Weaknesses and fixes of the Fair-Efficient Network

Chloé Terwagne[1], Chris Adam[1], Delanoe Pirard[1] and Paul Parent[1]

[1] Université Libre de Bruxelles

Abstract

Fairness is a key feature that can improve social health while ensuring productivity in multi-agent systems. In addition, fairness guarantees that every agent participates to the system. Here, we review the Fair-Efficient Network (FEN) first described by Jiang and Lu (2019). The prominent components are a fair-efficient reward that compromises fairness and efficiency, a hierarchy that controls which aspect should be optimised, and a gossip algorithm allowing the learning to be decentralised. This structure provides fairness among agents while maintaining efficiency. We present a partial reproduction of their results in addition of a few possible weaknesses in this algorithm.

Introduction

Jiang and Lu (2019) proposes a new reinforcement learning (RL) algorithm where agents learn both fairness and efficiency. Fairness is essential in some systems such as traffic control lights Bakker et al. (2010), where resources have to be fairly allocated between agents. As we can understand from these real-life examples, fairness actually helps the system to become more efficient (traffic will be more fluid if traffic control lights work together instead of against each other). However, learning fairness and efficiency in a multi-agent system is a complex problem that has never been really solved before. Few methods consider fairness. Nonetheless, they are built for specific applications with specific domain knowledge and cannot be generalised Li et al. (2019). The model proposed by Jiang and Lu (2019), the Fair-efficient Network, has three attributes that will help address this problem. First, a fair-efficient reward that compromises between fairness and efficiency. Fairness is guaranteed if all agents succeed to maximise it. Then, a hierarchy helps achieve this fair-efficient reward optimisation by selecting different sub-policies concerned either by efficiency or fairness. Finally, the system learns in a fully decentralised way. Indeed, agents have access to local information only.

In this work, we will first introduce the FEN and its components. Thereafter, some parameters will be changed to challenge the model in several multi-agent scenarii. For instance, we bring modifications to Proximal Policy Optimisation, the RL algorithm used in the model. In addition, we run the model with and without hierarchy. Ultimately, we bring changes to the gossip algorithm which permits the decentralised learning.

Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is one of the best reinforcement learning algorithm designed by OpenAI (Schulman et al. (2017a)). However, its training relies on a dynamic dataset that depends on the current environment and policy. It means that the policy needs to be updated very carefully in order to keep a relevant dataset. Indeed, PPO is a policy gradient method that learns online. Therefore, it uses only the last batch of experiences to update its policy and updates quicker and every experience is analysed once (Schulman et al. (2017b)).

PPO estimates an advantage function ($\hat{A}_t$) to further compute the gradient estimator and updates its neural network. It represents how good the algorithm performs compared to its expectation. This expectation is measured by another neural network which gives a noisy estimate of the expected reward (Insights (2018)). The comparison between the real reward and this estimate gives the advantage function. Afterwards, the probability to play specific actions combined to this advantage function are used to update the neural network by stochastic gradient ascent.

The major problem of policy gradient methods is that they risk updating their policy too far away from their current policy and change their learning dataset in a dramatic way. Several algorithms were introduced in order to solve this problem such as Trust Region Methods (TRPO) (Schulman et al. (2015)). TRPO modifies its objective function to include a KL constraint that prevents big policy updates. However, implementing it can be tricky.

PPO brings innovation by its simplicity in addition of better efficiency. First, define $r_t(\theta)$, the probability ratio between the new updated policy outputs and the outputs of the previous old policy network (equation 1). In other words, $r_t(\theta)$ describes how likely an action is to be played compared to the previous policy. Now, multiplying this ratio by

the advantage function gives the TRPO objective in a simpler form (equation 2).

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{1}$$

$$L^{CPI}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \hat{E}_t\left[r_t(\theta)\hat{A}_t\right] \tag{2}$$

Thereafter, the objective function of PPO is presented by equation 3. Notice it's a pessimistic estimate by taking the minimum of two terms. The first term is the normal TRPO objective while the second second one is clipped. This clip operation plays a different role whether the advantage function is positive or negative. In the case where $\hat{A}_t$ is positive, the objective function is truncated for large values of r and prevents large updates of the policy. In contrast, if $\hat{A}_t$ is negative, the min operator comes into effect for small values of r. In a nutshell, PPO works based on the same principle as TRPO but uses a simpler objective function for an easier implementation and shows even better results in most cases (Schulman et al. (2017b)). Finally, an entropy terms is added to promote exploration.

$$L^{CPI}(\theta) = \hat{E}_t\left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{3}$$

Hierarchy

RL vs. Hierarchical Reinforcement Learning (HRL) Hierarchy was first used few decades ago in order to reduce the computational cost of planning (Currie and Tate (1991)). The same idea of hierarchy was implemented in RL. HRL breaks down specific parts of the learning process in order to alleviate this complexity. More specifically, the HRL learns to operate on different levels of temporal abstraction. For example, when following a recipe, a temporal abstraction is adopted. The recipe says "Cut four carrots" instead of "Take one carrot, take your knife, put the carrot on the table, cut the first slice, etc.". Our brain already knows sub-steps and does not require the recipe to indicate all of them. HRL works in a similar way. It is designed with several sub-policies instead of a single policy. These sub-policies work together in a hierarchical structure to achieve the goal set by the top level of the hierarchy. Furthermore, HRL provides several advantages compared to RL:

- improved exploration: the action space is reduced thanks to the high-level actions ;
- sample efficiency: the state space is reduced by allowing low-level policies to hide irrelevant information to the high-level policies ;
- transfer learning: an agent can re-use learnt sub-policies when executing another task.

