Forest Fire Control with Learning from Demonstration and Reinforcement Learning

1st Travis Hammond
University of Groningen
Groningen, The Netherlands
dashdeckers@gmail.com

2nd Dirk Jelle Schaap
University of Groningen
Groningen, The Netherlands
d.j.schaap@student.rug.nl

3rd Matthia Sabatelli
Montefiore Institute
Liège, Belgium
m.sabatelli@uliege.be

4th Marco A. Wiering
University of Groningen
Groningen, The Netherlands
m.a.wiering@rug.nl

***Abstract*—This paper describes a novel approach to control forest fires in a simulated environment using connectionist reinforcement learning (RL) algorithms. A forest fire simulator is introduced that allows to benchmark several popular model-free RL algorithms that are combined with multilayer perceptrons that serve as a value function approximator. For our experiments, we test in total four different algorithms: Q-Learning, SARSA, Dueling Q-Networks and a novel algorithm called Dueling-SARSA. To enable the algorithms to better cope with the difficulty to contain the forest fires when they start learning, we use demonstration data that is inserted in an experience-replay memory buffer before learning. In the experiments, the performance of these algorithms are compared under different experimental setups ranging from the complexity of the simulated environment to how much demonstration data is initially given. The results show that the demonstration data are necessary to learn very good policies for controlling the forest fires in our simulator and that the novel Dueling-SARSA algorithm performs best.**

***Index Terms*—Reinforcement learning, Multilayer perceptrons, Forest fire control, Dueling-SARSA**

I. Introduction

In 2019 there were very large forest fires in Siberia, the Amazon, Central Africa and Australia. Forest fires result in the tragic loss of lives and houses and have very large ecological consequences. Trees and plants are a key factor in the carbon cycle [1]. Using photosynthesis massive amounts of CO_2 are filtered from the atmosphere and stored. When fires destroy large forests, all this stored CO_2 is released back into the atmosphere, which leads to more global warming and this will also increase the likelihood and risk of future forest fires.

Fighting these forest fires is a challenging task. To extinguish a fire one or more of the following three required elements have to be eliminated: fuel, heat or oxygen. The ordinary tactic is to remove the heat and oxygen by spraying water or foam from hoses or aircraft, but large forest fires require more effort to be contained. Possible options include burning down specific areas in a controlled fashion or using bulldozers to cut fire lines. This paper focuses on the latter approach.

There is still not much research being done in the field of artificial intelligence to optimize forest fire control strategies. The CHARADE project [2] led to the first large software platform for the development of intelligent decision support systems and has been used to construct plans for controlling some forest fires. The planning system used case-based reasoning and was integrated with a Geographic information system and a model for simulating forest fires. Other research has been done on the detection and prediction of forest fires. In [3], the authors outline how reinforcement learning algorithms could be used to optimize forest fire control policies by interacting with a forest fire simulator. Later research explored the use of the enforced sub-populations (ESP) algorithm to evolve neural network controllers capable of controlling forest fires in a simulated environment [4], and a model of multi-agent coordination in fire fighting scenarios [5].

Contributions. In this paper, we explore how connectionist reinforcement learning (RL) can be used to allow an agent to learn how to contain forest fires in a simulated environment by using a bulldozer to cut fire lines. We introduce a new RL algorithm called Dueling-SARSA and compare it to three existing algorithms: Q-Learning [6], SARSA [7] and Dueling Q-Networks [8]. To deal with the large number of possible states, all algorithms are combined with multilayer perceptrons to learn the state-action value function. Because an agent trained from scratch will initially almost never stop an expanding forest fire, we employ an algorithm to generate initial demonstration data that can be easily integrated in the experience replay method [9]. Different experiments are performed with two different sizes of the forest area and we examine using different amounts of demonstration data. The results show that the use of enough demonstration data is important to successfully control the forest fires in the simulator. Furthermore, the results show that the novel dueling-SARSA algorithm obtains the best performances of all tested algorithms.

Paper outline. In Section II we describe the used reinforcement learning algorithms. Then Section III explains the workings of the forest fire simulator, the used reward function and the state representation. Then Section IV gives the experimental setup, and Section V presents the results. Finally, we conclude the paper in Section VI.

