Review: Accelerating Multi-agent Reinforcement Learning with Dynamic Co-learning

Steve Homer[1], Fabian Perez[1], Quinten Rosseel[1] and Matthias Humt[1]

[1]{steven.homer, fabian.perez, quinten.rosseel, matthias.humt}@vub.be

Abstract

This report reviews the paper from Garant et al. (2015) which introduces a new approach to identify periodical experience transfer opportunities in large-scale, stochastic, homogeneous multi-agent systems (MAS) operating in a distributed manner. By using supervisory agents that compute and reason over high-level characterizations (contexts), similarities between agents are identified. Experiments show that the use of this method in the right settings can accelerate MAS learning significantly. We frame our understanding of the work and conclude with an empirical validation.

Introduction

In the standard Reinforcement Learning (RL) setting (Sutton and Barto, 1998), an agent is placed into an unknown environment and provided with a set of actions from which it can choose. By performing an action, the agent can change its state which is then solely defined by the previous state the agent was in before and the chosen action. In certain states, defined by the problem setting, the environment provides feedback to the agent, often called *reward* which can be either positive or negative. From this, the agent begins to approximate the underlying reward function in trying to maximize the expected future reward. Once having started to learn, the agent has to decide whether to exploit current knowledge by choosing the actions it expects to yield the highest reward or to explore new states with potentially higher rewards by trying new state-action combinations. This methodology is typically chosen, if the state-space is large and explicitly defining the correct actions to achieve the goal is either infeasible or even impossible as they are not known. An often cited example in this context is learning to ride a bike (Randlov and Alstrom, 1998). The desired outcome is clear, but giving advice how to achieve it in terms of the correct movements to perform is quite difficult.

Multi-Agent Reinforcement Learning (MARL) is an extension to the standard RL problem setting, where multiple, often hundreds or thousands of autonomous agents try to pursue their individual goals simultaneously in a common environment. We speak of cooperative MARL, if multiple or all agents pursue the same goal, which means they try to maximize a common reward function (Tuyls and Nowé, 2005). If they succeed, we say that they follow a (near-optimal) joint policy where policy means a mapping from states to actions. To achieve this however has proven to be difficult for large-scale multi-agent systems as it is computationally expensive and requires a large number of update steps until it converges.

An important insight on the path to solving this problem is the observation, that an exploitable structure in the problem setting–a distributed load balancing problem (Figure 1), where the agents try to share the work among each other as to minimize processing time–in the form of contextually similar groups of agents emerges during learning. Real world examples of this effect can be observed in disaster planning and sensing networks. Agents working on similar tasks under comparable environmental dynamics might benefit from sharing information to accelerate their individual learning progress which is the paradigm proposed in the paper under review. Additionally, strategies to identify promising candidates for information sharing and group formation are proposed, which will be introduced in the next section along with the test setup. In previous work, agents have either been trained individually on their local environment and neighbors, not being able to benefit from the experience of their peers, or as a hive, optimizing a single joint policy, which becomes quickly intractable even for a moderately large number of agents. But identifying contextually similar groups which are likely to greatly benefit from information sharing comes with its own difficulties. What information should be transferred? What does *similar* mean? How well does this approach scale? Section will explore some of these questions in detail.

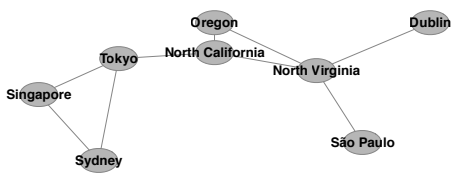

Figure 1: Small load balancing network (Garant et al., 2015)

Methods

Several questions arise when designing a solution for this problem. I.e. How to identify groups in which information sharing is possible? What information should be transferred between agents in those groups? How can convergence in distributed and concurrent settings be guaranteed? This section provides a high level overview of the authors conceptual framework and associates their scheme with our implementation of the model. We highlight several design decisions and attempt to provide insights in difficulties and non-trivialities that pop up in the concretization process.

Supervisory Architecture The authors propose a model of supervisor and subordinate agents. The subordinates can be seen as the domain specific workers of the system that aim to optimize a joint, domain-specific problem (e.g. load balancing distribution to minimise total service time). Supervisors keep track of subordinate groups and aim to accelerate the learning process using high level feature traits, collected from the subordinates.