For example, if the agent has already learnt how to "Cut four carrots", this knowledge can be re-used in a recipe "Make carrots soup" where substeps would be "Take four carrots, Cut four carrots, Mix the carrots".

Types of HRL Hereafter is presented two major founding HRL algorithms. The first one and the most common algorithm is Options Framework (Sutton et al. (1999). This method use Semi-Markov Decision Process. Such process allows the system to have different temporal resolution because the time between each timestep is not constant. Another well-known HRL algorithm is Feudal Learning (Dayan and Hinton (1993)) which presents two advantages. First, reward hiding: managers reward sub-managers for satisfying their commands (the reward does not come from the environment). Indeed, managers have absolute control. Second, information hiding: managers must not know what takes place in other sub-managers. This algorithm is limited in its possible applications but it became the base of more recent papers, such as FeUdal Networks (FuNs) (Sasha Vezhnevets et al. (2017)).

Distributed average consensus

The distributed average consensus Xiao et al. (2007) is a classic problem for distributed computing. It is widely studied in lots of application which use mobile autonomous agents such as our article here or such things as load balancing in parallel computers. The great principle of this problem is get the average of every values of the network in every node, allowing only local communication on the graph or, in our case, in the observation zone of the agents.

A formal definition of this is the following: Getting $\gamma = (\nu, \varepsilon)$ as an undirected connected graph with a set of node $\nu = 1, ..., n$ and its linked set of edge $i, j \in \varepsilon$ which is also unordered. Each node i has a real scalar $x_i(0)$ as original value at $t = 0$. In the purpose of computing the gossip algorithm, each node will cause its update based on its local value at each $t = 1, 2, ...$ and communication with its neighbours $\nu_i = \{j|\{i, j\} \in \varepsilon\}$

There is a well-known linear iterative algorithm that can handle that problem which will be describe below:

$$x_i(t+1) = x_i(t) + \sum_{j\in\nu_i} W_{ij}(x_j(t) - x_i(t)) \tag{4}$$

or

$$x_i(t+1) = W_{ii}x_i(t) + \sum_{j\neq i\in\nu_i} W_{ij}x_j(t) \tag{5}$$

for $i = 1, ..., n$, and $t = 0, 1, ...$ W_{ij} is a weight associated with the edge $\{i, j\}$.

The demonstration below comes from Xiao et al. (2007) and Lin Xiao and Boyd (2003) but for the sake of having a

better understanding how the gossip algorithm works, it is interesting to show it in this paper.

To have a functional algorithm, the weights must respect several conditions which will be exposed below. Furthermore, it needs to achieve an asymptotic average consensus with all the possible initial values. The starting points are as follows: it needs $W_{ij} = W_{ji}$ because the edges are undirected, $W_{ij} = 0$ for $j \notin \nu_i$ and $W_{ii} = 1 - \sum_{j \in \nu_i} W_{ij}$. Thus, it is possible to create a new square matrix W which satisfies by construction:

$$W = W^T, \qquad W\vec{1} = \vec{1}, \qquad W \in S \tag{6}$$

S is the set of sparsity matrices that is consistent with the graph γ:

$$S = \{W \in R^{n \times n} | W_{ij} = 0 \textbf{ if } i \neq j \textbf{ and } \{i,j\} \notin \varepsilon\}.$$

Then, it become possible to vectorize the equation above (4) such as the equation of linear regression:

$$x(t+1) = Wx(t), \qquad t = 0, 1, ...$$

with initial condition $x(0) = (x_1(0), ..., x_n(0))$.

Having a matrix W that satisfies the first conditions (6) is mandatory to have a functional algorithm. However, these conditions are necessary but not sufficient for the algorithm to converge asymptotically to the true average. In the purpose of perform this, the previous constraints will be added to the next one which will be explain below. First of all, it is necessary to have the following equation true:

$$\lim_{t \to \infty} x(t) = \lim_{t \to \infty} W^t x(0) = (1/n)\vec{1}\vec{1}^T x(0)$$

for all $x(0)$. The matrix $(1/n)\vec{1}\vec{1}^T$ is called the averaging matrix and will be called further J. So in our case if t goes to infinity, the previous equation will be equal to $(1/n)\vec{1}\vec{1}^T x(0)$ and this vector has all his components which are the average of every node of x.

Now, there is the new condition to have asymptotic average consensus in addition to the others (6):

$$\|W - J\| < 1 \tag{7}$$

That norm above is the asymptotic rate of convergence to consensus. So the algorithm need that the distance between the practical weights and the averaging matrix J to be less than one. Thus, more this distance get close to 0, more the worst case is low and we will achieve a faster averaging. Then, at each step the distance between the new node values and the true average will be smaller because:

$$\|x(t+1) - Jx(0)\| \leq \|W - J\| \|x(t) - Jx(0)\|.$$

There are weights that satisfy the constraints above 6 and 7. We can find them in several papers (e.g. Lin Xiao and Boyd (2003), Denantes et al. (2008) or Jalili (2012)). Some are easy to calculate but have a larger worst case, etc. However, in that article, the *Metropolis-Hastings* weights will be used.

$$W_{ij} = \begin{cases} 1/(max\{d_i, d_j\} + 1) & \{i,j\} \in \varepsilon \\ 1 - \sum_{j \in \nu_i} 1/(max\{d_i, d_j\} + 1) & i = j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $d_i = |\nu_i|$ is node degree.

There are some variation of the algorithm 4 like *distributed average consensus with additive noise* Xiao et al. (2007).