II. Reinforcement Learning

Reinforcement learning [10] is a machine learning paradigm that consists of an agent interacting with an environment. These environments can come in different forms, ranging from board-games and video-games [11, 12], to sophisticated robotic simulators [13]. In this work, we consider a simulation of a forest fire that needs to be contained. At each discrete time step $t \in \{1, 2, 3..., T\}$, the environment provides the agent with an observation of the environmental state $s_t \in \mathcal{S}$. This allows the agent to interact with the environment by choosing an action a_t from a set of possible actions $\mathcal{A} = \{1, ..., K\}$. The result of taking a particular action is a new state s_{t+1} which is associated with a reward r_t. The interaction between the agent and the environment can be modelled as a Markov Decision Process (MDP) where the probability of visiting state s_{t+1} is only dependent on the previous state s_t and the performed action a_t. The goal of the agent is to select actions such that the cumulative future reward from the current time step t is maximized. This is defined as:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{1}$$

where T is the time-step at which the simulation terminates and $\gamma \in [0, 1[$ is the discount factor that determines the trade-off between the importance of immediate and delayed rewards. A policy π is a mapping of states to actions (or distribution over actions). To determine the optimal policy π^* that leads to the highest reward as defined in Equation (1), the optimal action-value function (also known as Q^*) is defined as:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi]. \tag{2}$$

The Q-function can be computed by using dynamic programming methods that iteratively update the Bellman equation:

$$\begin{gathered} Q_{i+1}(s, a) = \textstyle\sum_{s'} P(s'|s, a)[R(s, a, s')+ \\ \gamma \max_{a'} Q_i(s', a')]. \end{gathered} \tag{3}$$

In this case $P(s'|s, a)$ is the probability of observing state s' after executing action a in state s, and $R(s, a, s')$ is the expected reward obtained after executing action a in state s and ending up in state s'. Such a value iteration algorithm will eventually converge to the optimal Q-function Q^* as $i \rightarrow \infty$ [10]. Once the Q-function is learned, the optimal policy can easily be derived by simply taking the highest-valued action in each state. For most practical problems, the transition function is not known and the number of possible states is very large, therefore dynamic programming cannot be used. In this case, connectionist reinforcement learning can be used instead. In connectionist reinforcement learning, the goal is to approximate this function $Q(s, a; \theta) \approx Q(s, a)$ with a neural network parameterized by θ, which is trained by an RL algorithm.

A. Reinforcement Learning Algorithms

We start by defining the neural architecture used by our RL agent: we use a single hidden-layer perceptron with 50 hidden units that are activated by a sigmoid non-linearity. The network takes as input a state representation and outputs an array of size K ($K = 4$ in our experiments), which corresponds to the Q-values of taking any of the possible actions. The network is trained with the `Adam` optimizer [14] over mini-batches of experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ which are stored in an experience-replay memory buffer $B_t = \{e_1, ...e_t\}$. The buffer allows for the storage of M experiences, once it is full the oldest experiences will be discarded to make room for new ones. Using an experience-replay buffer is a well-known strategy for improving the stability of RL algorithms which use a function approximator: it decorrelates the inputs that are used for training and avoids feedback loops which could make the network parameters diverge [12, 15]. Furthermore, the experience replay buffer also increases sample efficiency since it allows experiences to be reused multiple times over training. As will be explained later, another important advantage is that such memory-buffers can get pre-initialized with experiences before training starts, therefore providing the agent with useful demonstration data. The objective function that is used for training the neural network is either based on the Q-Learning [6] algorithm or on the SARSA [7] algorithm, a choice which is motivated by investigating the difference in performance between an *off-policy* learning algorithm and an *on-policy* learning one. In the first case the loss function that is minimized is:

$$L_i(\theta_i) = \sum_{(s,a,r,s') \sim U(B)} [(Y^{Target} - Q(s, a, \theta_i))^2]. \tag{4}$$

