Reproducing the results of the paper: "Local Coordination in Online Distributed Constraint Optimization Problems" (Brys et al., 2012)

Learning dynamics (INFO-F-409)

Leon Babin[1], Yanfang Guo[2], Anthony Piron[1] and Wenjia Wang[2]

[1]Master in Bio-informatics and Modelling, Université Libre de Bruxelles
[2]Master in Applied Computer Science, Vrije Universiteit Brussel

Abstract

Reproducibility is a major challenge in science and we are massively failing at it (Ioannidis, 2005). In computer science, arguably the easiest science to reproduce, this is unfortunately also a major issue (Collberg and Proebsting, 2016) (Collberg et al., 2015). In this paper, we will discover the complexity of reproducing research through an attempt to replicate the results of a specific paper for the learning dynamics course. The odds being against us (see previous papers), we obtained results diverging from the paper.

Introduction

In this paper, we will try to replicate the paper "Local Coordination in Online Distributed Constraint Optimization Problems" (Brys et al., 2012). This paper proposes a solution to a multiple agents coordination problem where individual choices have to be made in order to collect a optimal joint payoff. The classical Joint Action Learning (JAL) provides a solution to the problem but the action space grows exponentially in the number of agents (Claus and Boutilier, 1998). The problem becomes rapidly intractable in practicable cases.

The authors introduce a method called Local Joint Action Learners (LJAL) in which the agents limit their interactions with the other agents and only coordinate when necessary. Moreover, the algorithm is able to learn how to select the best agents to coordinate.

In the three next sections, we will describe in three steps the methods employed and the result of the reproduction. The three sections respectively expose:

- the LJAL algorithm applied on a distributed n-armed bandit problem ;
- the application of the LJAL algorithm on Distributed Constraint Optimization problem (DCOP) ;
- the final algorithm which learns for each agent the best candidates to coordinate to.

Local Joint Action Learners

Description of the game

It's a cooperative game requiring coordination between agents. Each of the different agents (5 here), plays a n-armed bandit game where they have to choose between different actions (4 here). They all get the same reward, and the goal is to get the best possible reward, while optimizing computation speed. They don't have a personal reward, the only thing they know is how much they get together at each round, the global reward.

With 5 agents and 4 actions for each agents, there is $4^5 = 1024$ possible rewards. Each combination of actions has a fixed reward, that has been previously drawn from a normal distribution N(0,50) (50 = 10*#agents).

Method

Joint Action Learner (JAL) is a Reinforcement Learning method where every agent knows about the action choices of the others estimate and can observe the rewards associated.

JAL is resources intensive because every single agent have to track the habits of every other agent. The action space grows exponentially in the number of agents (Claus and Boutilier, 1998).

Local Joint Action Learner (LJAL) is similar to JAL but each agent only observe a subset of the other agents. Each agent observes between zero and all the other agents in the game. Zero correspond to the Independent Learner (IL) and all to a JAL. The communication pattern between agents is represent by an oriented graph called a connection graph (CG) (see Figure 3).

Using LJAL, we can change the number of connections between players. If we have no connection, every agent is an Independent Learner (IL) (Figure 3), basing his future choices only on his own previous choices. IL is very fast because every player needs only to maintain an estimation for each possible action. This comes with a cost: we do not obtain a very good solution because the agents lack information about others. If we have all the possible connections, it's JAL which gets the best solution but

takes a lot of time because there is the number of actions to the power of the number of agents estimations to maintain ($(n_{actions})^{n_{agents}}$). With LJAL, we can create a connection graph with the number of connections we wish and consequently, to find some balance between speed and quality. The more numerous connections are, the more time it takes but the solution quality is higher. The number of connections will be referred as degree (of connection) or as averaged number of partners (Avg#Partners)

LJAL is a compromise between speed and quality in multi-agent coordination problems. It act as a intermediate between the fast IL and the quality favoring JAL.

Action selection method The action selection at each step t of the LJAL algorithm makes use of the following formulas:

- Estimation:

$$Q_{t+1}(a) = Q_t(a) + \alpha[r(t+1) - Q_t(a)] \tag{1}$$

  where Q is the estimated value, r is the reward obtained at each round, $\alpha \in [0, 1]$ is the learning rate parameter. $\alpha = 0$ means the Q value is never updated and 1 that the agent forgets everything and takes the new reard r as the new estimation.

