Heuristic Coordination in Cooperative Multi-Agent Reinforcement Learning

Ramon Petri[1], Eugenio Bargiacchi[2],
Huib Aldewereld[1], and Diederik M. Roijers[1,2]

[1] HU University of Applied Sciences, Utrecht, The Netherlands
ramon.petri@student.hu.nl &
{huib.aldewereld,diederik.yamamoto-roijers}@hu.nl
[2] Vrije Universiteit Brussel, Brussels, Belgium
{eugenio.bargiacchi,diederik.roijers}@vub.be

Abstract. Key to reinforcement learning in multi-agent systems is the ability to exploit the fact that agents only directly influence only a small subset of the other agents. Such *loose couplings* are often modelled using a graphical model: a coordination graph. Finding an (approximately) optimal joint action for a given coordination graph is therefore a central subroutine in cooperative multi-agent reinforcement learning (MARL). Much research in MARL focuses on how to gradually update the parameters of the coordination graph, whilst leaving the solving of the coordination graph up to a known typically exact and generic subroutine. However, exact methods – e.g., Variable Elimination – do not scale well, and generic methods do not exploit the MARL setting of gradually updating a coordination graph and recomputing the joint action to select. In this paper, we examine what happens if we use a heuristic method, i.e., local search, to select joint actions in MARL, and whether we can use outcome of this local search from a previous time-step to speed up and improve local search. We show empirically that by using local search, we can scale up to many agents and complex coordination graphs, and that by reusing joint actions from the previous time-step to initialise local search, we can both improve the quality of the joint actions found and the speed with which these joint actions are found.

Keywords: Coordination Graphs · Local Search · Multi-agent Reinforcement Learning · Multi-agent Thompson Sampling

1 Introduction

Coordination is an important aspect of everyday life – whether playing football, or participating in traffic. In artificial intelligence, coordination between multiple artificial agents is therefore a popular topic, with applications ranging from robotic rescue operations [Visser et al., 2014, Chalup et al., 2019] to maintenance scheduling on highways between multiple contractors [Scharpff et al., 2016, Scharpff, 2020].

Key to keeping cooperative multi-agent coordination tractable is to exploit so-called *loose couplings*, i.e., the property that individual agents typically only *directly* affect a small subset of the other agents. For example, imagine a wind farm, where each agent controls the yaw of a wind turbines [Verstraeten, 2021]. A turbine produces turbulence in its wake, which can affect the wind turbines behind it. The turbines behind the first one may in turn affect other wind turbines, ultimately still requiring coordination between the entire wind farm, but the first turbine only directly affects the ones behind it. Such loose couplings can be expressed using a graphical reward structure called a *coordination graph* [Verstraeten et al., 2021]. In coordination graphs, direct influence between agents is modelled as a local reward function, that has the joint action space of the agents affecting and being affected as its domain.

When the coordination graphs and all its local reward functions are known, the optimal joint action can be found using non-serial dynamic programming [Bertele and Brioschi, 1972], or as it is more commonly known in the agents community, *variable elimination (VE)* [Guestrin et al., 2002]. Variable elimination is an exact algorithm [Rosenthal, 1977], but due to the inherent hardness of solving coordination graphs, scales poorly in the connectivity of a graph, and typically also in the number of agents.

When the ground truth local reward functions in a coordination graph are unknown to the agents, we are in the multi-objective multi-armed bandit setting [Verstraeten et al., 2020]. In this setting, the local reward functions must be learned through interaction with the environment (e.g., the wind farm) [Bargiacchi et al., 2018]. A state-of-the-art algorithm for doing so is called *multi-agent Thompson sampling (MATS)* [Verstraeten et al., 2020]. MATS uses VE as a subroutine to find the optimal joint action for the graphs sampled before each interaction with the environment. As such, MATS scales poorly in the connectivity of the coordination graphs, and typically also the number of agents in the graph.

In this paper, we study the effect of using *local search (LS)* [Russell and Norvig, 2005] algorithms as a subroutine in multi-agent Thompson sampling. These heuristic algorithms scale extremely well in the number of agents and the connectivity of the coordination graphs, but they are of course not exact. We can therefore expect to incur more *regret* [Verstraeten et al., 2020], i.e., a larger cumulative difference between the optimal team rewards and the rewards resulting from the joint actions that are performed, then when using VE. We do however expect a large gain in runtime, especially for increasingly complex coordination graph.

