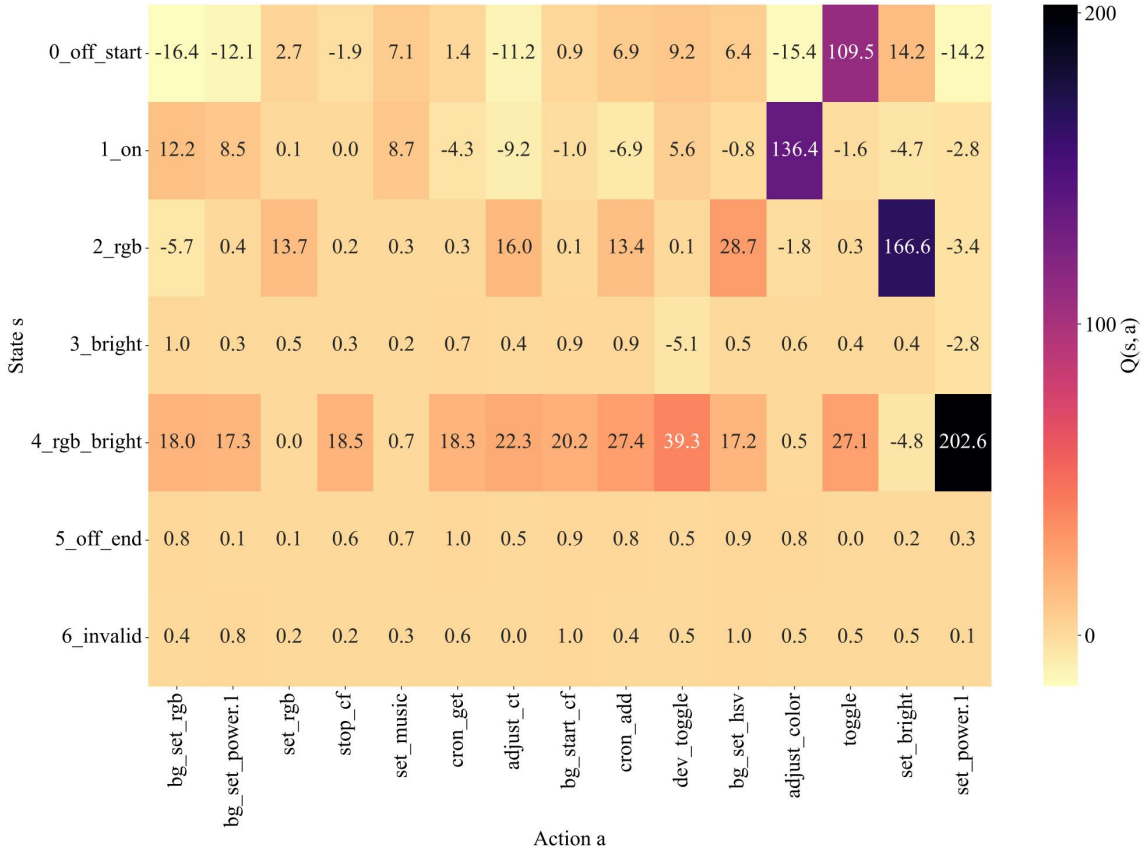


Qlearning(ϵ= 0.2, α = 0.2 and γ = 0.55)



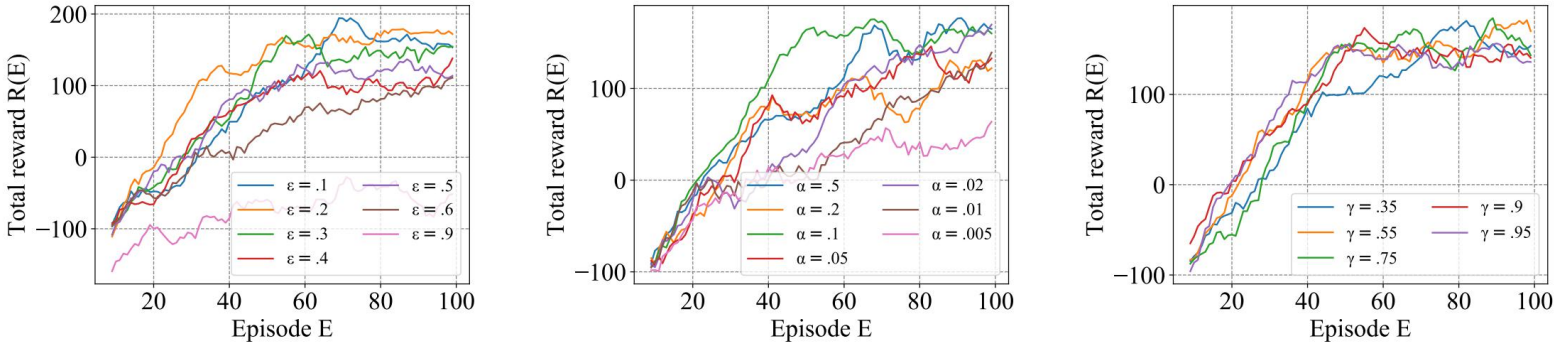
Q(λ)(ϵ= 0.2, α = 0.1 and γ = 0.95)