Where B is the experience-replay memory buffer from which experiences are uniform randomly sampled, and Y^{Target} is the target value for an experience defined as:

$$Y^{Target} = r + \gamma \max_{a} Q(s', a; \theta_t). \tag{5}$$

Note that this temporal difference target is computed by the target-network θ_t, which is a periodically updated frozen copy of the Q-network to make learning more stable [12]. The SARSA algorithm is a little bit different since as the acronym suggests, it also requires saving the next action into the memory buffer with the tuple (s, a, r, s', a') and uses this next action to compute the target value.. The loss function that is minimized by the neural network is therefore:

$$L_i(\theta_i) = \sum_{(s,a,r,s',a') \sim U(B)} [(Y^{SARSA} - Q(s, a, \theta_i))^2]. \tag{6}$$

where

$$Y^{SARSA} = r + \gamma Q(s', a'; \theta_t). \tag{7}$$

Note that SARSA also uses a target network in our research and all four RL algorithms do. One limitation of neural networks which are based on Q-Learning like algorithms is that they are not able to estimate the value of a state and the state-action values separately. This ability can however be very useful as originally presented in [8]. The dueling network architecture achieves this by having two streams each predicting either the value of a state, or the action advantages $A(s, a)$ for all possible actions. When a multilayer perceptron is used the two are merged together based on the following equation:

$$Q(s, a; \alpha, \beta) = V(s; \beta) + \Big(A(s, a; \alpha) - \tfrac{1}{K} \sum_{a'} A(s, a'; \alpha)\Big), \tag{8}$$

where α and β denote the weights of the two fully connected layers and K is the number of possible actions. This equation automatically prevents the state value layer from estimating anything related to the action advantages, since the sum of the advantages is kept to zero. The equation results in the Q-values which can be used in the same way as with the single stream Q-network. Basically, the dueling Q-Network uses another architecture to estimate the Q-function with some constraints on the values learned by the state-value and advantage function streams.

We combine this dueling approach with the target networks and experience replay memory-buffers both for training the Q-Learning algorithm and the SARSA algorithm. The latter algorithm is called Dueling-SARSA, which to the best of our knowledge is a novel algorithm that combines the SARSA algorithm with the dueling approach. The idea is very similar to the Dueling Q-Network, but the targets for the Q-values are determined by the update rule for SARSA. Because SARSA is an on-policy algorithm, Dueling-SARSA is also an on-policy algorithm.

III. Forest Fire Simulator

In this section we describe the environment which we have created for simulating and controlling forest fires. We will also explain the used reward function and the state representation.

A. Environment

We simulate the forest fire by using a grid of cells that comes in the shape of a square of either size 10×10 or 14×14. Each cell has several attributes that are related to how forest fires can spread in real-world situations. These attributes are the following:

- Heat-Potential: the amount of heat each cell can radiate to its neighbor cells once it is ignited. This has the effect of increasing the neighboring cells' temperatures over time.
- Ignition-Threshold: a threshold parameter on a cell's temperature, which once reached will make the cell ignite and start burning.
- Temperature: a parameter that defines the temperature of a cell. Once the temperature is equal to the ignition-threshold a cell will start to burn.
- Amount of Fuel: keeps track how much fuel is present in each cell. At each burning-iteration, each cell consumes fuel until there is no more fuel left. This makes the considered cell a dead (burned) cell.

The heat from a burning cell reaches its neighbor cells directly north, south, east, and west. If that cell is flammable its temperature is increased by the heat potential of the burning cell, otherwise, nothing happens. For a visual representation of the environment please see Figure 1. The green cells represent trees, and therefore cells that can become ignited. The agent is represented by the white tile. Wherever it moves it destroys the trees and an empty, inflammable (brown) cell is formed. A line of these dug cells forms a fire line over which the fire cannot spread. Dead cells are represented in black. At each time step, the agent has to move either north, south, east or west and is not allowed to idle on the same cell. The agent is always digging while it moves. The simulator reaches a terminal state and restarts if the agent dies (by entering a burning cell), or if there are no more burning cells.

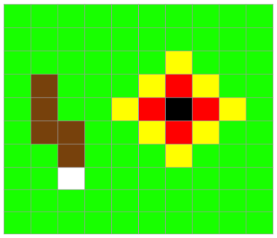

Fig. 1: A visual representation of the environment. The agent is shown in white, leaving behind a trail of inflammable dug cells (shown in brown). The trees (green) can ignite to become burning cells (red), which heat up the neighbouring cells (yellow). When a burning cell runs out of fuel it dies (black).