- Frequencies:

$$F^i_{a_j} = \frac{C^j_{a_j}}{\sum\limits_{b_j \in A_j} C^j_{b_j}} \tag{2}$$

  with $F^i_{a_j}$ the frequency estimated by agent i that agent j will play action a, and $C^j_{a_j}$ is the count of how many times agent j played action a. A_j is the set of possible actions for agent j.

- Expected Value (EV):

$$EV(a_i) = \sum_{a \in A^i} Q(a \cup a_i) \prod_j F^i_{a|j|} \tag{3}$$

  $EV(a_i)$ is the value expected by player i if he chooses action a. It's the sum of the different possible Q for action a, weighted by the observed frequencies. A^i represents all the possible joint actions of other players that agent i observe.

- Boltzmann distribution (aka softmax distribution):

$$Pr_{a_i} = \frac{e^{EV(a_i)/\tau}}{\sum\limits_{b_i=1}^{n} e^{EV(b_i)/\tau}} \tag{4}$$

  Pr_{a_i} is the probability for agent i to select the action a_i. τ is the temperature factor, it impacts the greedy/exploration ratio. An high temperature means more exploration. For example, in the firsts tests, the temperature is given by $\tau = 1000 * 0.94^{play}$, meaning that τ decreases with time (number of play) and consequently the exploration decreases as the time increases.

Algorithm

Algorithm 1 The LJAL algorithm.

function LJAL_ONE_STEP(agents, graph)
  for all agent in agents **do**
    $evs[] \leftarrow EV_{agent}(Q_{agent}, C_{agent})$
    $actions[agent] \leftarrow BoltzmannAction(evs)$
  $R \leftarrow Reward(actions)$
  for all agent in agents **do**
    $\pi_a[] \leftarrow \pi_{agent}(actions)$
    $Q_{agent}[\pi_a] \mathrel{+}\!\leftarrow \alpha * (R - Q_{agent}[\pi_a])$
    for all s in the successors of agent **do**
      $C^{agent}_{actions[s]} \mathrel{+}\!\leftarrow 1$
  return R

Initialize the Qs, Cs
$fixedReward \leftarrow \mathcal{N}(0, \sigma)$
for all t in MaxTime **do**
  LJAL_ONE_STEP(agents, graph)

The algorithm 1 describes the Local Joint Action Learning algorithm where:

- the Q, C data structure are as described in the previous section ;
- R is the reward ;
- $\pi_{agent}(actions)$ is the projection of the list of actions to the list of observed actions by the agent ;
- $BoltzmannAction(evs)$ chooses an action accordingly to the softmax distribution given the list of expected values for each actions.

The algorithm proceeds by following those steps:

1. Select an action ;
2. Compute the joint reward ;
3. Update Q, C for each agent ;
4. Return reward ;
5. Iterate.

This implementation follows loosely the algorithm structure described in Sutton and Barto (2017) as the description in the paper is mainly prosaic with the help of mathematical formulas.

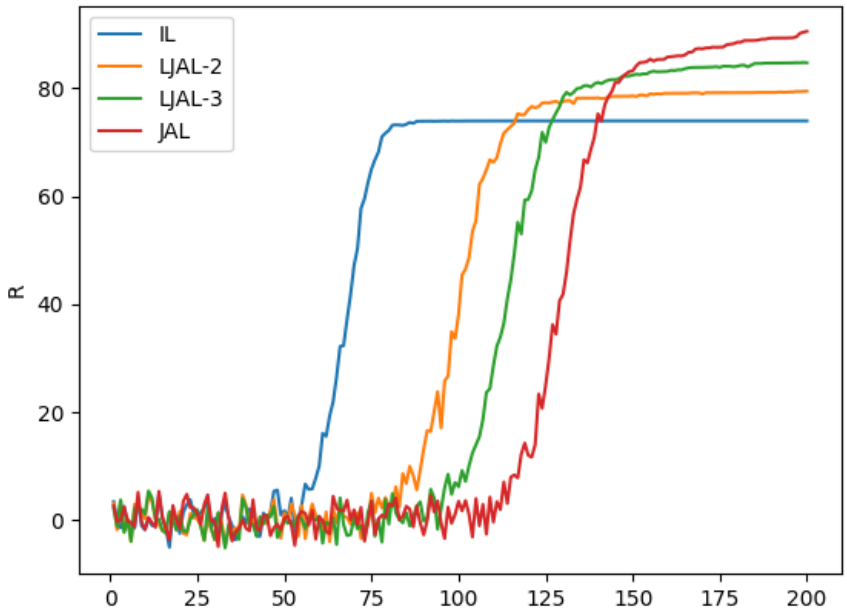

Figure 1: Comparison of independent learners, joint action learns and local joint action learners on a distributed bandit problem.

