Task Offloading in Computing Continuum Using Collaborative Reinforcement Learning

Alberto Robles-Enciso(✉) and Antonio F. Skarmeta

Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain
{alberto.roblese,skarmeta}@um.es

Abstract. One of the challenges in the Computing Continuum paradigm is the optimal distribution of the generated tasks between the devices in each layer (cloud-fog-edge). In this paper, we propose to use Reinforcement Learning (RL) to solve the Task Assignment Problem (TAP) at the edge layer and then we propose a novel multi-layer extension of RL (ML-RL) techniques that allows edge agents to query an upper-level agent with more knowledge to improve the performance in complex and uncertain situations. We first formulate the task assignment process considering the trade-off between energy consumption and execution time. We then present a greedy solution as a baseline and implement our two RL proposals in the PureEdgeSim simulator. Finally, several simulations of each algorithm are evaluated with different numbers of devices to verify scalability. The simulation results show that reinforcement learning solutions outperformed the heuristic-based solutions and our multi-layer approach can significantly improve performance in high device density scenarios.

Keywords: Internet of things · Edge computing · Computing continuum · Task offloading · Resource allocation · Reinforcement learning

1 Introduction

Latency-critical applications are a major concern in today's networks as they are saturated by a large number of devices continuously sending tasks. The edge computing paradigm is able to minimise end-user latency but has limited computing capacity, therefore to improve its performance, Computing Continuum proposes the combination of the edge and the cloud in a single interconnected workflow. However, to make efficient use of devices it is necessary to define a computation-offloading framework. Each edge device receives tasks with specific requirements and has to decide whether to perform the computation itself or offload the task to another edge node or the cloud.

This problem is the "*task assignment problem*" and is a combinatorial optimization problem defined as the process of determining where the computation of each task is performed in order to minimise certain parameters, such as the aforementioned latency and energy consumption. In our article we designed an edge computing architecture in which edge devices receive tasks and can process them locally or send them either to another edge device, to the fog server or the cloud. The main contributions of this work as the following:

– We define the task assignment problem in edge computing as an optimisation problem with a trade-off between latency and energy consumption.
– We introduce a reinforcement learning algorithm for the task offloading decision. Each edge device will be an RL agent that can decide to compute its tasks locally or send them to the edge, fog or cloud layer.
– We propose a novel RL approach based on a multi-layer system in which the RL agents of the devices can delegate the offloading decision to an agent of a higher layer.
– The performance of the proposal is compared with a Greedy and single-layer RL algorithm, showing that the proposed solution is superior to the other algorithms.

This paper is organized as follows. In Sect. 2 we explore the state of the art of the task assignment problem. In Sect. 3 we formulated the assignment problem with its main components. In Sect. 4 we introduce a simple reinforcement learning algorithm and propose our novel multi-layer RL approach. Finally, in Sect. 5 we evaluate the proposed algorithms and present the results.

2 Related Work

The task assignment problem can be solved by several methods using very different techniques [4] such as convex optimization techniques, Lyapunov optimization [13], Hungarian algorithm [7] and novel genetic algorithms [19]. In addition, new methods based on dynamic programming and machine learning [17] techniques have emerged, such as reinforcement learning (RL) and neural network reinforcement learning (Deep RL). Nonetheless, it is difficult to find optimal solutions to the task assignment problem, especially given the prohibitive computational complexity in IoT devices, so in practice heuristic-based techniques or methods that search for suboptimal solutions are often used.

One of the most common techniques are greedy algorithms, which provide sub-optimal solutions but at a low computational cost. In some cases, it is possible to achieve solutions very close to the optimal solution [11,20]. Similarly, an alternative approach to solving optimisation problems are algorithms based on metaheuristics, the most popular are Genetic Algorithms which are inspired by the process of natural selection [5,6].

On the other hand, Reinforcement Learning (RL) is a novel technique based on machine learning that is not part of the well-known supervised and unsupervised learning paradigms. The purpose of reinforcement learning is to learn an optimal, or near-optimal, policy that maximizes the reward function and provides an optimal set of actions for different agent states and environmental conditions. RL algorithms learn iteratively through the immediate rewards they receive each time they perform an action based on their state [9,16].

