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 setting 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, 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. 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, 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 Todo: Add examples. 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?

The authors introduce supervisory agents which are instructed to deal with those questions. A supervisory agent differs from a normal agent only in its ability to receive and act on communications from its subordinates. It forms contextually compatible groups of agents and identifies and shares relevant information between them. We will now take a look at how this is done explicitly.

Todo: Add citations

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.

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). 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. To be even more concise, another possibility would be experience merging within a given time window. This is disregarded due to increased complexity and information loss risks when states are not visited uniformly.

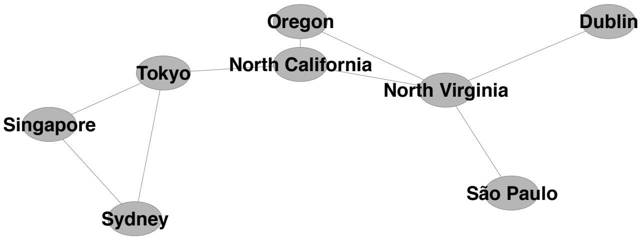

Figure 1: A small load balancing network (Garant et al., 2015)

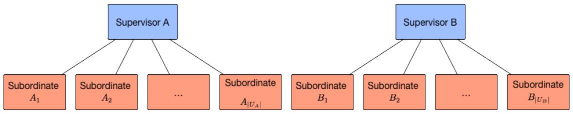

Figure 2: Supervisor hierarchy (Garant et al., 2015)

$$
\Big\langle (s_{it_0}, a_{it_0}, r_{it_0}, s'_{it_0}), \\
(s_{it_1}, a_{it_1}, r_{it_1}, s'_{it_0}), \\
\dots \\
(s_{it_e}, a_{it_e}, r_{it_e}, s'_{it_0}) \Big\rangle
$$

Todo: Explain this thing above

In a real word setting, experiences could be compressed by using linear regression equations, fitted to a variables set 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. 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

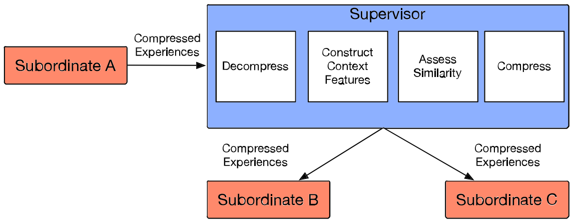

Figure 3: Supervisory structure (Garant et al., 2015)

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 was generated for the given parameters of size and branching factor. The supervisor topology was then imposed randomly on this network, differing from the original where the subordinate group of a supervisor was highly connected cluster. Since the supervisor assignment was random, no assumptions could be made about the connectivity of other agents in the subordinate group. In addition, the tasks were randomly allocated with random service time. The goal of utilizing so much randomness in the construction of the simulations was 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 improved 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 was 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 followed a similar pattern that allowed for the proper interaction with the Boltzmann distribution used in the sharing selection algorithm. (EQUATION) : $D^C(C1, C2) = round(0.1, e^-(abs(C1 - C2)))$ 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

Though stated in the paper 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 allowed us to have a simpler representation of state.

COMPLAINING

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..

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:

$$< State(load, environmentRate, agentRate), ProcessorForward(i,$$

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.

Results

To present our results, we use the same methodology as the original paper to facilitate comparison.

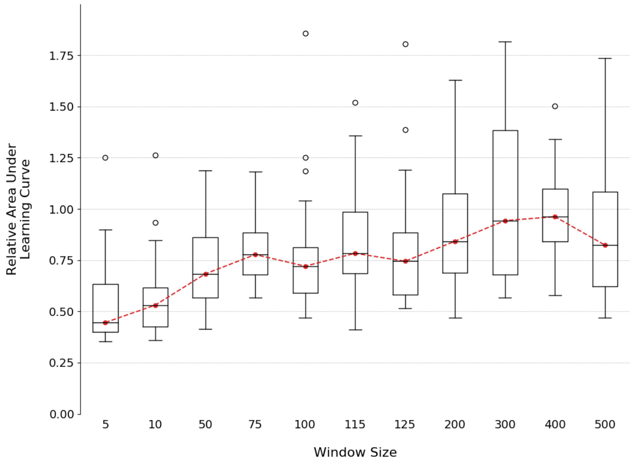

Figure 4: Effect of varying winow size K

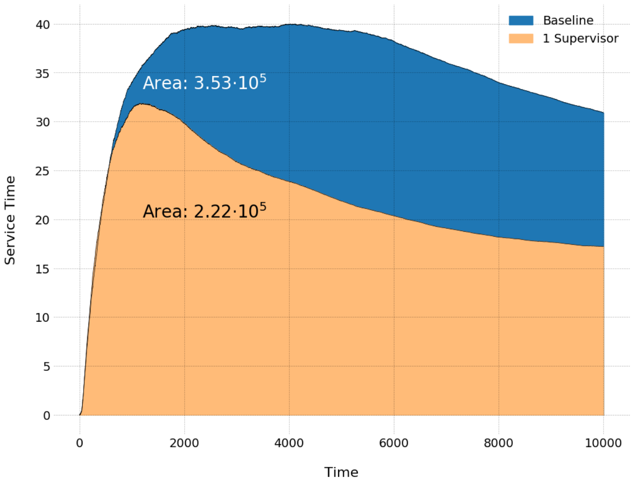

Figure 5: Task Allocation Performance Evaluation

Todo: Describe the performed simulations and their results
Todo: Include chosen parameter settings