Methods

Fair-efficient reward

Jiang and Lu (2019) defines the utility of agent *i* at timestep *t* as shown in the equation 9. *r* represents the environmental reward that an agent obtains when occupying resources. Indeed, *u* is the average reward over elapsed timesteps.

$$u_t^i = \frac{1}{t} \sum_{j=0}^{t} r_j^i \tag{9}$$

It is possible to measure the fairness (Jain et al. (1984)) using the coefficient of variation (CV) of agents' utilities, equation 10. $\bar{u}$ is the average utility of all agents. *n* is the number of agents in the system. A system is considered to be fairer than another when its CV is smaller.

$$CV = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \frac{(u^i - \bar{u})^2}{\bar{u}^2}} \tag{10}$$

Equation 11 represents the fair-efficient reward. In fact, the fairness objective of the system is decomposed for each agent (*r*) and each agent will try to maximise it. The numerator is linked to the efficiency of the agent. The denominator compute the deviation between the agent's utility from the average with the goal of penalising unfair behaviours.

Moreover, $\bar{u}$ which is included in the equation of the fair-efficient reward can actually coordinate agents' policies in a decentralised way.

$$\hat{r}_t^i = \frac{\bar{u_t}/c}{\epsilon + |u_t^i/\bar{u_t} - 1|} \tag{11}$$

Hierarchy

The hierarchy in FEN is different from the hierarchies previously cited in the introduction: Option Framework, Feudal Learning, and FeUdal Networks. Indeed, they cannot take into account both fairness and efficiency. To overcome this

problem, authors of FEN implemented a new system where a controller learns to select among several sub-policies. In order to maximise the fair-efficient reward, the controller will choose one among the different sub-policies which will then interact with the environment. However, the specificity of FEN is that one of the sub-policy is designated to maximise the environmental reward (in order to obtain efficiency) and other sub-policies explore diverse possible behaviours for fairness.

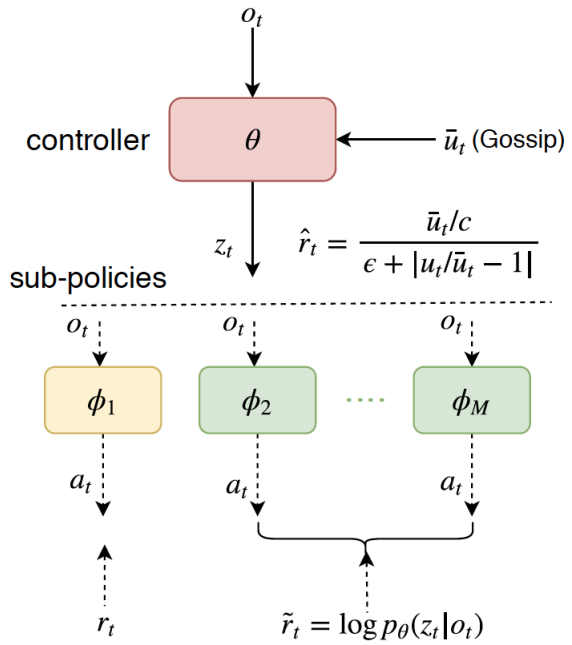

Figure 1: FEN architecture

In figure 1 proposed by Jiang and Lu (2019), we can understand the architecture of FEN hierarchy. It is composed of two layers, the top layer with the controller Θ and the bottom layer with the sub-policies ($\Phi_1, \Phi_2 ... \Phi_M$). Based on the observation o_t, the controller will choose among the sub-policies and will receive the fair efficient reward. The sub-policy will then take the action a_t in the environment. Φ_1 tries to maximise efficiency (the reward is in this case r_t) and $\Phi_2 ... \Phi_M$ explore diverse behaviours for fairness (the reward is in this case $\log p_\Theta(z_t|o_t)$). Nevertheless, the algorithm proposed by Jiang and Lu (2019) in their GitHub repository only use the probability instead of the log probability as we did.

FEN without hierarchy To highlight the importance of hierarchy, FEN algorithm is run with and without hierarchy in the job scheduling problem. Based on the code available on the GitHub repository: `https://github.com/PKU-AI-Edge/FEN/blob/master/job.py`, modifications were made in order to train only one policy. The training of the policy maximises the fair-efficient reward directly, equation 11.

Decentralised learning

In order to update the $\bar{u}$ every timestep, Jiang and Lu (2019) use the *distributed average consensus*, also know as *gossip algorithm*, as we can see in the equation 12.

$$\bar{u}_i(t+1) = \bar{u}_i(t) + \sum_{j \in N_i} w_{ij} \times (\bar{u}_j(t) - \bar{u}_i(t)) \quad (12)$$

Centralised learning requires more computational power as the number of agents increase (curse of dimensionality). This limitation is avoided using decentralised learning.

Jiang and Lu (2019) propose 2 ways to implement this algorithm. One presented in the paper and one presented in the code. The first one is located in the sub-policies update section each T steps (see algorithm 1) and the second one is performed each step. Not knowing which of these two were actually used, both were tested.