The process starts by creating *subordinate groups* (Figure 2) consisting of supervisors and subordinates. These groups remain fixed over the entire learning process. After groups are formed, supervisors periodically search for *contextual compatible* subordinates in their subordinate group to compose experience *sharing groups*. In these groups, subordinates are able to share learned experiences with any other member from the sharing group. Experience sharing between subordinates from different supervisors is not allowed. The contextual compatibility depends on factors like metric-defined interactions within a certain group of agents, also referred to as interaction sparsity in literature (e.g. D^C distance measure in implementation) or aggregate effects over group interactions (e.g. amount of sent messages between agents over a time period).

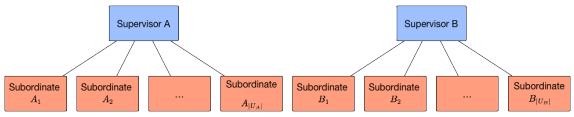

Figure 2: Supervisor hierarchy (Garant et al., 2015)

In order to keep the shared experiences concise, time-windows of t timesteps are defined to characterize policy, reward and transition changes. Considering that decisions are framed as a Markov Decision Process (MDPs), experiences for one time window $[t_0, t_e]$ can be represented as a vector of tuples with s_i, a_i, r_i and s'_i corresponding to current state, action, reward and next state of the subordinate as depicted in figure 3. To be even more concise, another possibility would be to merge experiences within a given time window. This is disregarded due to increased complexity and information loss risks when states are not visited uniformly.

$$\Big\langle (s_{i_{t_0}}, a_{i_{t_0}}, r_{i_{t_0}}, s'_{i_{t_0}}), \\ (s_{i_{t_1}}, a_{i_{t_1}}, r_{i_{t_1}}, s'_{i_{t_0}}), \\ \dots \\ (s_{i_{t_e}}, a_{i_{t_e}}, r_{i_{t_e}}, s'_{i_{t_0}}) \Big\rangle$$

Figure 3: Vector of experience tuples

In a real word setting, experiences could be compressed by using linear regression equations, fitted to variable sets in the window. This is especially effective when agents infrequently change state. These *compressed experiences* can be shared between subordinates using a supervisory agent as proxy. This allows the supervisor to derive context features from reported experiences and share it with subordinates that desire so. Though useful to save network bandwidth in a practical sense, from a theoretical point of view, compression is not desirable since it only adds complexity to the system. We chose to send raw, uncompressed experiences in order to update subordinate Q-values. The diagram below gives an overview of the model.

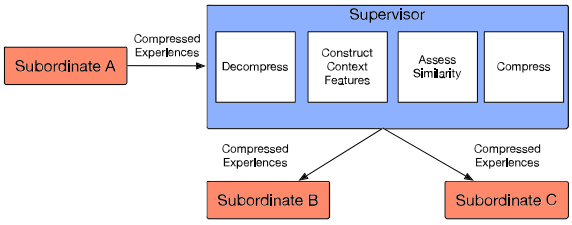

Figure 4: Supervisory structure (Garant et al., 2015)

To conclude the authors theoretical framework we discuss how context features are constructed from the experiences that are passed along. We note that features like experience imbalance, policy divergence, unobservable state features and disjoint state visitations are attention points to take into account during context feature construction. More information on these concepts can be found in the paper. Since the context feature construction is domain-dependent, the authors only provide a general approach for doing so. The following paragraphs will explain our approach dealing with the proposed problem, but also ambiguities and missing information.

Random Graph, Supervisory Topology, Task Allocation For each trial, a random network is generated for the given

size and branching factor parameters. The supervisor topology is then imposed randomly on this network, differing from the original where the subordinate group of a supervisor is a highly connected cluster. Since the supervisor assignment is random, no assumptions can be made about the connectivity of other agents in the subordinate group. In addition, tasks are randomly allocated with random service time. The goal of utilizing so much randomness in the construction of the simulations is to minimize design bias in the network and preclude any cross-domain assumptions that might creep in, especially in supervisor assignment.

Worker Q-Learning with Softmax and Experience Integration Workers in the network improve their performance using a basic Q-learning algorithm with $\gamma = 0.1$ and a *softmax* decision policy with $\tau = 0.1$. Integrating experiences from other agents is done in a naive fashion, extracting the reward and neighbor index from the experience to update its corresponding Q-values for each shared experience.