We observe that when we use LS as a subroutine inside of MATS, there are some aspects that we can exploit. Firstly, as the newly obtained information obtained at each timestep has a relatively smaller impact on the posterior beliefs over the true mean rewards, the sampled coordination graphs at each timestep are increasingly similar. Secondly, at each timestep, we produce a joint action to execute using LS. Finally, local search algorithms can benefit from initialization with a good initial solution, i.e., an educated guess for a joint action. We therefore

propose to reuse the joint action found by LS at the previous timestep as the starting point for LS in the next timestep. Using this together with an iterative search scheme, this leads to our *reusing iterative local search (RILS)* algorithm. We show experimentally that RILS is able to find good approximate solutions for coordination graphs. When used in MATS, this leads to higher regret, but at a fraction of the runtime of MATS with VE as a subroutine. Furthermore, the difference in regret becomes smaller as the coordination graphs become more complex, while the difference in runtime becomes larger. We therefore conclude that RILS is a suitable algorithm to scale up to complex coordination graphs in multi-agent multi-armed bandits.

2 Background

A *coordination graph (CoG)* models the (sparse) relationships between multiple cooperative agents. In a coordination graph, each agent is represented by an individual node. An edge between two nodes indicates that coordination between their associated agents is required to achieve optimal behaviour. We note that while edges in coordination graphs are often represented in a pairwise fashion, we use a hyper-edge representation, where each edge can connects several agents at once [Roijers et al., 2015b]. Each hyper-edge is associated to a *local reward function*, which specifies the rewards for a subset of agents.

In planning, the local reward functions specify a deterministic (or expected) local reward given a local joint action by the connected agents [Roijers, 2016]. In this paper, however, we are concerned with a reinforcement learning setting in which the local reward functions are stochastic, and specified using a distribution over local rewards. This is called a multi-agent multi-armed bandit (MAMAB) [Bargiacchi et al., 2018]. More formally, a multi-agent multi-armed bandit (MAMAB), is a tuple $\langle D, A, f \rangle$ where:

- D is the set of all m agents.
- $A = A_1 \times \ldots \times A_m$ is the joint action space.
- $f : A \rightarrow \mathbf{R}$ is the global reward function, i.e. a random function[3] associating each full joint action to a sampled reward. In a MAMAB, f can be decomposed into a set of ρ independent local reward functions, such that $f(a) = \sum_{e=1}^{\rho} f^e(a^e)$. Note that each of these constituent components are again random functions.

Intuitively, it should be possible to exploit the structure of f and the coordination graph to quickly learn the local reward functions, without incurring in an exponential regret from the large full joint-action space of a multi-agent setting (the curse of dimensionality). Multi-agent Thompson Sampling (MATS) [Verstraeten et al., 2020] is an algorithm that does precisely this: it maintains a

[3] A random function is the function equivalent of a random variable, i.e., a function which is defined in terms of an experiment of which the outcome varies according to a given probability distribution. As such evaluating a random function for the same input twice may yield a different output.

posterior distribution of the mean rewards for each possible local joint action, which it samples when it needs to act in the MAMAB. Such a sample leads to a coordination graph with non-stochastic rewards. The full joint action selected is then the one that maximizes the reward across all sampled local arms. This strategy provably [Verstraeten et al., 2020] results in a regret that is linear in the number of agents, rather than exponential.

In order to select the best joint action, MATS must maximize across all local arms in a computationally efficient manner. To do this, MATS relies on an exact algorithm that was originally devised to marginalize discrete variables in probabilistic graphical models. This is not surprising, as the concept of a coordination graph is analogous to an undirected graphical model.

In particular, MATS uses a well-known algorithm called Variable Elimination (VE) [Bertele and Brioschi, 1972, Rosenthal, 1977, Guestrin et al., 2002]. Originally developed to perform exact inference, VE can be used to determine the optimal action of multiple agents maximizing a factored reward function, in our case f. VE is an iterative algorithm, which progressively removes each agent from the coordination graph after computing its best response w.r.t. its neighbors, i.e., for each local joint action of the neighbors it determines the action that maximizes the total reward of the group. The advantage of VE over naive brute force search is in its computational complexity, which is combinatorial on the induced width of the graph, i.e., the largest local action space considered during the elimination process.