Learner	Avg # Partners	Speed	Solution Quality	95% Conf. Interval
IL	0.00	$\times 16.6$	81.7%	73.94 ± 0.26
LJAL-2	2.00	$\times 9.6$	87.8%	79.44 ± 0.27
LJAL-3	3.00	$\times 3.7$	93.6%	84.70 ± 0.29
JAL	4.00	$\times 1.0$	100.0%	90.51 ± 0.31

Table 1: Distributed bandit problem: Comparison of speed and solution quality.

Results

We ran the algorithm with four different types of Learners. IL, JAL, LJAL-2 and LJAL-3. LJAL-2 and LJAL-3 have a randomdly generated connections graph respectively with a degree of 2 and 3 (the average number of partners for each agents).

The results are average over 10000 trials of 200 plays with α=0.5.

We see in Figure 1 that IL is the fastest to reach an Equilibrium, and that the reward is the lowest. A contrario, we observe that JAL gets the best reward of all at the end but takes longer to reach this equilibrium. We also confirm with LJAL-2 and LJAL-3 that we can achieve a pretty good solution with much less connections than in JAL. As expected as the number of connections increases, the solution quality gets better but at the cost of the speed to reach it.

As shown in Table 1, computation time is longer when there are more connections.

LJAL results are strictly in between the both extreme cases IL and JAL. LJAL performances don't go beyond the expected range of values obtained by IL and JAL in this case.

Our results are consistent with those of the original paper.

Distributed Constraint Optimization Problem

Description of the game

In this section, we will use the LJAL algorithm to solve a distributed constraint optimization problem (DCOP). in DCOP, a variable is assigned to each agent in the domain corresponding to the variable as it would select an arm from an distributed n-armed bandit problem. Additionally, there are constraints $c_i(D_1, \ldots, D_n) \mapsto \mathbb{R}$ between the variable domains projecting to domain of the rewards. In our specific case, we consider binary constraints ($c_i(D_i, D_j)$ the constraint depend only of two variable domains) of two type of constraints: important and unimportant. A cost w_i is associated with constraints. The w_i for important constraints is 0.9 and the w_i for unimportant constraints is 0.1. The rewards are given by $\mathcal{N}(0, \sigma w_i)$. The full description of the problem is given in Brys et al. (2012).

In this game, we have a set of agents (7 here), and each agent could choose from 4 actions. The constraints between (agent1, agent2), (agent2, agent3), (agent1, agent3), (agent5, agent6) are important, so the w_i are set to 0.9. While the other constraints are unimportant and the w_i are set to 0.1.

We use five different coordination graphs which are IL, JAL, LJAL-1, LJAL-2, LJAL-3 to play the game. The LJAL-1 is a random 2-degree coordination graph. The LJAL-2 is a coordination graph matching the problem structure (the constraints). The LJAL-3 has a supplementary connection between agent1 and agent5 than LJAL-2. The LJAL-4 has a additional connection between agent 4 and agent 7 than LJAL-3. Connection graphs are shown at figure 3

Method

In part 1, the problem was symmetric: each agent and each connection has the same impact on the final reward. In this case, we have a distributed Constraint Optimization Problem (DCOP) where constraints are weight put on connections between agents. In DCOP, all connections don't have the same weight meaning that two agent with a lot of weight on their connection have a greater impact on the final reward. Weight values range from 0 to 1. A weight of 0 means a connection without any impact on the reward. A weight of 1, a connection with great impact on the final reward and consequently, the coordination between those 2 agents will be important.

We can illustrate the binary constraints with a graph[1] varying the thickness of each edge (the constraint): the thicker the connection, the more impact it has on the reward.

With weights on connections, we assume that the connection graphs giving the best results will be the ones reflecting weights on the constraints. If the constraints with a huge weight appears in the connection graph, we do expect a the better quality of the solution.