Distance Function The distance functions for each type of context feature follow a similar pattern that allows for proper interaction with the Boltzmann distribution used in the sharing selection algorithm.

$$D^C(C_1, C_2) = \lceil 10e^{-|C_1 - C_2|} \rceil / 10$$

The quantized inverse exponential absolute difference produces discrete values that are near 1 when the distance is near 0 and and near zero when the distance grows large. This allows the Boltzmann distribution to correctly weight small context feature distances with higher probabilities.

Context Feature Construction Although the paper states that context feature construction is a transformation of an agents experiences for a given window into a vector or context features, it is only possible to create the context features only from the experiences of an agent by encapsulating all of the information relevant to the context features in the state of the agent. This sounds trivial, but in this case requires recording the local state (i.e. load, environment task allocation rate, and agent task allocation rate) of each of its neighbors into its own state. Instead of encapsulating all of this data in this extended state, in our implementation the supervisor queries the agent and its neighbors for the information it requires and creates the context features from those responses. These two methods are equivalent because the exact same information is gathered from querying as extracting from state. This allows us to have a simpler state representation.

Results and Discussion

To present our results, we use the same methodology as the original paper to facilitate comparison. The primary measure of performance is the *Area Under the Learning Curve* (AUC). It is computed by taking the difference between the originally assigned service time of each task and the actual time needed by the network to solve it. The mean of this difference in service time per time step is then smoothed using an exponential moving average with a smoothing factor $\alpha = 2/(span + 1)$, where *span* is set to the time period for which the simulation was run (i.e. $10,000$). The minimum mean value is then subtracted from each remaining value to "lower" the curve onto the x-axis. Taking the area under this curve results into the AUC measure. This measure reflects the performance of a parameterization of the algorithm, as a we want the difference in assigned and consumed service time to be as small as possible over as long a time span as possible, which is precisely what the AUC provides.

Figure 5 gives an example of the baseline architecture compared to the one supervisory configuration. To facilitate reproduction of our results, we provide, for each figure, the exact parameterization used to generate it. We will use, where applicable, S for the number of supervisors, K for the size of the learning window, N for the size of the network. For all simulations, a *maximum branching factor* of 10 and a *maximum service time* of Todo: Add value were used and the results are the average over 30 trials. When we speak of the *baseline configuration* the following parameters were used: $S = 0, K = 25, N = 100$.

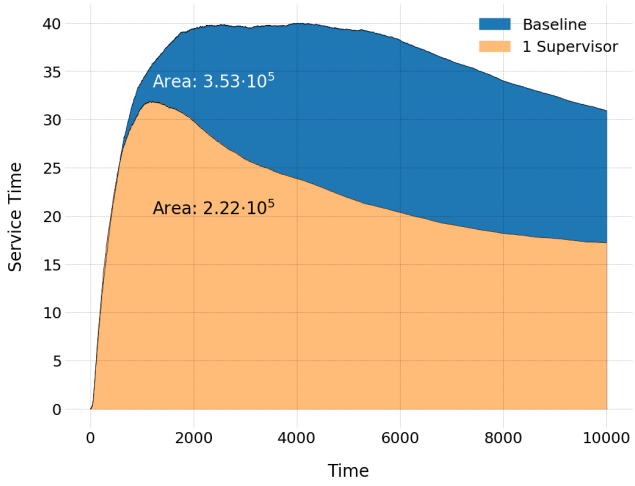

Figure 5: Task Allocation Performance Evaluation Parameters: $K = 25, N = 100$

Clearly, the one supervisory configuration performs much better than the baseline configuration. In the beginning, when the network has not yet learned to distribute the load, the average service time rises quickly. Around 1000 time steps however, the supervisor has identified sharing groups and optimizes the learning process of the subordinates, resulting in fast convergence to a better task distribution. The agents in the baseline configuration also learn to distribute tasks, but do so at a much slower pace and with less success. Compared

to the figure 5 of the original paper, the results qualitatively comparable, but not identical. The rate and difficulty of the tasks generated by the environment seem to be higher, as the maxima for both the baseline and one supervisor learning curve are stronger. Todo: Add other reasons for this difference.

To see the effect of changing the parameters of the model, we will now take a look at a performance evaluation for different learning window sizes K. The AUC is not compared directly, but in comparison to the baseline architecture (for which changes in the learning window have no effect). Figure 6 below depicts the results.