Discussion

Todo: Summary and explanation of our work

Conclusion

Todo: Add related work and an outlook
Todo: Make sure all questions given below are answered

1. Does the introduction explain clearly the content of the paper

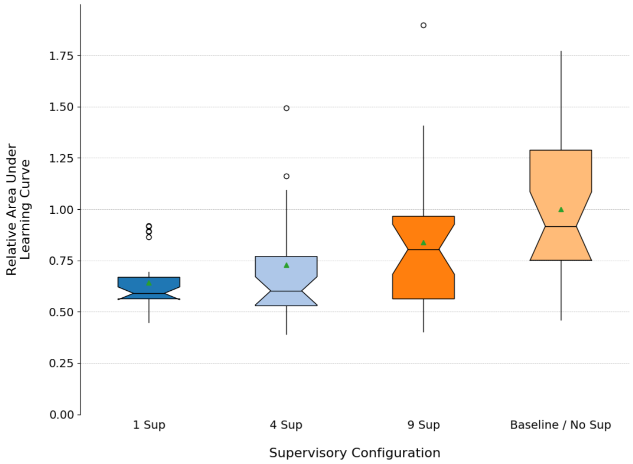

Figure 6: Relative Performance by Supervisory Configuration

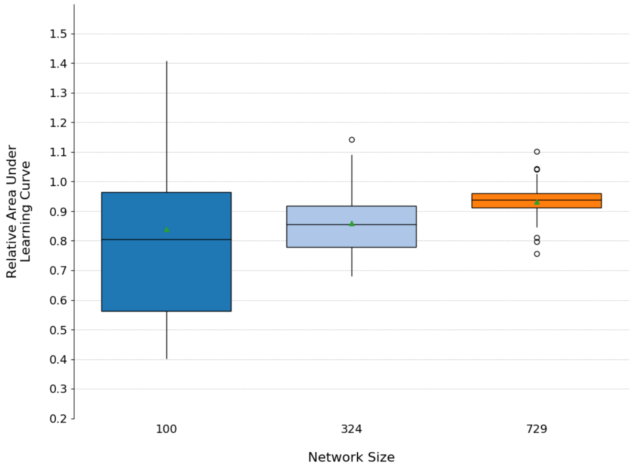

Figure 7: Relative Performance as network size is varied

2. whether there is sufficient background information to understand the relevance of the work
3. whether the methods are clearly explained (can the results be reproduced?)
4. whether the results answer the questions asked in the paper.
5. whether all questions are answered
6. whether the conclusion is sufficient
7. and whether the overall style is ok and
8. whether you believe things are missing in the discussion.
9. etc.
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).

(Carroll and Seppi, 2005), (Garant et al., 2015),
(Ghavamzadeh et al., 2006), (Gmytrasiewicz and Doshi, 2005),
(Guestrin et al., 2002), (Kitano et al., 1999),
(Lazaric et al., 2008), (Littman, 2001),
(Nair et al., 2005), (Nedic and Ozdaglar, 2009),
(Oliehoek et al., 2008), (Price and Boutilier, 2003),
(Rényi et al., 1961), (Taylor and Stone, 2009),
(Vickrey and Koller, 2002), (Witwicki and Durfee, 2010),
(Zhang et al., 2010), (Zhang and Lesser, 2013)

References

Carroll, J. L. and Seppi, K. (2005). Task similarity measures for transfer in reinforcement learning task libraries. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 803–808. IEEE.

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.

Ghavamzadeh, M., Mahadevan, S., and Makar, R. (2006). Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229.

Gmytrasiewicz, P. J. and Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.(JAIR)*, 24:49–79.

Guestrin, C., Koller, D., and Parr, R. (2002). Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 739–743. IEEE.

Lazaric, A., Restelli, M., and Bonarini, A. (2008). Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551. ACM.

Littman, M. L. (2001). Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66.

Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI*, volume 5, pages 133–139.

Nedic, A. and Ozdaglar, A. (2009). Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61.

Oliehoek, F. A., Spaan, M. T., Whiteson, S., and Vlassis, N. (2008). Exploiting locality of interaction in factored dec-pomdps. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 517–524. International Foundation for Autonomous Agents and Multiagent Systems.

Price, B. and Boutilier, C. (2003). Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629.

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.

Rényi, A. et al. (1961). On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Vickrey, D. and Koller, D. (2002). Multi-agent algorithms for solving graphical games. In *AAAI/IAAI*, pages 345–351.

Witwicki, S. J. and Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled dec-pomdps. In *ICAPS*, pages 185–192.

Zhang, C. and Lesser, V. (2013). Coordinating multi-agent reinforcement learning with limited communication. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1101–1108. International Foundation for Autonomous Agents and Multiagent Systems.

Zhang, C., Lesser, V., and Abdallah, S. (2010). Self-organization for coordinating decentralized reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 739–746. International Foundation for Autonomous Agents and Multiagent Systems.