Algorithm 1 FEN training from Jiang and Lu (2019)

Initialise u_i, $\bar{u}_i$, the sub-policies ϕ and the controller θ
for episode = 1,...,M **do**
  for t=1,...,max-episode-length **do**
    The chosen policy ϕ_z acts to the environment
    and gets the reward $\begin{cases} r_t, & \text{if } z = 1 \\ \log p_\theta(z|o_t), & \text{else} \end{cases}$
    if $t\%T = 0$ **then**
      Update the policy ϕ_z using PPO
      Update $\bar{u}_i$ (eventually using Gossip)
      Calculate $\hat{r}^i = \frac{\bar{u}_t/c}{\epsilon + |u_t^i/\bar{u}_t - 1|}$
      The controller selects one of the sub-policies
    end if
  end for
  Update the controller θ using PPO
end for

Results

For the sake of reproducibility, the entire code is available in this GitHub repository: `https://github.com/Aedelon/fair-efficient_network`

Job scheduling

The job scheduling problem consists of 1 resource for 4 agents in a 5x5 grid world, as illustrated in figure 2. At the beginning of each episode, the four agents and the resource are randomly placed. Nevertheless, the resource can't be placed near the edge. Each agent has 5 available actions: move in one of the four directions or stay still. Each agent has an observation grid of size 3x3 centred on itself. Two agents can't be on the same grid ensuring the resource can't be occupied by more than one agent at each step. When

the agent is on the same grid as the resource (stay or move to the resource's location),it gets a reward of 1. When the agent doesn't occupy the resource, it gets no reward (reward = 0). FEN is used to investigate whether the agents can learn to share the resource in an efficient way. In this case, the controller and the sub-policies are shared for the learning.

	A			
		A		A
			R	
A				

Figure 2: Illustration of the job scheduling scenario. Where A represents an agent and R represents the resource.

Grid size variation for FEN and FEN without hierarchy The job scheduling scenario is run 5 times for each following grid size: 5x5, 9x9 and 25x25. All the other parameters are left to default (table 2). Figure 3 shows the learning curves for three different environment sizes. While figure 4 shows the learning curves for three different environment sizes of the FEN and FEN without hierarchy algorithms. The mean of the fair-efficient reward (equation 11) is the fair-efficient reward averaged over four agents and then averaged over the five simulations.

In figure 3, the learning curve of the smallest grid world has the larger mean the fair-efficient reward during the 800 first episodes with a maximum reward of 4. However, the 5x5 and 9x9 curves have relatively similar mean fair-efficient reward for the 200 latest episodes. Although the 5x5 and 9x9 learning curves are not monotonically increasing, the mean fair-efficient reward tends to grow over time (number of episodes). The 25x25 grid world has, for a majority of episodes, a mean fair-efficient reward of zero. Most of the positive mean fair-efficient rewards appear at the end of the simulation, as shown in figure 3.

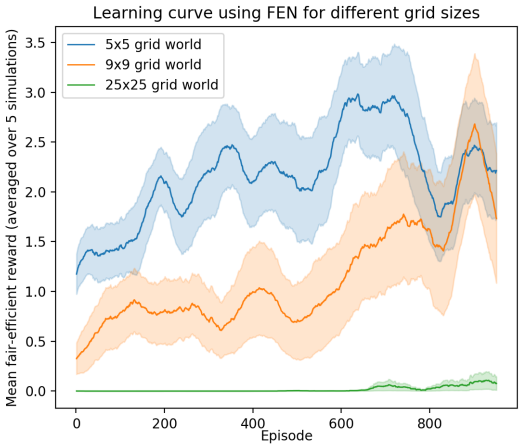

Figure 3: Averaged learning curves over 5 simulations for each grid size. The simulations were run with the parameters summarised in table 2, except the grid size parameter which is 5x5, 9x9 and 25x25 grid world for the blue, orange and green curves respectively. Data has been smoothed by a 50-episodes sliding window. The shaded areas correspond to the standard error of each curve.

Figure 4 shows that FEN without hierarchy has a mean fair-efficient reward lower than FEN with hierarchy regardless the size of the environment. Moreover, for the three subplot, the curve of FEN without hierarchy does not show a clear growing trend as it's the case for the curve of FEN with hierarchy.

Gossip algorithm To test the efficiency of the gossip algorithm, several tests were carried out on the job scheduling game. A comparison of the learning curves of the mean fair-efficient reward between the centralised, decentralised at each step and decentralised at each T steps - as it is in the original paper of Jiang and Lu (2019)) is made. The results presented in this section are obtained with the default parameters (table 2). In addition, some comparisons of the calculated mean rewards of agents over one episode in each of the three decentralisation algorithms.

To begin, the figure 5 shows the 3 curves stated above. The first one, in blue, is the curve that represents the centralised learning (e.g. every agent get the average of every $\bar{u}$ of the agents) for this game. Surprisingly, this one has fair-efficient reward scores smaller than the two other decentralised ones. Otherwise, the latter have more or less the same results over 1000 episodes.

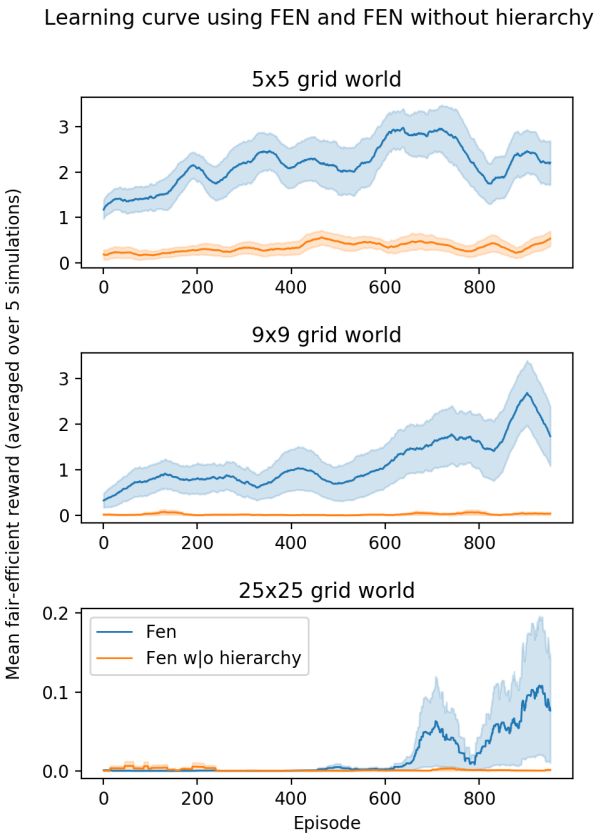

Figure 4: Averaged learning curves over 5 simulations for the FEN algorithm with and without hierarchy for 3 different grid sizes. The simulations were run with the parameters summarised in table 2, except the grid size parameter which is 5x5 *(top)*, 9x9 *(middle)* and 25x25 grid world *(bottom)*. Data has been smoothed by a 50-episodes sliding window. The shaded areas correspond to the standard error of each curve.