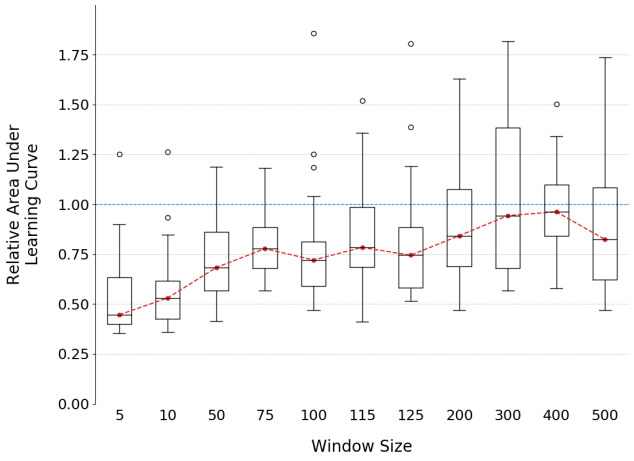

Figure 6: Effect of varying winow size K
Parameters: $S = 1/0, N = 100$

Again, our results compare well to those seen in the paper. A rise in K is generally positively correlated with a rise in relative AUC. There are occasional outliers, just like in the paper, which could be due to small sample size. Todo: Anything else? The biggest difference is, that even for $K = 500$ the performance of the one supervisory configuration still outperforms the baseline. As seen before in figure 5, this could be due to easier environment settings. Todo: Or what?

In figure 7 we explore the effect on the learning behavior when using a greater number of supervisors. The one supervisor scenario is a best case assumption, where sharing groups can be formed between the best suited candidates from the whole population of agents. With a rising number of supervisors the probability to identify contextually similar agents decreases, resulting in a loss of performance. This effect can be observed in both our results and those from Garant et al.

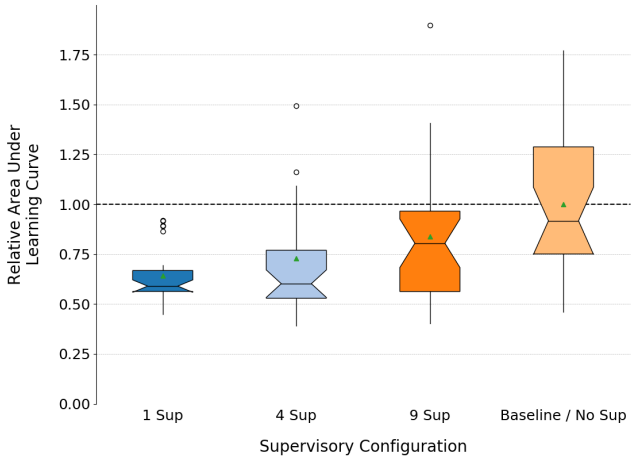

Figure 7: Relative Performance by Supervisory Config.
Parameters: $K = 25, N = 100$

While the variance of the results increases with a growing number of supervisors, consistent with the reference, it does so more rapidly in our work. Additionally the variance of the baseline configuration is much higher, possibly due to a different approach in computing the *relative* AUC for which there is unfortunately no information provided in the original work. Todo: Any other ideas? Our approach takes the average over all 30 trials of the baseline configuration to reference against. To highlight the trend towards the baseline performance when adding supervisors, we added the mean value for all trials represented as small green triangles.

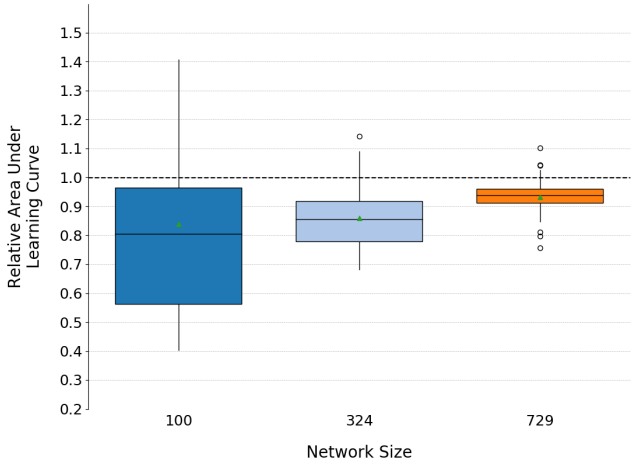

Figure 8: Relative Performance as network size is varied
Parameters: $S = 1/0, K = 25$