The computational complexity of VE for sparse coordination graphs is much lower than naive brute forcing, which is exponential in the number of agents. However, VE still tends to perform poorly when dealing with large number of agents, as the induced width typically increases with the number of agents, albeit much slower than the number of agents itself.In turn, this prevents using the MATS algorithm in large scale bandits, unless VE is replaced by an approximate selection technique.

A popular approximate optimization algorithm that is called Iterative Local Search (ILS). This technique is based on Local Search (LS) as described by [Russell and Norvig, 2005] and has an extension in the form of an iterative variant (ILS) as described by [Lourenço et al., 2003]. Local search algorithms take an optimisation problem – such as a coordination graph – and find approximate solutions by starting with a random solution, and looking in the neighborhood of the current solution, as defined by a set of allowed small mutations, for improvements. Iteratively applying such improvements until no improvements can be found in the neighborhood leads to a so-called local optimum. ILS can escape such local optima, by performing larger random mutations and re-applying local search.

3 Algorithms

In this paper we investigate the potential of applying *local search* and *iterative local search* schemes as an approximate subroutine in MATS [Verstraeten et al.,

2020] to replace the exact VE subroutine. First, we define how local search can be applied to coordination graphs. Secondly, we create an algorithm that performs iterative local search while exploiting the *reinforcement learning* setting. Specifically, at each timestep, the MATS algorithm learns more about the local rewards functions, and updates its local posterior mean reward distributions, before re-sampling a coordination graph to select the next timestep's joint action (using the VE or local search subroutines). We make the following observations about the learning process of MATS using (iterative) LS as a subroutine:

- The sampled coordination graphs at each timestep are increasingly similar. This is because the new information gathered at each timestep has a diminishing impact on the posterior mean reward distributions with respect to the information already gathered. Morevoer, over time, the posterior mean reward distributions become increasingly certain about what the local mean rewards ought to be, leading to narrower distributions and therefore more similar samples.
- At each timestep, we produce a joint action to execute.
- Local search algorithms can benefit from initialization with a good initial solution.

Combining these observations, we observe that it is likely beneficial to reuse the joint action found and executed at the previous timestep as the initial starting solution for (iterative) local search in the current timestep. We propose an algorithm - the Reusing Iterative Local Search algorithm (RILS) - that does so. We thus exploit the multi-agent reinforement learning setting to speed up our heuristic search subroutine.

3.1 Local Search for Coordination Graphs

The Local Search (LS) algorithm for coordination graphs (Algorithm 1) works by incrementally updating a joint action with local improvements, i.e., changes in actions for a single agent that improve the global reward. Starting from a random joint action (an array of individual actions for each agent), ar, the algorithm goes through all the agents in the graph in random order and for each agent, v, through all the actions, a available to that agent, to check whether replacing $ar[v]$ with a yields an improvement. We note that to do so, the algorithm only needs to calculate the difference, Δ, in reward for the sum of local reward functions that have agent v in scope. This therefore takes only a fraction of the time of a full evaluation of ar over the entire graph. If that Δ is bigger than zero, i.e., a is an improvement upon the current action of agent v, $ar[v]$ is changed to a. The algorithm uses a flag *changed* to check whether the last pass over the agents yielded an improvement; it is set to continue the while loop until no higher rewards can be found. When no local improvements upon ar can be found by updating the action of any of the agents in the coordination graph, LS has converged to a local optimum and the total team reward is returned. The total team reward is evaluated by a function named $evalTeamReward$ that sums

over all the local reward functions. This makes *evalTeamReward* an expensive operation, and it is therefore key to the performance of LS as a subroutine within MATS that this happens only once per call to LS.

Algorithm 1 Local Search

```
1: procedure LOCAL SEARCH(startAction)
2:     ar ← startAction or a random joint action if startAction is null
3:     changed ← true
4:     while changed do
5:         changed ← false
6:         for v ∈ agents do                ▷ In a different random order each iteration.
7:             for a ∈ actions[v] do
8:                 Δ ← evaluateLocalActionChange(ar, v, a)
9:                 if Δ > 0 then
10:                    ar[v] ← a
11:                    changed ← true
12:                end if
13:            end for
14:        end for
15:    end while
16:    return (ar, evalTeamReward(ar))
```

LS can be used by itself as a subroutine within MATS. In this case, LS then starts from a random joint action each time that it is called. While this is no doubt a highly efficient heuristic, it lacks in two key aspects: the local optima achieved do not converge to the optimal joint action over time, due to the random nature of LS, and, in the multi-agent reinforcement learning setting, we start anew at each timestep to select a joint action, disregarding the information about how good the joint action that LS found on the previous timestep was.