B. Reward Function

The reward function of the environment is defined by the following equation:

$$r_t = \left\{ \begin{array}{lr} 1000, & \text{Fire is contained} \\ 1000 * (p), & \text{Fire burns out} \\ -1000, & \text{Agent dies} \\ -1, & \text{Otherwise} \end{array} \right\}, \tag{9}$$

where p is the percentage of the grid which is not damaged by either a fire or by the digging of the agent. Different rewards emitted at the same time step are added together. A total reward of 1700 would mean that the agent successfully controlled the fire and saved around 75% of the forest area. The containment of a fire is defined as

$$\sum_{f \in \mathcal{F}} \sum_{b \in \mathcal{B}} astar(f, b) = 0, \tag{10}$$

where f is a burning cell from the set of currently burning cells $\mathcal{F}$ and b is a cell on the border of the grid from the fixed set of border cells $\mathcal{B}$. The function $astar$ is defined as

$$astar(f, b) = \left\{ \begin{array}{lr} 1, & \text{if A* path exists} \\ 0, & \text{Otherwise} \end{array} \right\}, \tag{11}$$

where a path is a sequence of directly connecting cells starting with cell f and ending with cell b, determined using the A* path-finding algorithm and not allowing diagonal steps. The intuition is that if there exists a path between any burning cell and any cell on the border of the map, then there exists a way for the fire to spread beyond control and therefore it is not contained.

C. State Representation

We represent the state of the environment, as it is visible to the agent, with three matrices of size N^2 with a boolean domain which results in a flattened array of $3 \times N^2$ boolean inputs. This allows to represent the environment with three grids: the first layer contains the position of the agent where all entries are represented by a 0 except for the one denoting the position of the agent which has a value of 1. The second layer represents the positions of the fire and follows a similar logic. Cells that are on fire are represented by a 1, while non-burning cells are set to 0. In the third layer, we represent the fire lines that are cut by the agent by again following this boolean approach. When the size of the grid is $N = 10$ this results in a state representation consisting of 300 inputs.

This state representation is shown graphically in Figure 2. Note that this representation can better represent a state when compared to a single matrix representing the color or gray-scaled map as input. The reason is that the grids contain meaningful information, which makes it easier for the neural network to learn the value function as was also shown in [16].

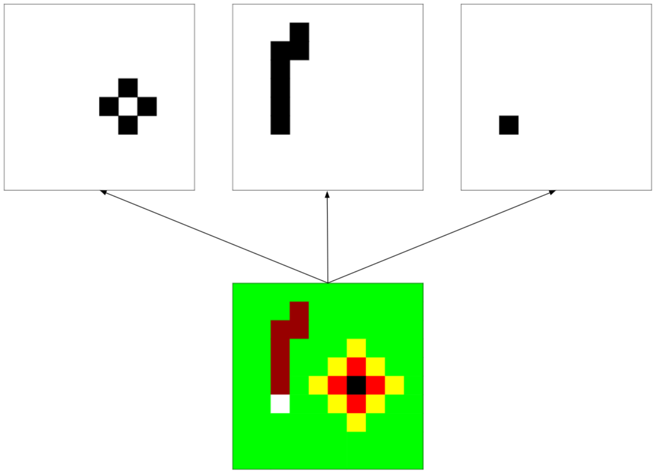

Fig. 2: An example of the state representation. Each layer shows an important aspect of the RL environment: The location of the fire, inflammable cells and the agent itself.

IV. Experimental Setup

The reward function defined in Section III-A provides delayed rewards. Positive rewards are only given at the end of the simulation if the forest fire is contained, and therefore the agent might require additional guidance to learn to contain the fire in a reasonable training time. To achieve this we decided to fill the experience-replay memory buffer with demonstration data already before the learning starts. The purpose of this demonstration data is to show the agent how it might be able to collect the containment reward. We have therefore developed a simple algorithm that makes the agent move around the fire clockwise by choosing randomly one of the two possible actions that lead the agent in the specified direction, (unless one of the actions would lead to the death of the agent in which case the agent chooses the safe action). These two possible actions depend on the position of the agent relative to the fire as shown in Figure 3. The environment is reset upon containment as defined in Equation (10). Only trajectories leading to successful containment are stored. This results in an average of 35 memories per episode when it comes to the 10×10 grid, and of ≈ 48 memories per episode when it comes to the 14×14 grid. In our experiments, we examine the influence of different amounts of demonstration data.