Several plots show the comparison of the calculated average of the agents reward ($\bar{u}$) at every steps for every cases over one episode. The first plot 6 shows the results for the first episode. This graph shows that the first steps are a bit chaotic for each case. The decentralised learning all the T steps shows a much larger standard error than for the other cases. However, all three eventually stabilise around 0.04 with a standard error that gradually shrinks.

In the second figure (7), what is important to observe is that the "centralised" curve begins to stabilise by slightly dominating the other two but not at the optimum (in our scenario here - $c = 0.25$).

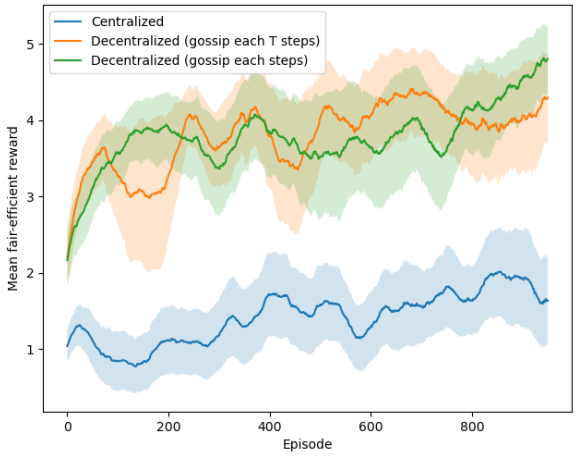

Figure 5: 3 averaged curves of the mean fair-efficient rewards over 5 simulations for job scheduling scenario. The blue curve represents the centralised learning. The orange curve stands for the gossip algorithm performed at each T steps. The green curve is the centralised learning occurring at each step. The data are smoothed like the other graphs

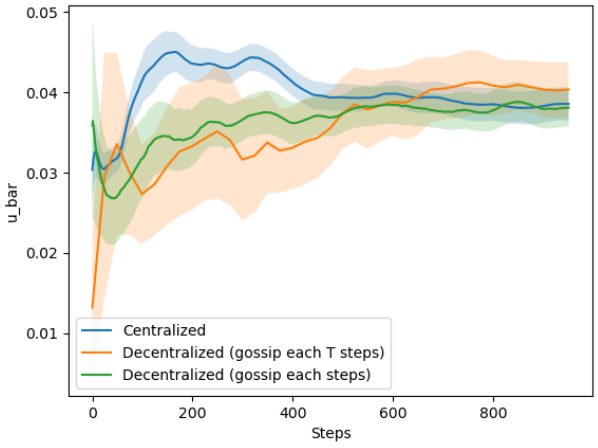

Figure 6: Averaged curves the $\bar{u}$ for episode 1 over 5 simulations for job scheduling scenario. The blue curve represents the centralised learning. The orange curve stands for the gossip algorithm performed each T steps. The green curve is the centralised learning occurring at each step. The data are smoothed like the other graphs

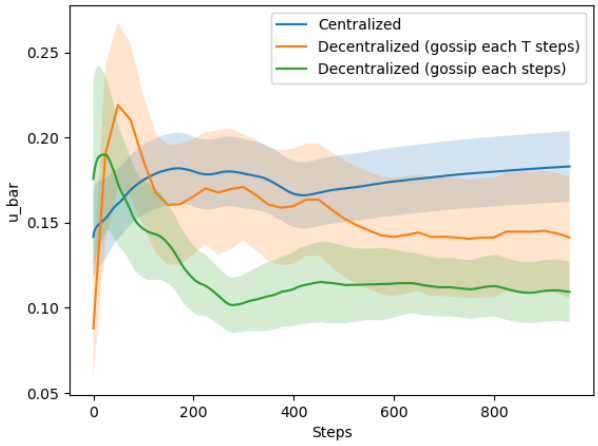

Figure 7: Averaged curves the $\bar{u}$ for episode 500 over 5 simulations for job scheduling scenario. The blue curve represents the centralised learning. The orange curve stands for the gossip algorithm performed each T steps. The green curve is the centralised learning occurring at each step. The data are smoothed like the other graphs

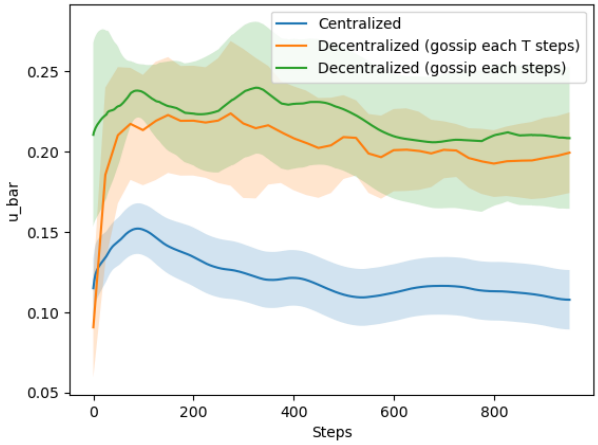

Figure 8: Averaged curves the $\bar{u}$ for episode 999 over 5 simulations for job scheduling scenario. The blue curve represents the centralised learning. The orange curve stands for the gossip algorithm performed each T steps. The green curve is the centralised learning occurring at each step. The data are smoothed like the other graphs

The last figure (8) shows a correction compared to the previous one. We can see that the blue curve is still stabilised albeit slightly declining. The other two curves are on the way to reaching the optimum. These are still as slow at startup as in other episodes, but end up doing better. We see here that there is a link between the computation of the mean of $\bar{u}$ and the mean fair-efficient rewards in figure 5. The reader may also notice that the standard error of the green curve has become larger than the standard error of the orange curve unlike the other episodes.