In some cases, it is not possible to make use of a reinforcement learning algorithm directly, such as in scenarios where the agents' state is a large number of variables, making Q-Learning algorithms inefficient, or even when the state variables are not discrete. To address these limitations, researchers propose the use of neural networks to model the agent's learning process [2,18]. In general, Reinforcement Learning and Deep RL algorithms have a common problem, the convergence time. This type of algorithm requires a series of iterations to reach the optimal solution, and in some cases using random factor policies may take even longer to reach the optimal solution [3].

Finally, some approaches in the literature use other novel techniques to solve TAP. In [12] Dadmehr Rahbari and Mohsen Nickray use classification and regression trees to solve the problem. In [1] Mainak Adhikari et al. design a delay-dependent priority-aware offloading (DPTO) strategy for scheduling and processing tasks, generated from IoT devices to suitable computing devices. In [8] Lindong Liu et al. propose a supervised machine learning approach to solve the TAP based on classification data mining technique.

3 Task Assignment Problem

Computing continuum systems are composed of a large number of heterogeneous devices with different characteristics and roles. Some devices have high computational power and serve as a host for processing tasks, while others with lower computational power constantly generate tasks for the applications they run. This forms a layered architecture where devices are separated into levels according to their role.

On top of this architecture appears a flow of offloaded tasks, as some lower-capacity devices decide to send tasks to more powerful devices for processing. We define a task as an indivisible piece of computation generated by a particular application, which has its own characteristics and constraints such as maximum latency, data size and computational resources required. One of the key components of these architectures is the Task Assignment Problem since it is necessary to determine the best possible distribution of tasks between devices at each layer.

3.1 System Model

Our proposed system consists of devices that are separated into three layers, depending on their role, Fig. 1 shows the proposed offloading architecture. The layer closest to the users is the edge layer, which consists of heterogeneous edge devices that might have an intermittent connection and a dynamic position. This layer has the lowest latency and computational capacity, and is where tasks are generated from the edge devices that host IoT applications. Tasks can be processed locally or sent to devices in any of the three layers according to the decision of the offloading algorithm.

The fog layer is the middle layer where fog Servers are deployed. Fog Servers are small datacenters with intermediate computational capacity located between edge devices and the cloud, hence they have intermediate latency.

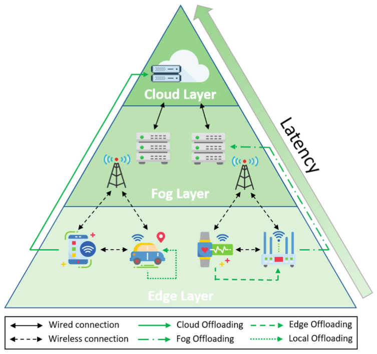


Fig. 1. Edge computing architecture.

The upper layer is the cloud layer, which has very high computational capacity and medium-high latency. The cloud acts as a single device, but in real deployments it is typically a large number of high-performance computers in a datacenter, therefore its computational capacity can be very high. However, their latency is also high due to the distance to end-users.

In the edge layer each edge device executes an offloading algorithm that decides, using the local information of the device, where it offloads the tasks it generates. We call to the decision-making system that uses the perception of the environment an agent.

3.2 Problem Definition

As shown above, a device can decide to execute a task locally ($a = 0$) or send it to a adjacent node ($a = 1$), a fog server ($a = 2$) or the cloud ($a = 3$), resulting in a specific cost as a weighted sum of execution time and energy consumption.

We formulate the cost of our optimization problem as a piecewise function that depends on the offloading action, which we define in detail in a paper still under revision. In a real environment, it may be possible that the offloading process fails ($a = -1$), so it is necessary to define a penalty cost δ in the segmented function Eq. 1.

Therefore, our proposal consists of designing an optimisation scheme in which the cost resulting from the allocation of tasks ($\boldsymbol{K}$) produced by a device $\boldsymbol{d}$ is minimised. Therefore, each device $\boldsymbol{d}$ aims to perform the best possible assignment of actions for each task $\boldsymbol{k}$ to minimise the resulting cost of all assignments.