The last parameter influence we want to explore is the network size. Figure 8 shows the relative performance of three different sizes against their respective baselines. It has to be noted, that individual baseline data had to be generated

for each network size, as this is the first parameter affecting its performance, contrary to the learning window size and the number of supervisors (obviously, as there is none). Here we see the first real divergence from the results seen in the paper. While in the original work performance increases with growing network size–explained by the increase in sharing opportunities per supervisor (see also Figure 7)–it drops in our approach. Another notable difference is the drastic decrease in variance when growing the network while this cannot be observed in the reference work. Todo: Someone has to explain this...

Conclusion

Todo: As this section was moved, the introduction should probably be changed

Complains That being said, it shows that the choice of context features heavily influences the representation of the state of an agent. For example here, an agent records information in its state that one would intuitively not think of as that agents state, like each of its neighbors loads and rates. In multi-agent reinforcement learning, agents generally want to make decisions based solely on information gained from reward signals, in this case from other neighbors. It seems odd then to need to record the internal state of those neighbors for the context features to be created solely rom experiences. If you choose a more intuitive representation of state, i.e. information completely local to the agent, then creating the context features outlined in the paper is impossible. That is why the querying implementation was used here, because information required to create the context features could not be recovered from experiences alone without being extremely generous in what counts as the state of an agent..
Todo: More complaining after..?

The problem stems from the fact that supervisor has no knowledge of the problem domain a priori, and therefore must extract all information from experiences. To see why it is impossible to extract context features from experiences alone, consider the following intuitive representation of an experience for the load-balancing problem, over a time window of size K:

```
State(load, environmentRate, agentRate)
Process or Forward(i, agentID)
 State (load, environmentRate, agentRate)
InverseService(r)
```

Todo: Is this (above) what you wanted Steve?
The supervisor can recover the neighbors of the agent by looking at where the agent has forwarded to this window. That might not be all, or any, of its neighbors so we only recovered a subset of those neighbors.

If we allow the supervisor to look at other experiences in its memory of other agents (which the paper does not explicitly allow, but seems reasonable), then we can recover the relevant information from the experiences of the agents that are neighbors of the given agent, but only if those neighbors are in the supervisors memory.

Therefore, to get all of the information necessary to accurately calculate the context features for a given agent, that agent would have had to forward to all of its neighbors at least once in the window and those neighbors would also have had to be part of the same subordinate group (and then theres still problems and the edges of the subordinate group).

Since this intuitive representation is untenable, we have to choose between using a bloated, unintuitive representation of state, or just querying the agents for the information we require. We chose the latter.

Summary This paper reviewed a method to accelerate learning in MAS, proposed by Garant et al. (2015). By means of a personal implementation, we examined theoretical and empirical analysis of the authors. This allowed us to reason and comment on design decisions, metrics, optimizations and ambiguities that followed from the work. After providing an introduction and methodology background, we highlighted implementation details and concluded the paper with our obtained results and corresponding discussion. Overall, we observed comparable results with slight variance but noted that these results were very sensitive to MAS and architecture settings.

Outlook Todo: outlook
Todo: Make sure all questions given below are answered

1. Does the introduction explain clearly the content of the paper - Maybe not?
2. whether there is sufficient background information to understand the relevance of the work - Yes!
3. whether the methods are clearly explained (can the results be reproduced?) - Yes!
4. whether the results answer the questions asked in the paper. Todo: Identify questions
5. whether all questions are answered - Which ones??
6. whether the conclusion is sufficient - We'll see.
7. and whether the overall style is ok and - Oh yes!
8. whether you believe things are missing in the discussion. - Nope.
9. etc. - Yes.
10. 3 positive points concerning the work, clearly specifying why you think they are well-done or interesting
11. 3 negative points, which may include missing/unclear explanations or suggestions for improvement
12. at least 3 clear and relevant questions on the content or the methods used which can be asked (next to other questions).

References

Garant, D., da Silva, B. C., Lesser, V., and Zhang, C. (2015). Accelerating multi-agent reinforcement learning with dynamic co-learning. Technical report, Technical report.

Randlov, J. and Alstrom, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tuyls, K. and Nowé, A. (2005). Evolutionary game theory and multi-agent reinforcement learning. *The Knowledge Engineering Review*, 20(1):63–90.