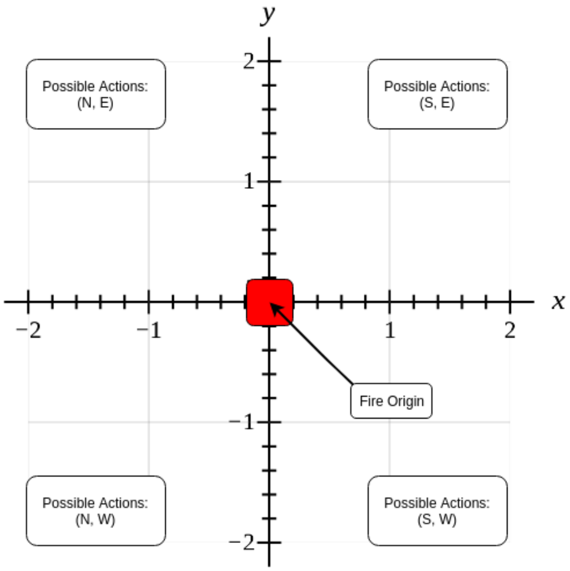

Fig. 3: The possible actions available to the agent based on the position (quadrant) of the agent relative to the origin of the fire.

.

To be able to reliably compare the performance of the different RL algorithms, we created an algorithm which serves as a baseline. The algorithm, shown in Algorithm 1, follows the same logic used for the creation of the demonstration data required by the memory buffers, except that it continues to run until the fire has burnt out and therefore does not stop as soon as the fire is contained. In step 4 of the algorithm, a random action from the two actions that lead the agent to the next subgoal is chosen.

Algorithm 1 Baseline algorithm to contain the fire

```
 1: procedure RunBaseline
 2:     totalreward = 0
 3:     while not done do
 4:         action = random(possible actions)
 5:         if action is dangerous then
 6:             action = other possible action
 7:         end if
 8:         reward, done = execute(action)
 9:         totalreward = totalreward + reward
10:     end while
11:     return totalreward
12: end procedure
```

We investigate the performance of both Q-Learning and SARSA, with and without the respective dueling extension, for a total of four different tested algorithms. The hyperparameters that have been used throughout all experiments are reported in Table I.

TABLE I: All relevant hyperparameters used for the training process. These values were selected by performing an informal search using the Q-Learning algorithm without dueling networks. The target network is updated every C episodes. The epsilon value is decayed after every episode.

Memory size	20000
Batch size	32
Target update (C)	20
Gamma (γ)	0.999
Alpha (α)	0.005
Epsilon decay (ϵ)	0.01
Epsilon maximum	1
Epsilon minimum	0.005

V. Results

We now report the results which were obtained with the experiments. For each algorithm and respective parameter combination, we ran 10 simulations of 10,000 episodes each. The performance of the different algorithms is compared with the baseline algorithm introduced before. We do this for two different grids of sizes with $N = 10$ and $N = 14$ respectively. Furthermore, we use three different methods of using demonstration data, which differ in the number of episodes that initialize the experience-replay memory buffer before training starts. We ran experiments after filling the buffer with 0, 100 and 1000 episodes. At the start of each simulation, the environment is initialized with trees, while a single cell at the center of the map is ignited. The agent starts at a random location on a circle centered around the initial burning cell with a radius of either 1, 2 or 3 cells. The distance to the fire center is randomly determined. All line plots represent averages of 10 simulation runs ± 1 std-deviation, while all tables report the final averaged performances on the final 2500 episodes.

A. Results on 10×10 *Environments*

We start by discussing the results obtained on a simulated environment of size 10×10. The first, second and third plots of each row of Figure 4 report the three cases in which the algorithms are given 0, 100 and 1000 episodes of demonstration data respectively. We can start by observing that all algorithms have difficulties to perform better than the baseline algorithm when no experiences from demonstrations are provided to the memory-buffer. Only the Dueling Q-Network is able to learn to perform quite well, although less than the baseline algorithm. All other algorithms perform similarly to each other and significantly worse than the Dueling Q-Network. The average performances on the final 2500 episodes are also shown in Table II.

In the second plot we can notice that the performance of the algorithms gets more similar among the different tested approaches. More notably, the Dueling Q-Network, which was the best performing algorithm in the previous experiment, is now the worst-performing one, while SARSA and Dueling SARSA improved their performance a lot when given 100 episodes of demonstration data. The average performances on the final 2500 episodes are also shown in Table III.