Algorithm

Algorithm 2 Distributed Constraint Optimization Problem

function SET_REWARD(full_connection_graph)
  for all node1 in full_connection_graph.nodes **do**
    for all node2 in full_connection_graph.nodes **do**
      $w_i \leftarrow getWeight(Edge(node1, node2))$
      $\sigma \leftarrow 10 \# agent$
      $RewardMat[node1][node2] \leftarrow \mathcal{N}(\mu,\, w_i\sigma)$
  return $RewardMat$

function ONE_STEP(agent,graph)
  for all agent in agents **do**
    $evs[] \leftarrow EV_{agent}(Q_{agent}, C_{agent})$
    $actions[agent] \leftarrow BoltzmannAction(evs)$
  $R \leftarrow 0$
  for all agent1 in agents **do**
    for all agent2 in agents **do**
      $R+ \leftarrow Reward(agent1.action, agent2.action)$
  for all agent in agents **do**
    $\pi_a[] \leftarrow \pi_{agent}(actions)$
    $Q_{agent}[\pi_a] +\leftarrow \alpha * (R - Q_{agent}[\pi_a])$
    for all s in the successors of agent **do**
      $C^{agent}_{actions[s]} +\leftarrow 1$
  return R

Initialize the Qs, Cs
$fixedReward \leftarrow$ SET_REWARD(full_connection_graph)
for all t in MaxTime **do**
  DCOP_ONE_STEP(agents, graph)

The algorithm 2 describes how we use the LJAL to solve the Distributed Constraint Optimization ProblemCAI and CAO (2010). In this game, each agent need to play with

[1] This graph is not the coordination graph.

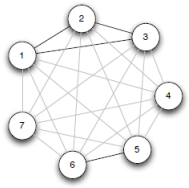

Figure 2: Weight of the constraints between agents. Dark edges mean important constraints ($w_i = 0.9$), light edges are unimportant constraints ($w_i = 0.1$) (source: Brys et al. (2012)).

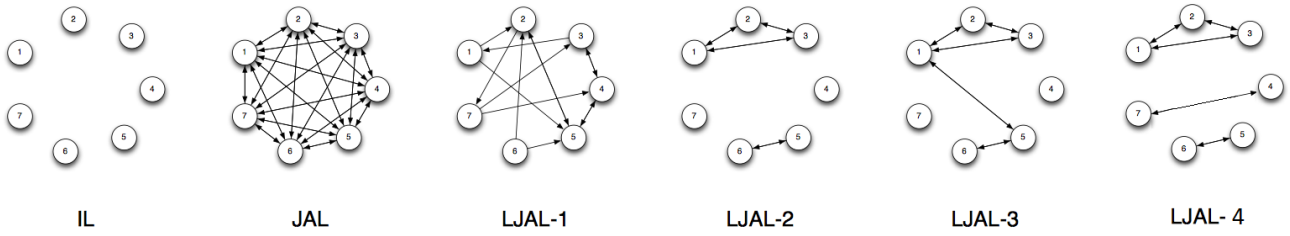

Figure 3: Different local joint action learners, visualized by their coordination graphs. (source: Brys et al. (2012)).

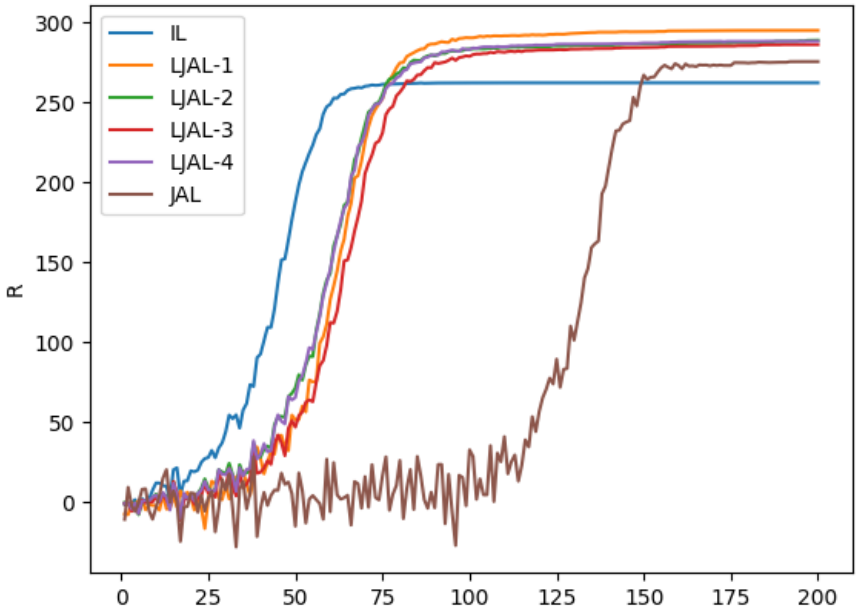

Figure 4: Comparison of independent learners, joint action learns and local joint action learners on a distributed constraint optimization problem.