3.2 Reusing Iterative Local Search (RILS)

Iterative Local Search (ILS) uses Local Search as a subroutine to escape local optima, by making larger randomized changes, called perturbations, to the solution after LS runs into a local optimum and then rerunning LS to see whether this leads to further improvements. Note that as this can in principle continue indefinitely, typically a maximum number of iterations is set to limit the number of trials in which ILS can maximize its result.

The added randomization uses a so called perturbation probability (PP), i.e., with a probability PP each part of the solution is set to a random value. For coordination graphs we employ local-reward-function-based perturbations, which means that we iterate over all local reward functions, and with probability PP, the actions for all agents in scope of the reward function are changed to a random action. We chose this over an agent-based perturbation strategy, because

Algorithm 2 RILS

```
1: procedure Reusing Iterative Local Search(numOfTrials, PP, PRandom, previousAction)
2:     if previousAction = Empty ∨ rn < PRandom then
3:         ar ← randomAction()
4:     else
5:         ar ← previousAction
6:     end if
7:     val ← evalTeamReward(ar)
8:     for i ← 0 to numOfTrials do
9:         ac ← ar
10:        rn ← randomnumber ∈ [0, 1]
11:        if previousAction = Empty ∨ rn < PRandom then
12:            ac ← randomAction()
13:        else
14:        for Each local reward function f^e do
15:            rn ← randomnumber ∈ [0, 1]
16:            if rn < PP then
17:                Change actions of all agents in scope of f^e to a random action in ac.
18:            end if
19:        end for
20:        ac', val' ← LS(ac)                                   ▷ Algorithm 1
21:        if val' > val then
22:            ar ← ac'
23:            val ← val'
24:        end if
25:    end for
26:    previousAction ← ar
27:    return ar, val
```

if the action of a single agent changes, without any actions of its neighbours changing, LS will change this action straight back towards the previously found local optimum.[4]

In order to exploit the multi-agent reinforcement learning setting in the MATS algorithm, we propose Reusing Iterative Local Search (RILS) (Algorithm 2). The algorithm works by checking if a *previousAction* is available, from the previous iteration of the MATS algorithm. If not (i.e., this is the first timestep of MATS), the current joint action ar gets initialized with a random joint action. If it is available, ar is initialized with the *previousAction*. Subsequently, ar is evaluated to get its team reward, val. Note that it is necessary to re-evaluate the previous joint action between timesteps, as MATS samples the local reward functions each timestep, leading to slightly different local rewards.

[4] In order to make sure, we did in fact try out the agent-based perturbation strategy as well, but this indeed proved far less effective, so for the remainder of this paper we only use the local-reward-function-based perturbation strategy.

The main loop of RILS runs through a number of trials, $numOfTrials$, to try and find better solutions than the previous joint action (or the randomly generated one). This is done by first perturbating ac using the previously described local-reward-function-based perturbation strategy, after which it gets passed to Algorithm 1: LS. If the new local optimum found by LS, ac' with value val' improves over the previous ar, this joint action ac' replaces the current best, ar.

While iteratively improving upon the same joint action can be effective, and can save a lot of runtime due to efficient initialisation, there is also a risk. Specifically, RILS might get stuck in the same local optimum for a very long time, especially if that local optimum turns out to be hard to escape by small random perturbations. Therefore, RILS also has a very small probability, $PRandom$, to start from a completely random solution at the beginning of its main loop.

When the number of trials are up, RILS stores the best found joint action, ar in $previousAction$, and returns it along with its team reward, val.

4 Experiments

We now compare *Local Search (LS, Algorithm 1)* and *Reusing Iterative Local Search (RILS, Algorithm 2)* against Variable Elimination (VE) [Guestrin et al., 2002] as a subroutine within the Multi-Agent Thompson Sampling (MATS) algorithm [Verstraeten et al., 2020], both in terms of regret and in terms of runtime, for increasingly complex MOMABs. We use the implementations of VE and MATS found in the AI-Toolbox [Bargiacchi et al., 2020]. Additionally, our implementation of LS will be released in the same toolbox.