TABLE II: Averages of the last 2500 episodes given 0 episodes of demonstration data. The numbers in bold indicate the highest average and best rewards.

Algorithm	**Average Reward**	**Std. Error**	**Best Reward**
Baseline	**1129**	80	**1387**
Q-Network	221	283	715
SARSA	132	240	563
Dueling Q-Network	956	352	1335
Dueling SARSA	241	296	582

TABLE III: Averages of the last 2500 episodes given 100 episodes of demonstration data.

Algorithm	**Average Reward**	**Std. Error**	**Best Reward**
Baseline	**1129**	80	1387
Q-Network	878	357	**1758**
SARSA	776	237	1292
Dueling Q-Network	521	378	1535
Dueling SARSA	1031	162	1312

Finally, in the third plot we see that each algorithm performs differently. The Q-Network performs worst, but still performs better than when less memories are given to the replay-memory buffer. Interestingly SARSA is now able to beat the baseline algorithm and the same holds for Dueling SARSA which outperforms SARSA. It is worth noting that the two networks based on the SARSA algorithm do not only perform better but also show more stable training as can be seen by the shaded areas around the line plots representing the standard deviation over the different simulation runs. The average performances on the final 2500 episodes are also shown in Table IV.

TABLE IV: Averages of the last 2500 episodes given 1000 episodes of demonstration data. The asterisk (*) indicates the average reward is greater than the average reward of the baseline.

Algorithm	**Average Reward**	**Std. Error**	**Best Reward**
Baseline	1129	80	1387
Q-Network	907	343	1696
SARSA	1607*	108	1748
Dueling Q-Network	1369*	276	1826
Dueling SARSA	**1745***	83	**1860**

Based on these results we can observe that a key element that makes the algorithms perform well is the amount of demonstration data the algorithms are provided with at the beginning of training. The novel dueling-SARSA algorithm