My experiment result: Another Example



Q(λ): Q state-action value matrix

5.3 Parameter Tuning



(a) Tuning ϵ (b) Tuning α (c) Tuning γ

Fig. 9: Q-learning performance for Goal 2: (a) tuning of ϵ with $\alpha = 0.05$, $\gamma = 0.95$; (b) tuning of α with $\epsilon = 0.2$, $\gamma = 0.95$; (c) tuning of γ with $\epsilon = 0.2$, $\alpha = 0.1$.

The paper performs greedy experiments to find the optimal parameter, in which varys only one parameter at a time. More specifically, starting from values (ϵ = 0.6, α = 0.05 and γ = 0.95), we first tune ϵ with α and γ fixed. Then, we fix the best ϵ, and optimize α. Finally, we optimize γ given the best ϵ and α.

We see that ϵ and α are the parameters affecting the most the RL algorithm. The figure above shows that large ϵ can even prevent the algorithm to reach the maximum reward. In a nutshell, better not explore too much. Similar comment applies to low values of α, that is better to learn fast.

5.3 Algorithms Comparison

The paper compares the best configuration for the four algorithms in **Goal 2**. It shows that Q(λ) obtains the highest rewards during the initial 50 episodes, which learns faster than others. Yet, from episode 50 onward Q-learning wins, reaching the maximum values at around 200 episodes.

We see in the bottom plot that Q-learning learns shorter paths (on average) after 200 episodes. That is, it reaches the goal with less steps, obtaining higher rewards than the other algorithms.

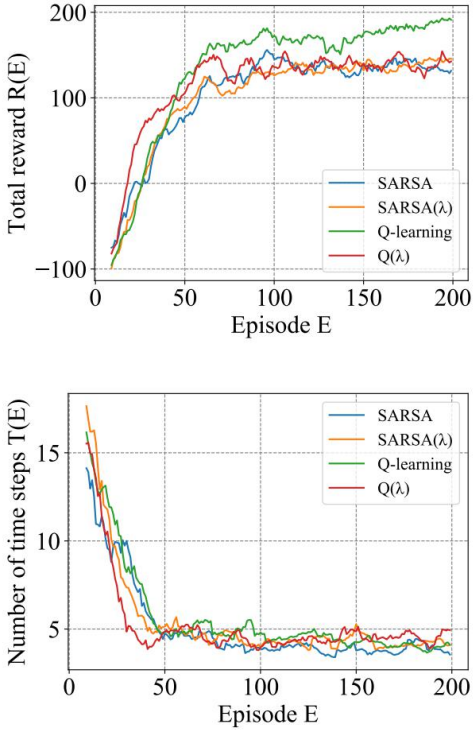


Fig. 10: Algorithm comparison for Goal 2. SARSA: $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.75$; Q-learning: $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.55$; SARSA(λ): $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.75$, $\lambda = 0.5$; Q(λ): $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.55$, $\lambda = 0.9$.

7. Critical Thinking

- The referenced paper shows RL-IoT could discover semantics of unknown IoT messages in the case of Yeelight smart bulb. However, in the general case this could not be easy, e.g., when the protocol uses binary format.

- To reduce the action space, this paper considers the action as only one command, with its parameters that are randomly chosen in valid ranges. Yet there could be combinations of commands and their parameters.

- This paper assumes that the sets of states *S* and actions *A* are finite sets. If not, there exist methods which combine standard RL algorithms with function approximation techniques, such as neural networks.

- Even with the reduced commands and states, it is exhaustive to finish one execution. I almost spend 7 hours in executing 100 episodes for Goal 2, due to rate-limits (60 commmand message per sec) caused by the Yeelight protocol.

8. References

[1] Referenced paper (RL-IoT: Reinforcement Learning to Interact with IoT Devices): https://arxiv.org/abs/2105.00884
[2] Implementation of RL algorithms in TCP toycase scenario: https://github.com/giuliapuntoit/RL-tcp-toycase
[3] SARSA implementation example: https://www.geeksforgeeks.org/sarsa-reinforcement-learning/?ref=rp
[4] How to evaluate RL algorithms: https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e
[5] Yeelight smart bulb: https://us.yeelight.com/