Matthew effect

In a system, the Matthew effect is defined as the increasing wealth of the richest at the expense of the poorest. The situation presented here contains 10 Pacman players looking to eat phantoms. The three static phantoms are randomly placed in a square map. Every time one is eaten, another one pops somewhere else. When a Pacman gets food, it gets a reward of 1, grows in size and increases its speed unless the development upper bound has been reached. Therefore, bigger Pacmen will be able to reach food faster than smaller Pacmen.

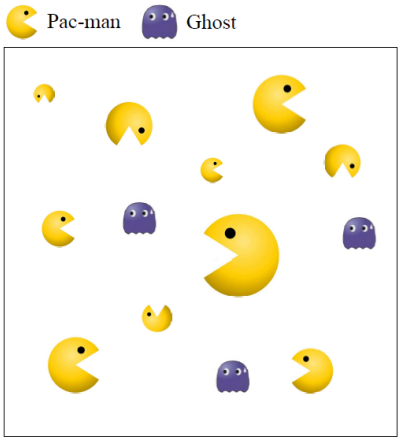

Figure 9: Matthew effect scenario (Jiang and Lu (2019))

PPO epsilon hyperparameter and learning stability Jiang and Lu (2019) already demonstrated that fairness is achieved. This result is confirmed in our experiments (table 1). However, further investigations shows that the learning process tends to worsen after a few episodes. To overcome this problem, the policy update dynamic has been changed in order to enhance the learning process. It turned out that reducing the epsilon hyperparameter of PPO improved the learning process (figure 10). In this case, the controller and the sub-policies are also shared for the learning.

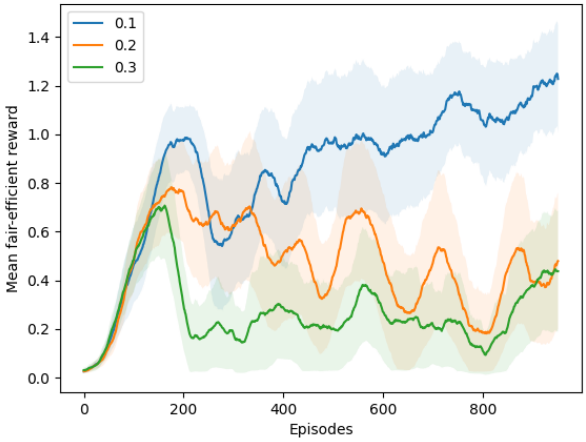

Figure 10: Learning curves of the Matthew effect scenario averaged over 5 runs. Data has been smoothed by a 50-episodes sliding window. The epsilon hyperparameter of PPO is varying from 0.1 to 0.3 by step of 0.1. The other parameters were left to default (table 3). The standard errors are also displayed on the plot.

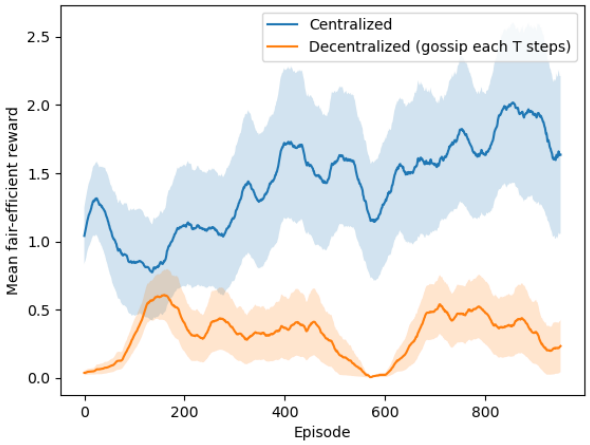

Figure 11: Averaged curves of the mean fair-efficient rewards over 5 simulations for Matthew's effect scenario. The blue curve represents the centralised learning. The orange curve stands for the gossip algorithm performed each T steps. The data are smoothed by 50-episode sliding window and the standard errors are also displayed on the plot.

ϵ	Average reward / step	Std
0.1	$0.3926 \pm 0.9\%$	$0.0041 \pm 1.1\%$
0.2	$0.2846 \pm 1.2\%$	$0.0020 \pm 1.2\%$
0.3	$0.22 \pm 1.2\%$	$0.0033 \pm 1.9\%$

Table 1: In the Matthew effect scenario: average total reward per step and standard deviation of agents' rewards per step. Only the last episode was considered during which the hierarchy and agents are fully trained.