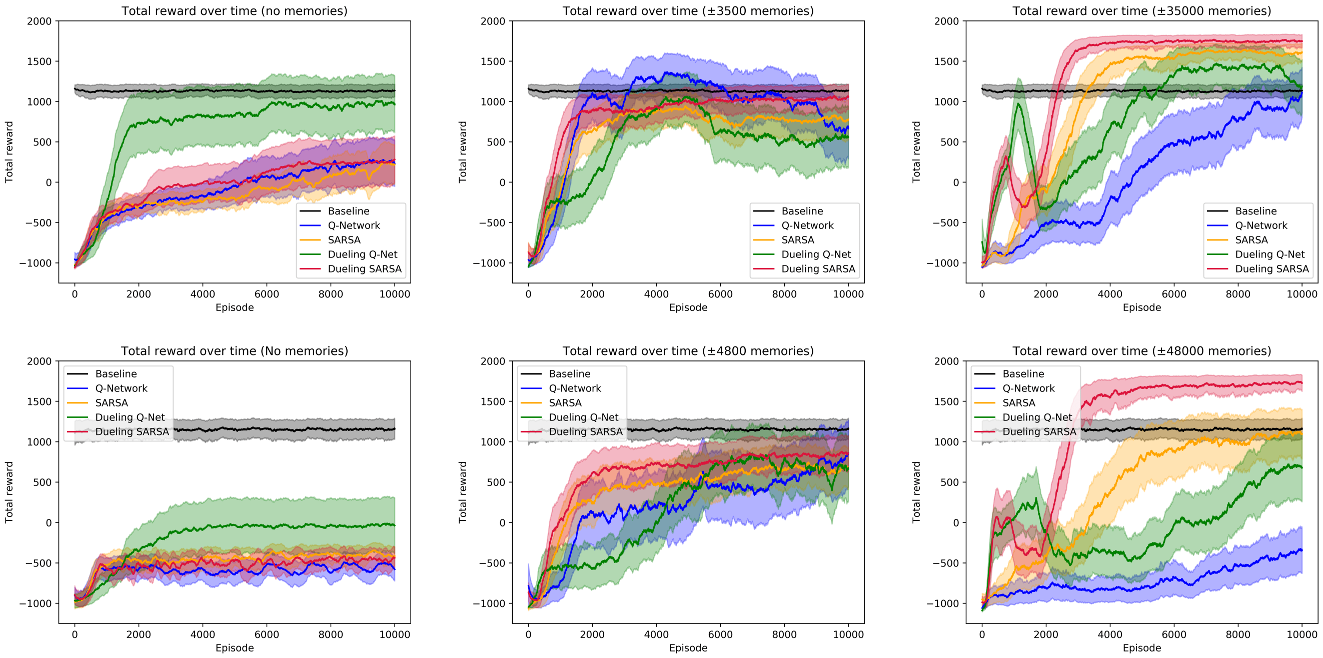

Fig. 4: The results obtained when testing different RL algorithms in the forest fire control simulator. The first row reports the results obtained on a grid of size 10×10, while the second row reports results on a larger grid of size 14×14. The first, second and third plot of each row correspond to the results obtained when initializing the experience replay memory buffer with different amounts of experiences, by using 0, 100 or 1000 episodes of demonstration data.

performs best of all RL algorithms when demonstration data is given. Its final performance when 1000 episodes of demonstration data are given, is very good with a score of 1745 on average. With this amount of demonstration data, SARSA performs second best. It is remarkable that in these experiments, the on-policy algorithms Dueling-SARSA and SARSA profit the most from the demonstration data. In most research off-policy algorithms are used when learning from demonstration is used, but our results show that this does not have to be the optimal choice.

B. *Results on* 14×14 *Environments*

We now report the results that were obtained on a larger environment of size 14×14. From the first plot of the second row of Figure 4, we can observe that the results of all RL algorithms is much worse compared to the results on the 10×10 environment. Without using demonstration data, the Dueling Q-Network again performs best. However, the results of all algorithms are much worse than those of the baseline algorithm. The average performances on the final 2500 episodes can also be found in Table V.

When looking at the second plot of Figure 4, we see that all algorithms are learning to optimize their policies, but do not outperform the baseline algorithm with the amount of training episodes they received. The average performances on the final 2500 episodes are also shown in Table VI.

In the final plot of Figure 4, we can see that only the Dueling SARSA algorithm clearly outperforms the baseline algorithm, while SARSA performs at a similar level as the baseline algorithm by the end of training. The Q-network performs very bad and is not able to obtain positive cumulative reward scores. The average performances on the final 2500 episodes are also shown in Table VII.

TABLE V: Averages of the last 2500 episodes given 0 episodes of demonstration data.

Algorithm	Average Reward	Std. Error	Best Reward
Baseline	**1152**	125	**1513**
Q-Network	-550	144	-139
SARSA	-398	116	-92
Dueling Q-Network	-40	335	349
Dueling SARSA	-455	134	-44

TABLE VI: Averages of the last 2500 episodes given 100 episodes of demonstration data.

Algorithm	Average Reward	Std. Error	Best Reward
Baseline	**1152**	125	1513
Q-Network	652	418	169
SARSA	670	257	1275
Dueling Q-Network	667	404	**1748**
Dueling SARSA	836	224	1249

As expected, all RL algorithms perform worse on the

TABLE VII: Averages of the last 2500 episodes given 1000 episodes of demonstration data.

Algorithm	**Average Reward**	**Std. Error**	**Best Reward**
Baseline	1152	125	1513
Q-Network	-459	253	411
SARSA	1057	316	1626
Dueling Q-Network	522	406	1534
Dueling SARSA	**1713***	108	**1846**

larger environment compared to the 10 × 10 scenario. The RL algorithms suffer from the curse of dimensionality, and for the larger environment the neural networks receive 588 inputs compared to 300 inputs in the smaller environment. This makes it much harder to learn an accurate Q-function with the MLPs. The baseline algorithm obtains slightly higher scores on the larger environment, which can be explained by the fact that a larger proportion of the forest area is on average protected from the fire.

With the large amount of demonstration data, Dueling-SARSA and SARSA perform the best. So these results confirm the results on the smaller environment and indicate that on-policy methods can outperform off-policy algorithms when demonstrations are given and experience replay is used. Dueling-SARSA performs very well with enough demonstration data and significantly outperforms all other algorithms. From this we can conclude that Dueling-SARSA combines the benefits of learning *on-policy*, and therefore being less prone to divergence, while it also takes advantage from estimating both the value of a state in addition to the respective Q-values as initially introduced for the Dueling-Q Networks.