The optimization problem is formulated in Eq. 2.

$$C_{d,k}(a) = \begin{cases} C^{l}_{d,k} & \text{if } a \text{ is equal to } 0 \\ C^{m}_{d,k} & \text{if } a \text{ is equal to } 1 \\ C^{f}_{d,k} & \text{if } a \text{ is equal to } 2 \\ C^{c}_{d,k} & \text{if } a \text{ is equal to } 3 \\ \delta_d & \text{if } a \text{ is equal to } -1 \end{cases} \tag{1}$$

$$G_d = \min_{a_k} \sum_{k=1}^{K} C_{d,k}(a_k) \tag{2}$$

subject to:

$$\sum_{k=1}^{K'} T^{d,k}_{mips} \leqslant D^{d}_{mips}, \forall k, \; L_{d,k} \leqslant T^{d,k}_{dl}, \; a_k \in \{-1, 0, 1, 2, 3\}$$

The optimization problem is subject to the following constraints: The tasks assigned to a device ($\boldsymbol{K'}$) must not exceed the computing capacity of the device, no task must exceed its maximum latency (deadline) and the possible actions that can be taken are error, local processing, offload to an adjacent node, offload to the fog layer and offload to the cloud layer.

4 RL-Based Task Offloading Algorithm

To solve the TAP defined in Eq. 2 we use a reinforcement learning approach based on Tabular Q-Learning. Each edge device runs its own reinforcement learning algorithm to explore the optimal task offloading policy by minimizing the long-term cumulative discounted cost. Figure 2 shows an overview of our agent.

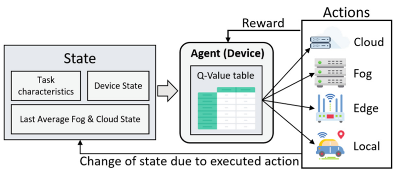


Fig. 2. Q-Learning agent.

When a task is received, the agent will decide offloading action ($a_t \in A = \{0, 1, 2, 3\}$), whenever possible, whether to process the task locally (*action 0*), send it to a nearby node (*action 1*), send it to the fog layer (*action 2*) or send it to the cloud (*action 3*). The decision will depend on the environment, which is based on the characteristics of the task, the state of the device and the last average state of fog and cloud.

The learning process uses a Q-Value table to store and query the value of the Q-function for each state-action. When an action is performed, the new Q-Value in the table is updated according to the following one-step Q update formula:

$$Q(s_t, a_t) = (1 - \alpha)\, Q(s_t, a_t) \; + \; \alpha\, (C_t + \gamma \min_a Q(s_{t+1}, a)) \tag{3}$$

The reward obtained after the execution of an action is a piecewise function of two elements that depends on the execution time of the task. If the execution time of the task ($T^t_{end} - T^t_{start}$) is less than its deadline (T^t_{dl}), the reward obtained is a weighted sum between the execution time and the energy consumption of the task (T^t_{energy}). Otherwise, the reward is the same but multiplied by a penalty δ factor.

$$C_t = \begin{cases} (T^t_{end} - T^t_{start}) + \beta T^t_{energy} & T^t_{end} - T^t_{start} < T^t_{dl} \\ \delta \cdot ((T^t_{end} - T^t_{start}) + \beta T^t_{energy}) & \text{otherwise} \end{cases} \tag{4}$$

The reward function only considers energy and execution time, all other parameters are part of the state and do not need to be included as they will have an indirect impact on the latency.

In this basic RL approach, each device works independently using its local information and aggregated global information. Thus, decisions are made according to the local state and knowledge of the agent. However, the biased view of the environment and the lack of knowledge in the early stages of the algorithm causes low performance in complex situations. To overcome this drawback, we propose to allow the RL agent of a device to delegate the offloading decision to a upper level agent in case it does not have enough information. The upper level agent, e.g. deployed in the fog layer, will decide according to its knowledge and the global state of the system. The offloading decision will be sent to the querying device and both the local and the upper level agent will learn from the reward obtained after executing the action. Figure 3 shows the process of the offloading query.

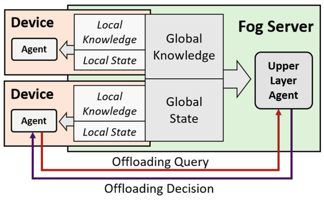


Fig. 3. Offloading query process.

In this enhanced architecture, both edge devices and fog servers run an independent RL algorithm that can collaborate between layers. The offloading query allows a passive knowledge transfer from fog agents to edge agents, especially useful when an edge agent starts with no knowledge and performs queries to learn from the decisions of the upper level agent.

We extend the functionality of the RL algorithm to include in the set of feasible actions of edge devices the new offloading query action. Edge devices and fog servers execute the same algorithm but with different behaviour, as they have a different view of the environment.