Learner	Avg # Partners	Speed	Solution Quality	95% Conf. Interval
IL	0.00	$\times 396$	95.2%	262.31 ± 0.80
LJAL-1	2.00	$\times 192$	107.1%	295.05 ± 0.70
LJAL-2	1.14	$\times 292$	104.8%	288.90 ± 0.72
LJAL-3	1.43	$\times 195$	103.9%	286.20 ± 0.69
LJAL-4	1.43	$\times 183$	104.6%	288.33 ± 0.64
JAL	6.00	$\times 1$	100.0%	275.55 ± 5.16

Table 2: distributed constraint optimization problem: comparison of speed and solution quality.

each other and the final reward is given by the sum of reward for each sub-game.

The algorithm 2 proceeds by following those steps

1. Set the reward for this matrix according to the constraints weights;
2. Initialize the Qs,Cs;
3. Select an action ;
4. Computing the sum reward ;
5. Update Q, C for each agent ;
6. Return reward ;
7. Iterate.

Results

We used 6 different CGs. IL and JAL, and 4 other graphs. LJAL-1 is a randomly generated CG with a degree of 2 (average of 2 partners by agents). LJAL-2 is a constructed CG that reproduce heavy weighted constraints only ($w_i = 0.9$). LJAL-3 and 4 are the same as LJAL-2, but with 1 arc added (see Figure 3).

As in part I, IL is the quickest and gets the worst solution of all (see figure 4 and table 2). JAL is still the slowest to reach a steady solution. Curiously, it doesn't get the best answer. This might be partially caused by a lower number of samples for JAL (1000 sample average for JAL versus 10000 for the others). This cannot be the full explanation as the 95% confidence interval is far bellow the LJAL algorithms (All values are statistically significant). This begs the question: Is to much bad information detrimental?

LJAL-2 reproduces, in the connection graph, the constraints with a heavy weight ($w_i = 0.9$). Therefore, the computation speed is the fastest (apart from IL) because it has the less connections (Avg#Partners=1.14). The Solution Quality is still high, despite the few connections (9.6 above IL).

LJAL-3 and LJAL-4 are the same connection graphs as LJAL-2, but we added an arc between agents 1 and 5 in LJAL-3, and between 4 and 7 in LJAL-4 (see Figure 3). The Impact of the added arc on the solution is very different for those 2 cases. In LJAL-3, we added an arc with a light weight but between two agents involved in heavy weighted constraints. Therefore, it is harder for them to estimate properly the different rewards with the new irrelevant information to process. It slows down computation, and they get a solution slightly lower (but statistically significant) than LJAL-2. Even if it's minimal, unnecessary informations impact both computation time and solution quality.

On the other hand for LJAL-4, we added an arc between 2 agents not involved with heavy constraints. Agents 1, 2, 3, 5 and 6 behave the same way they did in LJAL-2. Computation speed and solution quality are only a little lower than LJAL-2 (0.2 difference in solution quality and not statistically significant difference).

Adding more connections can have a negative impact on the solution, and especially for agents that are involved in high weighted constraints. The difference between LJAL-3 and LJAL-4 shows us that more connection not only slower the computation, but can lower the solution quality.

LJAL-1 is a random generated graph with a connection degree of 2 (each agents observe 2 other agents). It has the most connections (after JAL), and get the best answer of all. It goes against the conclusion of the article which states that non useful connections, i.e. introducing more variability, tends to lower the estimation quality.

Another strange thing is that despite having more connections than LJAL-3 (Avrg nbrpartner=1.43), the computation speed almost the same, it even is a bit faster for LJAL-1 than LJAL-3[2].

Unexpectedly, we didn't get the same results as the original article. One explanation might be the difference of value for the learning rate α. We used $\alpha = 0.5$ for this part[3], but we don't know what value they used in the original article. Furthermore, the α value has proven to be a huge source a variability during our tests. Another explanation could be the number of trials we did, that were only 10000 instead of the 100000 made in the paper but the confidence intervals

[2] I got contacted by the ULB datacenter: I was not using properly the multi-processing on the cluster (Hydra). This can be the cause of the variability. Unfortunately, it is too late to re-run the very intensive computations.