VI. Conclusion

In this paper, we have studied the problem of controlling forest fires with connectionist RL. A novel forest fire simulation environment is introduced that served to study performances of four different RL algorithms. Among the tested algorithms is the novel Dueling-SARSA algorithm, which obtained the best results. Furthermore, we noticed that on-policy algorithms such as SARSA and Dueling-SARSA performed better than off-policy RL algorithms when enough demonstrations were given to the algorithms. This is in contrast to the fact that most researchers believe that off-policy algorithms are better able to learn from demonstrations and experience replay.

In future work, we want to examine the efficiency of on-policy algorithms when combined with learning from demonstration on other problems. Furthermore, we would like to examine if the Dueling-SARSA algorithm also performs better than other RL algorithms on different problems, such as in the `Atari Arcade Learning` environment. We believe that Dueling-SARSA is also likely to benefit from all the improvements [12, 17, 18], which have been proposed over the years and made Deep Reinforcement Learning (DRL) very successful.

We also want to make our forest fire simulator more complex and allow multiple reinforcement learning agents to learn cooperative forest fire control strategies. Furthermore, we want to study the effectiveness of convolutional neural networks (CNNs) to learn to approximate the value functions, instead of the multilayer perceptrons used in this paper. Because the environmental state can be represented with an image, CNNs could be much more effective for handling very large environments. Finally, the RL algorithms could profit from several extensions that have made DRL more sample efficient and robust to sparse rewards problems [19, 20, 21, 22]. Such extensions will hopefully lead to effective forest fire control policies for very complex scenarios in the future.

References

[1] E. Kasischke, N. Christensen Jr, and B. Stocks, “Fire, global warming, and the carbon balance of boreal forests,” *Ecological applications*, vol. 5, pp. 437–451, 1995.

[2]

[3] M. Wiering and M. Doringo, “Learning to control forest fires,” in *Proceedings of the 12th international Symposium on ’Computer Science for Environmental Protection’*, H. Haasis and K. Ranze, Eds., 1998, pp. 378–388.

[4] M. Wiering, F. Mignogna, and B. Maassen, “Evolving neural networks for forest fire control,” in *Benelearn ’05: Proceedings of the 14th Belgian-Dutch Conference on Machine Learning*, M. van Otterlo, M. Poel, and A. Nijholt, Eds., 2005, pp. 113–120.

[5] D. Moura and E. Oliveira, “Fighting fire with agents: an agent coordination model for simulated firefighting,” in *Proceedings of the 2007 spring simulation multiconference-Volume 2*. Society for Computer Simulation International, 2007, pp. 71–78.

[6] C. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, University of Cambridge, 1989.

[7] G. Rummery and M. Niranjan, “On-line Q-learning using connectionist systems,” University of Cambridge, Department of Engineering Cambridge, England, Tech. Rep., 1994.

[8] Z. Wang, N. de Freitas, and M. Lanctot, “Dueling network architectures for deep reinforcement learning,” *CoRR*, vol. abs/1511.06581, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[9] L.-J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992.

[10] R. Sutton and A. Barto, *Reinforcement Learning: an Introduction*, 2nd ed. The MIT Press, 2018.

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fid-

jeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[13] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[14] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[15] J. Tsitsiklis and B. Van Roy, “Analysis of temporal-diffference learning with function approximation,” in *Advances in neural information processing systems*, 1997, pp. 1075–1081.

[16] S. Knegt, M. Drugan, and M. Wiering, “Opponent modelling in the game of Tron using reinforcement learning,” in *ICAART 2018 - Proceedings of the 10th International Conference on Agents and Artificial Intelligence*, vol. 2, 2018, pp. 29–40.

[17] M. Sabatelli, G. Louppe, P. Geurts, and M. Wiering, “Deep Quality-Value (DQV) Learning,” *arXiv preprint arXiv:1810.00368*, 2018.

[18] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” *CoRR*, vol. abs/1710.02298, 2017. [Online]. Available: http://arxiv.org/abs/1710.02298

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[20] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, “Noisy networks for exploration,” *arXiv preprint arXiv:1706.10295*, 2017.

[21] H. Zou, T. Ren, D. Yan, H. Su, and J. Zhu, “Reward shaping via meta-learning,” *arXiv preprint arXiv:1901.09330*, 2019.

[22] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.