The edge device agent now has the action $(A \cup \{4\})$ of performing the offloading query. When a reward is received for an action queried to a fog server, it is considered to have a higher value since it is assumed that the upper agent will take better decisions. Also, the Q-Value of the query action is penalized as it is used to decrease the probability of being selected while increasing the agent's local knowledge. On the other hand, the upper layer agent waits to receive offloading queries from other agents and uses its own knowledge and view of the current state, including cloud and fog realtime CPU usages, to take an offloading decision and send it back. As a result, a reward will be obtained from the execution of the offloading action that will improve the knowledge of both the agent who made the request and the upper layer agent.

Algorithm 1: ε-greedy Multilayer Q-Learning Algorithm

```
Parameters: discount factor $\gamma$, learning rate $\alpha$, exploration rate $\varepsilon$, penalty factor $\delta$, query reward factor $\rho$
            and query use penalty $\omega$
1  begin
2      for each step $t$ do
3          Observe actual state $s_t$
4          Determine feasible action set $A'$ from $A$
5          $isQuery \leftarrow false$
6          $e \leftarrow$ random number from [0,1]
7          if $e < \varepsilon$ then
8              $a_t \leftarrow$ randomly select an action from $A'$
9          else
10             $a_t \leftarrow \arg\min_{a \in A'} Q(s_t, a)$
11         end
12         if $a_t$ is to ask a fog server then
13             $isQuery \leftarrow true$
14             Send the offloading request to a fog server
15             $a_t \leftarrow$ get the fog server decision
16         end
17         Execute or send the offloading action $a_t$
18         Wait for the task to be completed
19         Observe new state $s_{t+1}$
20         Calculate reward $C_t$ by (4)
21         if $isQuery$ then
22             $C_t \leftarrow \rho \cdot C_t$
23             $C_t^q \leftarrow \omega \cdot t \cdot C_t$
24             Update $Q(s_t, 4)$ using (3) with $C_t^q$
25         end
26         Update $Q(s_t, a_t)$ according to (3) with $C_t$
27     end
28 end
```

The proposed multilayer solution to the offloading problem based on ε-greedy Q-Learning is shown in Algorithm 1. As mentioned before, all devices run the same algorithm but with their own view of the state and their own knowledge. This means that each device will manage an independent Q-Table that will be trained locally. In addition, the upper layer agent will take advantage of all interactions with the devices to update their global status.

5 Performance Evaluation

In this section we will evaluate our proposed solution compared to a greedy and a single layer reinforcement learning approach in a simulated edge computing environment. We will perform simulations to compare the performance of each algorithm using a set of metrics. The simulation results are available in our GitHub repository [15].

5.1 Methodology

The purpose of our evaluation is to obtain enough data to fairly compare the offloading algorithms. Therefore, we will make use of an edge computing simulator to test the behaviour of the algorithms from low-density to high-density device scenarios. The output of each simulation will be a set of metrics used to determine the performance of the algorithm in the specific simulation scenario. To prevent inaccurate results caused by the random component of the simulator, the metrics will be calculated by averaging the result of several simulations on the same conditions.

5.2 Experiment Setup

The evaluation has been performed on a modified version of PureEdgeSim v4.2 [10], the source code of our extension is available on GitHub [14]. The simulated edge computing scenario consists of three layers of devices (edge-fog-cloud) randomly distributed over an area of 200×200 metres.

To verify the scalability of the proposed algorithms, the number of edge devices in each simulation is increased by 10 until 200. Each simulation lasts 10 min and is executed 10 times per configuration to calculate the average result. The bandwidth of the connection between devices is 100 megabits per second at the edge and fog layers, while the connection to the cloud layer is 20 megabits per second. The maximum range of the wireless connection of the edge devices is 40 m. The simulation parameters of the PureEdgeSim environment are summarised in Table 1.

5.3 Metrics

To compare the performance of each algorithm we define the following benchmark metrics:

- **Task Success Rate:** The percentage of the tasks that finish their execution over the total. A task is not considered to complete its execution correctly if its execution time exceeds its deadline or if the offloading process fails. This metric is one of the most important for the evaluation.
- **Average Total Time:** The total time required to complete successfully a task, which includes the execution time and the time to send the task to the processing node. This metric is especially useful for comparing the latency incurred by each algorithm.