[3] Multiple value not reproduced here where tested.

seems to disprove this hypothesis.

However differences in our results (especially for JAL and LJAL-1) remain unexplained.

Learning Coordination graphs

Description of the game

In the previous section, we discussed that matching the CG to the problem structure could help to improve the solution quality. But the problem structure may not be known beforehand, so in this section we want use the LJAL to learn the coordination graph.

In this game, learning a CG could be seen as a distributed n-arm bandit problem. In this game, the optLJAL-1 and optLJAL-2 are that we limit the number of cooperation partners to 1 and 2, the actions for optLJAL-1 and optLJAL-2 are all possible neighbor selections. After several iterations, the LJAL is expected to find the connection graphs.

Algorithm

Algorithm 3 Coordination graph learning algorithm.

```
function BUILDDCOPGRAPH(actions)
    // build dcop graph from the actions
    edges ← ACTIONTOEDGES(actions)
    Initialize the DCOP graph with n_agents=7, n_actions = 4
    for all edge in edges do
        DCOPGraph.add(edge)
    return DCOPGraph

function REWARD(actions,LCGGraph)
    DCOPGraph = BUILDDCOPGRAPH(actions)
    R = 0
    for all t in n_times do
        R+ ← DCOPGraph.R
    return R / n_times

Initialize the Qs, Cs, α, τ for LCGGraph
for all t in MaxTime do
    LJAL_ONE_STEP(agents, LCGGraph)
```

The algorithm 3 describes how we learn the coordination graphs. We first initialize the actions of choosing neighbors, then use our LJAL to learn the optimal actions. The algorithm 3 proceeds by following those steps:

1. Each agent selects an action ;
2. Build the coordination graph according to the actions of selecting neighbors;
3. Use this coordination graph to play the DCOP game in part2.
4. Get the reward;
5. Update Q, C for each agent ;
6. Return reward ;
7. Iterate.

Method

We play the same game as in DCOP part but this time we don't construct the connection graph before: we let the agent choose the connections by playing an other distributed n-armed bandit game (IL). In this meta-game, the actions are a list of the agent which they want to coordinate with. Each agent chooses a fixed number of other agents to connect with. An agent can choose himself meaning that he doesn't coordinate with someone else.

Once the agents have chosen their actions (which agents they coordinate with), we obtain a connection graph. The meta-game use this graph to play a DCOP game (200 plays, averaged 100 times) and compute the reward for this graph. This reward is used in the meta-game to estimate the quality of the selected graph.

Then they repeat the n-armed bandit game again based on the new estimations, and so on.

Agents are Independent Learner in the meta-game, in order to reduce complexity, but it could potentially be done using JAL or LJAL as well, but computation time already is prohibitive[4].

In order to simplify, the agents may only choose the fix constant number of connections (we tested with one and two connections). They may choose themselves and therefore, choose to not connect. Consequently, the average number of partners is most of the time below one or two.

With this method, agents can find the best connection graph for a certain number of connections (degree of connection).

Results

The results are reproduced on figure 5 and table 3. We were unable to reproduce the results of the paper. The algorithm seems not being able the learn the graph. It is quite possible that this is just a question of parameter tuning. But the performance of our program is not sufficient to test with multiple settings. We probably will have to rewrite the program in a more efficient language than python[5] like C/C++. We were only able to average 40 trials. We do not see the slightness tendency to learn, just random noise.

Discussion

Based on the results we got, it is hard to draw any releavent conclusion about the efficiency of this method. However there is still some teachings worth remembering from this experiment.

First thing, we saw in the first part LJAL that LJAL was a good compromise between IL and JAL in terms of speed

[4] In the paper, $1500 * 1000 * 200 * 100 = 30G$ "one step" function call in the DCOP function. And there is still loops to compute the frequencies, and so on.

[5] Even with numpy and some part compiled with Cython, this is not sufficient.