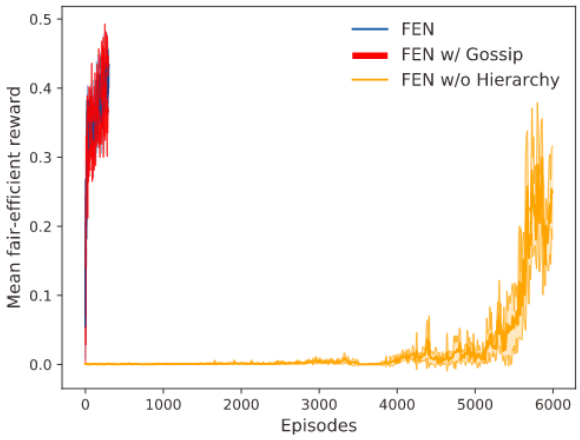

Figure 12: Learning curves over 5 simulations of FEN, FEN with gossip and FEN without Hierarchy from the original paper Jiang and Lu (2019)

Gossip algorithm To test the efficiency of gossip in this game, a simple comparison between the centralised algorithm and the gossip algorithm (at all T steps) was performed (with $\epsilon = 0.1$ in the PPO algorithms). We can see in the following graph (figure 11) that, here, unlike job scheduling, the centralised algorithm obtains much better results even if the curves follow the same trends. Here, the code from the GitHub repository provided by Jiang and Lu (2019) has been modified to correspond with the gossip algorithm indicated in the paper Jiang and Lu (2019). It is seen here that the results are different from those displayed in the original paper (figure 12).

Manufacturing plant

This last scenario takes place in a 8x8 grid where 8 gems of 3 different types are placed randomly. 5 agents are looking for 1 gem of each different types. Every time an agent grabs a gem, it gets a reward of 0.01 and an new gem appears randomly. Once an agent has a specific set of gems, it crafts a product and get a reward of 1. The observation grid of players is a 5x5 grid centred on itself. However, there not an unlimited number of gem thus the game will ends when

there is no remaining gem. In this case, the controller and the sub-policies are not shared for the learning.

Neural network layer size for the sub-policies Just like the Matthew effect scenario, the dynamic of the learning rate is influenced by the hyperparameters of the PPO algorithm. In this case, the layer size of the neural network of the sub-policies seems to improve the results (figure 13). In the early phase of the learning process, the fair-efficient reward drops. However, increasing the number of nodes in the hidden layer of the neural network of PPO shows reduces this early drop and improves the following episodes.

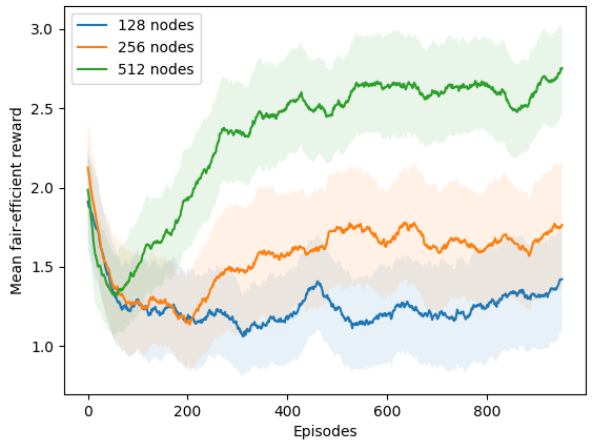

Figure 13: Learning curves of the manufacturing plant scenario averaged over 5 runs. Data has been smoothed by a 50-episodes sliding window. The PPO hidden layer size of the sub-policies is varying. The other parameters were left to default (table 4). The standard errors are also displayed on the plot.

Discussion

Computational power Despite having a lot of different data, the length and amount of simulations has reached its limits considering our working conditions. Indeed, one simulation takes about 5 hours using a supercomputer. Therefore, we couldn't test the learning process of the network for more than 1000 episodes. Even though the original authors of FEN barely tested the algorithm over 1000 episodes, the default parameter found in their GitHub repository was 100000 episodes. To realise, it would take over 20 days to run a single simulation with this amount of episodes. Moreover, five simulations per experiment can be seen as a minimum. Thus, increasing the size of our dataset is to be considered.

Hierarchy The hierarchy clearly improves the performance of FEN over 1000 timesteps for the job scheduling scenario regardless of the 3 different environment sizes.

Grid size Although the learning is slower for the 9x9 grid than for the 5x5 grid, we can see that after 1000 episodes the two learning curves converge towards the same value in the job scheduling scenario. However, when the algorithm is run on a larger scale, such as 25x25, the learning process is slower. Indeed, for the same number of agents and one resource, the environment to explore is way larger (625 squares). Moreover, the observation of each agent is still 3x3 which becomes very small relative to the environment. Better performance for a 25x25 grid world would require either a larger number of episodes (e.g. 100000 instead of 1000) or a larger observation area for each agent.

Hyperparameters Furthermore, the learning dynamic has been shown to be able to be improved for the Matthew effect and manufacturing plant scenarii by tuning PPO hyperparameters. However, it should be noticed that default parameters work the best in most cases and adjusting parameters is a careful operation which is specific for each scenario. A note can be added, test the scenarii with unique controller and sub-policies for each agent in job scheduling or Matthew's effect could be interesting for future works.

Gossip algorithm Surprisingly, for job scheduling, tests using the gossip algorithm work better than with a centralised server. One possible explanation is that the different $\bar{u}$ averages for each agent will generate a different and more optimal update than if the average is calculated centrally. We also see that the centralised version of the algorithm tends to stabilise faster. Perhaps because the averages are similar, the Reinforcement Learning algorithm does not explore as much as the others and then tends to exploit its learning faster. It could be interesting to try different weights formulas for distributed average consensus.

In the other hands, for the gossip algorithm in the Matthew's effect is not as efficient as the job scheduling scenario. It can be also because they don't do the observation of the others Pac-Men in the source code of the original paper but they chose random Pac-Men to perform that with a gossip algorithm way different that they explain in the paper. In this work, we use the observation only with the gossip algorithm because it cannot be done without. However, it is now known that the hyperparameters have more impact on the Matthew's effect than on job scheduling. Perhaps, it is worth to investigate others hyperparameters with gossip algorithm to see what is working better.