Table 1. PureEdgeSim simulation parameters

Simulation Parameter	Value
Simulation duration	10 min
Number of averaged simulations	10 per configuration
Min number of edge devices	10
Max number of edge devices	200
Simulation area	200 m × 200 m
Edge and Fog Bandwidth	100 Mbps
Cloud Bandwidth	20 Mbps
Edge devices range	40 m

- **Complete Average Total Time:** Same as **Average Total Time** but also considering the time wasted on tasks that were not executed successfully.
- **Failed tasks due to latency:** The number of tasks that have failed because their execution time exceeds their maximum allowed latency.
- **Average CPU Usage per device:** Average CPU usage of a device. Useful to determine how much the computational resources of the devices are used.
- **Average Energy Consumption per Device:** Average power consumption of one device.

5.4 Compared Methods

We have implemented in the simulator three algorithms to evaluate their performance. The greedy solution will serve as a reference for comparison with the single-layer RL algorithm and our proposed multi-layer guided RL.

In addition, we designed three methods based on the implementations of the reinforcement learning solutions explained in previous sections. The first one is the basic implementation of a RL algorithm that runs locally on each device without external knowledge, in the tests we will denote it as "*Local RL*". The second and third methods are the same implementation of the multi-layer RL algorithm but with different initial conditions. The "*RL Multilayer Empty*" version starts each simulation with all Q-Tables (knowledge) of the devices completely empty, while "*RL Multilayer*" version uses on the fog servers the Q-Table resulting from the previous simulations, with the same configuration, to simulate the behaviour of a system that starts with knowledge to improve initial performance. The parameters used by both methods are summarised in Table 2.

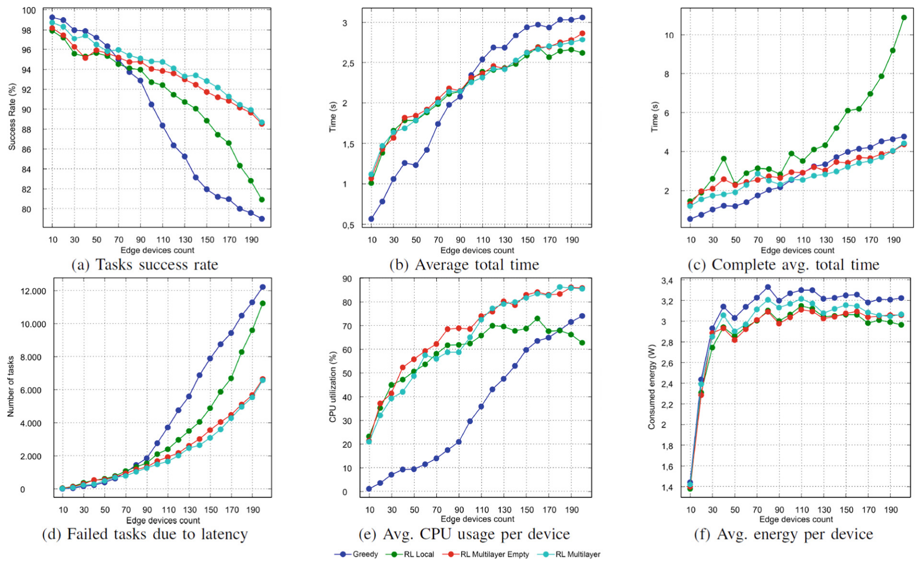
Table 2. Reinforcement learning algorithm parameters

Parameter/RL Algorithm	Single	Multi
Learning rate α	0.6	0.6
Latency-Energy Trade-off β	0.003	0.003
Discount factor γ	0.3	0.3
Failure penalty δ	1000	1000
Average CPU refresh rate	60 s	60 s
Query reward factor ρ	-	0.2
Query use penalty ω	-	10
Initial Q-Value	200	200
Initial Query Q-Value	-	10

5.5 Experimental Results and Analysis

In this section we will show the most important results of the simulations performed for each algorithm and configuration. Each of the subfigures of 4 represents the metrics that were defined to make the comparison between algorithms.

One of the most critical results is the success rate in task execution, since in practice this has the most negative impact on the end-user. Figure 4a shows the success rate resulting from each algorithm when performing the simulation.



(a) Tasks success rate (b) Average total time (c) Complete avg. total time (d) Failed tasks due to latency (e) Avg. CPU usage per device (f) Avg. energy per device

Fig. 4. Simulations results

As we can see, in low device density scenarios, the greedy method outperforms the others until it reaches a medium density, 70 devices, where its success rate starts to drop. In the high device density scenarios the performance of the greedy method and the RL single-layer are very low while both multi-layer methods are able to keep an acceptable performance.