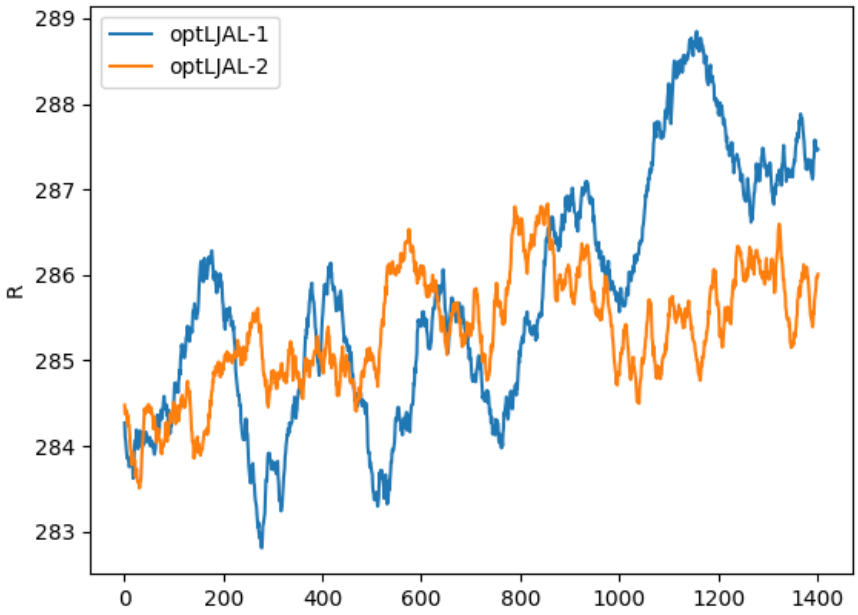

Figure 5: Comparison of learners the learners of coordination graph with one edge by node at maximum (optJAL-1) and two edge by node at maximum (optJAL-2) (Moving average over 100 points).

Learner	Avg # Partners	Speed	Solution Quality	95% Conf. Interval
optLJAL-1	0.92	$\times 1.0$	100.6%	286.19 ± 9.19
optLJAL-2	1.62	$\times 1.0$	100.0%	284.39 ± 10.25

Table 3: Comparison of learners the learners of coordination graph with one edge by node at maximum (optJAL-1) and two edge by node at maximum (optJAL-2): comparison of speed and solution quality.

and Quality of the solution. LJAL is a generalization were IL and JAL can be seen as the both extremities.

Second thing, we still manage to prove that the addition of irrelevant information on some agents nonetheless reduces computation speed and can even affect the quality of the solution.

Another thing underlined by our work is that missing information, such as the value of the learning parameter α in the original paper can affect in a negative way the reproductability of an experiment and greatly challenge inexperienced researchers. We can relate to the issue exposed in Collberg and Proebsting (2016) and Collberg et al. (2015).

The lack of computational power, as well as the use of a not efficient enough programming language can also greatly affect computation time, affecting researchers ability to conduct a sufficient amount of trials and parameter testings[6]. This issue was hardly predictable as there is only relative timing in the paper and not absolute timing data.

An idea to extend the work of the original article would have be to try understanding the structure of connection graphs obtained with optLJAL (see (Brys et al., 2012)). One hypothesis was that those graphs would have reflected the "follow the leader" strategy of the agents described in the same article. Unfortunately due to the inability to accurately reproduce previous work, this hypothesis was not explored.

Conclusion

Local Joint Actions Learner method was developed as a trade off between Joint Action Learner and Independant Learner for coordination problem in cooperative game.

LJAL has shown its efficiency for this problem: a small number of connections between agents significantly increases the solution quality.

LJAL is also efficient when coordination between certain agents is more important than other, and has even shown better results than JAL (Although this might be due to an undetermined mistake in our work).

Using this method to let agents learn how to connect has been unconclusive in our research. However the original paper (see (Brys et al., 2012)) was conclusive, and our implementation of the algorithm could probably be improved in order to get better results.

References

Brys, T., De Hauwere, Y.-M., Nowé, A., and Vrancx, P. (2012). *Local Coordination in Online Distributed Constraint Optimization Problems*, pages 31–47. Springer Berlin Heidelberg, Berlin, Heidelberg.

CAI, S.-j. and CAO, J. (2010). Distributed constraint optimization. *Journal of Chinese Computer Systems*, 8:034.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 746–752, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Collberg, C., Proebsting, T., and Warren, A. M. (2015). Repeatability and benefaction in computer systems research. studie.

Collberg, C. and Proebsting, T. A. (2016). Repeatability in computer systems research. *Commun. ACM*, 59(3):62–69.

Ioannidis, J. P. A. (2005). Why most published research findings are false. *PLoS Med*, 2(8):e124.

Sutton, R. S. and Barto, A. G. (2017). *Reinforcement Learning : An Introduction (Draft)*.

[6] as well as their sleeping time