Code and paper review The paper that has been worked on is subject to some comments. Indeed, Jiang and Lu (2019) invites us to see their GitHub and read their codes. Unfortunately, there are a lot of differences between what is explained in the paper and what is found in their implementation. There are several examples. First, the sizes of the lay-

ers of the PPO controller are not the same as those indicated in the paper (256 vs. 128). Secondly, the implementation of the gossip algorithm provided in the GitHub of the paper (Jiang and Lu (2019)) differs from the explanation in the paper on several points. Third, c is equal to 0.25, 0.15, 0.003 in job scheduling, Matthew effect and manufacturing plant scenarii respectively. in their code even though it is defined as the maximum environmental reward the agent obtains during a timestep. Fourth, for the manufacturing plant environment, inconsistencies are present in the original GitHub code (Jiang and Lu (2019)) such as:

- the reward bonus for collecting a gem is defined as 0.1 as opposed to 0.01 ;
- T=500 as opposed to 50 ;
- the authors do not take the three nearest neighbours but three random neighbours.

This is a little disappointing as it was quite hard to know if they had used the algorithms found in GitHub or others that are not part of their repository. The information is somewhat contradictory. It was kind of them to provide the code but it led to a lot of uncertainty eventually. On the other hand, the paper is relatively well written and is generally quite clear. It should also be noted that the paper has never been peer reviewed before the authors of this article.

Acknowledgements

Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region

References

Bakker, B., Whiteson, S., Kester, L., and Groen, F. C. (2010). Traffic light control by multiagent reinforcement learning systems. In *Interactive Collaborative Information Systems*, pages 475–510. Springer.

Currie, K. and Tate, A. (1991). O-plan: the open planning architecture. *Artificial intelligence*, 52(1):49–86.

Dayan, P. and Hinton, G. (1993). Feudal reinforcement learning. *URL: http://www. cs.toronto. edu/˜ fritz/absps/dh93. pdf*.

Denantes, P., Benezit, F., Thiran, P., and Vetterli, M. (2008). Which distributed averaging algorithm should i choose for my sensor network? In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 986–994.

Insights, A. (2018). Policy gradient methods and proximal policy optimization (ppo). `https://www.youtube.com/watch?v=5P7I-xPq8u8`.

Jain, R. K., Chiu, D.-M. W., Hawe, W. R., et al. (1984). A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*.

Jalili, M. (2012). A simple consensus algorithm for distributed averaging in random geographical networks. *Pramana*, 79.

Jiang, J. and Lu, Z. (2019). Learning fairness in multi-agent systems. In *NeurIPS*.

Li, X., Zhang, J., Bian, J., Tong, Y., and Liu, T.-Y. (2019). A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network. *arXiv preprint arXiv:1903.00714*.

Lin Xiao and Boyd, S. (2003). Fast linear iterations for distributed averaging. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, volume 5, pages 4997–5002 Vol.5.

Sasha Vezhnevets, A., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *arXiv*, pages arXiv–1703.

Schulman, J., Klimov, O., Wolski, F., Dhariwal, P., and Radford, A. (2017a). Proximal policy optimization. `https://openai.com/blog/openai-baselines-ppo/`. Accessed: 2020-03-01.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Xiao, L., Boyd, S., and Kim, S.-J. (2007). Distributed average consensus with least-mean-square deviation. *Journal of Parallel and Distributed Computing*, 67(1):33 – 46.

Appendix

Parameter	value
Grid world	5x5
# agents	4
# resource	1
# episodes	1000
# steps	1000
T (POO batch size)	25
# s-p	4
Learning rate for value n.	10^{-3}
Learning rate for PPO n.	$3 \cdot 10^{-3}$
ϵ for PPO	0.2
ϵ of the fair-efficient reward	0.1
Discounted factor γ	0.98
Controller PPO	1 dense layer, 128 nodes
Controller value n.	1 dense layer, 128 nodes
S-p PPO	1 dense layer, 256 nodes
S-p value n.	1 dense layer, 256 nodes

Table 2: Default parameter setting for the job scheduling scenario. s-p stands for sub-policies, n for network and nb for number.

Parameter	value
# agents	10
# resources	3
# episodes	1000
# steps	1000
T (POO batch size)	50
# s-p	4
Learning rate for value n.	10^{-3}
Learning rate for PPO n.	$3 \cdot 10^{-3}$
ϵ for PPO	0.2
ϵ of the fair-efficient reward	0.1
Discounted factor γ	0.98
Controller PPO	1 dense layer, 128 units
Controller value n.	1 dense layer, 128 units
S-p PPO	1 dense layer, 256 units
S-p value n.	1 dense layer, 256 units

Table 3: Default parameter setting for the matthewеffect scenario. s-p stands for sub-policies, n for network and nb for number.

Parameter	value
Grid world	8x8
# agents	5
# resource	8
# episodes	1000
# steps	1000
T (POO batch size)	500
# s-p	4
Learning rate for value n.	10^{-3}
Learning rate for PPO n.	$3 \cdot 10^{-3}$
ϵ for PPO	0.2
ϵ of the fair-efficient reward	0.1
Discounted factor γ	0.98
Controller PPO	1 dense layer, 128 nodes
Controller value n.	1 dense layer, 128 nodes
S-p PPO	1 dense layer, 256 nodes
S-p value n.	1 dense layer, 256 nodes

Table 4: Default parameter setting for the manufacturing plant scenario. s-p stands for sub-policies, n for network and nb for number.