This behaviour is due to the fact that in low device density scenarios there are not a large number of tasks and most of them can be executed by the fog and cloud servers without saturation, so the heuristics of the greedy method gives a better result than the reinforcement learning algorithms. When a medium density of devices is reached, the appropriate use of resources becomes more relevant and algorithms using reinforcement learning techniques are able to adapt dynamically to keep the success rate as high as possible. In high density scenarios with a large number of tasks the optimal use of processing nodes is critical, therefore the greedy method cannot achieve good results and even the single-layer RL method cannot improve the result. In contrast, multi-layer RL methods achieve a high success rate due to the possibility of delegating offloading decisions to higher level agents. In fact, the best success rate is achieved with multi-layer RL method that start with the Q-Table of the fog servers filled with the values learned from previous simulations since it allows to provide useful knowledge to the devices in the early stages of the learning process.

The average time required to complete a task for each algorithm and number of devices is shown in Fig. 4b. Similar to the success rate, the average total time of the greedy algorithm drastically changes its performance based on the number of devices, while reinforcement learning algorithms slowly change the average total time. The three RL methods provide a similar average total time as this metric only considers tasks that have successfully completed their execution.

If we consider the time lost due to tasks that do not execute correctly because of latency, we can see in Fig. 4c the real impact of the algorithm's actions when deciding to do an unsuitable offloading. The behaviour of the greedy algorithm is similar to Fig. 4b, but the single-layer RL method substantially increases the time in high device density scenarios as the impact of bad decisions in complex situations is very high. In contrast, multi-layer RL methods avoid the initial uncertainty by delegating the decision, thereby making better offloading decisions that reduce latency failures as can be seen in Fig. 4d.

One more relevant result that can be analysed is the average CPU usage per device, which indicates the degree of utilization of the system's computational resources. A proper distribution of tasks among the devices results in a high average CPU usage per device as resource utilisation is maximised. In contrast, low CPU usage indicates that the algorithm is saturating a few devices while many others are idle. As shown in Fig. 4e, which represents the simulator output for this metric, the two multi-layer RL methods stand out from others, and the greedy method shows a low use of computational resources.

Similarly, the performance of algorithms can be measured in terms of their energy consumption as this is one of the components of the optimization problem, Fig. 4f shows the average energy consumption per device obtained from the simulations. The greedy algorithm presents the highest energy consumption while the RL algorithms show the lowest energy consumption.

After having seen the performance of the algorithms in different simulator scenarios, we can conclude that the greedy algorithm offers acceptable performance in low and medium device density scenarios. However, as device density increases, more complex methods must be applied to maintain system performance. Reinforcement learning algorithms are able to adapt to complex scenarios at a low computational cost, thus providing the best results in simulations. Furthermore, our multi-layer approach stands out from other methods because in complex high-density scenarios it shows high performance in the most important metrics. This improvement is due to enhanced offloading decision system by using external knowledge and serves as evidence of the good performance of our multi-layer RL proposal. Therefore, reinforcement learning algorithms are good methods for solving the task assignment problem and our proposal is a useful and easily applicable extension to any RL algorithm to improve its performance.

6 Conclusion

In this paper, we have presented the task assignment problem as a key component of collaborative edge computing architectures. As shown in the first section, there are several methods for solving TAP, but those based on artificial intelligence are the most promising. Reinforcement learning is presented as a solution for the task offloading process in our proposed three-layer edge-fog-cloud architecture.

In this work we have studied different configurations to understand the impact of task distribution and limited vision of RL agents and how this impacts the performance behaviour of the algorithm in complex situations. To overcome these drawbacks, we propose a novel extension of reinforcement learning techniques that allows agents to query an upper-level agent with more knowledge and a broader view of the environment.

We have implemented our proposals together with a greedy alternative in a modified version of PureEdgeSim simulator and performed several tests to compare the performance of each algorithm in different situations following a set of metrics, providing access to the results and simulations for reproducibility. The experimental results showed that, compared with the greedy and classical RL algorithms, under multiple conditions, our proposed multilayer RL algorithm achieved much better performance in scenarios with a high number of devices and tasks.

Acknowledgments. This work was supported by the FPI Grant 21463/FPI/20 of the Seneca Foundation in Region of Murcia (Spain), partially funded by project PID2020–112675RB–C44 and PTAS–20211009 MCIN/AEI/10.13039/501100011033 and by the “European Union NextGenerationEU/PRTR”.

References

1. Adhikari, M., Mukherjee, M., Srirama, S.: DPTO: a deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. IEEE Internet Things J. **7**, 5773–5782 (2020)
2. Alfakih, T., Hassan, M., Gumaei, A., Savaglio, C., Fortino, G.: Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access **8**, 54074–54084 (2020)
3. Argerich, M.F., Fürst, J., Cheng, B.: Tutor4RL: guiding reinforcement learning with external knowledge. In: AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering (2020)
4. Bellendorf, J., Ádám Mann, Z.: Classification of optimization problems in fog computing. Futur. Gener. Comput. Syst. **107**, 158–176 (2020). https://doi.org/10.1016/j.future.2020.01.036
5. Ghanavati, S., Abawajy, J., Izadi, D.: An energy aware task scheduling model using ant-mating optimization in fog computing environment. IEEE Trans. Serv. Comput. **15**(4), 2007-2017 (2020)
6. Jia, Z., Yu, J., Ai, X., Xu, X., Yang, D.: Cooperative multiple task assignment problem with stochastic velocities and time windows for heterogeneous unmanned aerial vehicles using a genetic algorithm. Aerosp. Sci. Technol. **76**, 112–125 (2018). https://doi.org/10.1016/j.ast.2018.01.025
7. Liang, J., Long, Y., Mei, Y., Wang, T., Jin, Q.: A distributed intelligent hungarian algorithm for workload balance in sensor-cloud systems based on urban fog computing. IEEE Access **7**, 77649–77658 (2019). https://doi.org/10.1109/ACCESS.2019.2922322
8. Liu, L., Qi, D., Zhou, N., Wu, Y.: A task scheduling algorithm based on classification mining in fog computing environment. Wirel. Commun. Mobile Comput. **2018**, 1–11 (2018). https://doi.org/10.1155/2018/2102348
9. Liu, X., Qin, Z., Gao, Y.: Resource allocation for edge computing in IoT networks via reinforcement learning. In: ICC 2019–2019 IEEE International Conference on Communications (ICC), pp. 1–6 (2019)
10. Mechalikh, C., Taktak, H., Moussa, F.: Pureedgesim: a simulation framework for performance evaluation of cloud, edge and mist computing environments. Comput. Sci. Inf. Syst. **18**, 42 (2020). https://doi.org/10.2298/CSIS200301042M
11. Rahbari, D., Nickray, M.: Low-latency and energy-efficient scheduling in fog-based IoT applications. Turk. J. Electr. Eng. Comput. Sci. **27**, 1406–1427 (2019)
12. Rahbari, D., Nickray, M.: Task offloading in mobile fog computing by classification and regression tree. Peer-to-Peer Netw. Appl. **13**, 104–122 (2020)
13. Ren, C., Lyu, X., Ni, W., Tian, H., Song, W., Liu, R.P.: Distributed online optimization of fog computing for internet of things under finite device buffers. IEEE Internet Things J. **7**(6), 5434–5448 (2020). https://doi.org/10.1109/JIOT.2020.2979353
14. Robles-Enciso, A.: Pureedgesim RL extension (2021). https://github.com/alb1183/ML-RL-PureEdgeSim
15. Robles-Enciso, A.: ML-RL Simulations results (2022). https://github.com/alb1183/ML-RL-simulations
16. Sen, T., Shen, H.: Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems. In: 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), pp. 1–10 (2019)

17. Shakarami, A., Ghobaei-Arani, M., Shahidinejad, A.: A survey on the computation offloading approaches in mobile edge computing: a machine learning-based perspective. Comput. Netw. **182**, 107496 (2020). https://doi.org/10.1016/j.comnet.2020.107496
18. Wang, J., Zhao, L., Liu, J., Kato, N.: Smart resource allocation for mobile edge computing: a deep reinforcement learning approach. IEEE Trans. Emerg. Top. Comput. **9**(3), 1529–1541 (2019)
19. Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., Rovatsos, M.: Fog orchestration for internet of things services. IEEE Internet Comput. **21**(2), 16–24 (2017). https://doi.org/10.1109/MIC.2017.36
20. Zhang, G., Shen, F., Liu, Z., Yang, Y., Wang, K., Zhou, M.T.: FEMTO: fair and energy-minimized task offloading for fog-enabled IoT networks. IEEE Internet Things J. **6**(3), 4388–4400 (2019). https://doi.org/10.1109/JIOT.2018.2887229