Our experiments are based on the Gem Mining problem from [Bargiacchi et al., 2018, Verstraeten et al., 2020], which is adapted from the Mining Day problem from [Roijers et al., 2015b], which is a multi-objective coordination graph benchmark problem. Gem Mining is engineered in such a way that the induced width – the primary indicator for the complexity of a coordination graph – can be controlled without changing the number of agents.

In Gem Mining, a mining company mines gems from a set of mines (local reward functions) located in the mountains (see Figure 1). The mine workers live in villages at the foot of the mountains. The company has one van in each village (agents) for transporting workers and must determine every morning to which mine each van should go (actions), but vans can only travel to nearby mines (graph connectivity). Workers are more efficient when there are more workers at a mine: the probability of finding a gem in a mine is $x \cdot 1.03^{w-1}$, where x is the base probability of finding a gem in a mine and w is the number of workers at the mine. To generate an instance with v villages (agents), we randomly assign 1-5 workers to each village and connect it to a between y and z mines. Each village is only connected to mines with a greater or equal index, i.e., if village i is connected to m mines, it is connected to mines i to $i + m - 1$. The last village is connected to z mines and thus the number of mines is $v + z - 1$.

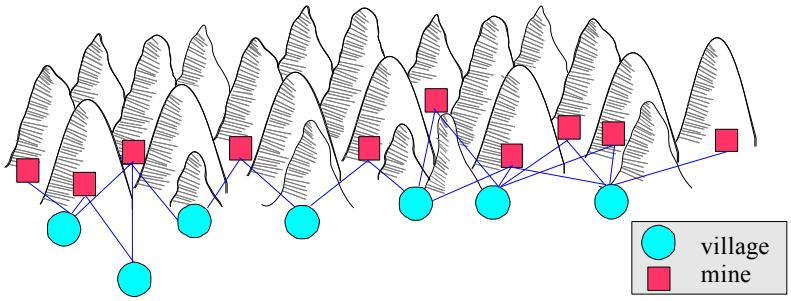


Fig. 1. Gem Mining example. Each village represents an agent, while the mines represent the local reward functions.

4.1 Gem Mining Problem

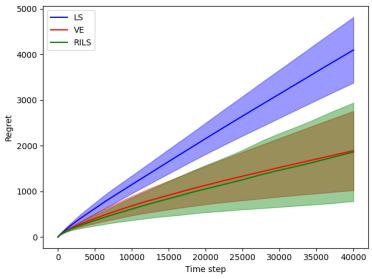
The Gem Mining problem is so constructed that the induced width is limited. This induced width, is the number of neighboring agents (villages) at the time the agent (village) is eliminated by the variable elimination (VE) algorithm [Guestrin et al., 2002].

The induced width can be used as a measure of the complexity of the graph. Due to the chain-like shape of the Gem Mining problem, the elimination order for VE can always be chosen to be from left to right (or the reverse) with minimal resulting induced width. Specifically, as the left-most mine in the graph always has a maximal number of neighbours z, and eliminating that agent does not increase the the number of neighbours of the subsequent villages, the induced width of a Gem Mining problem is always z. The Gem Mining problem is therefore well-suited to show how algorithms behave when the graph complexity increases.

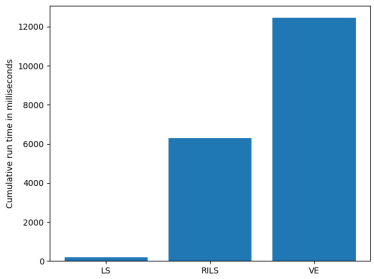
4.2 Results

We run MATS using LS, RILS and VE as a subroutine on the same randomly generated Gem Mining instances of 20 agents (villages), with varying induced width, i.e., between 3 and 6. For each induced width level, we perform 20 runs. For RILS, we use $numOfTrials = 15$, $PP = 0.001$, and $PRandom = 0.0001$, for each experiment. All experiments were performed on a TOXIC-15CL872-1060 customised BTO laptop, with 16.0 GB ram and a processor intel core I7-8750H of 2.20 GHz and 6 cores

For 1-3 villaged per mine, and a resulting induced width of 3, we observe in Figure 2(a) that the cumulative regret of MATS while using LS as a subroutine is significantly higher than that of MATS with VE or RILS. However, MATS with LS uses only a fraction of the runtime (Figure 2(b)) of MATS with VE or RILS. VE uses the most runtime, with RILS using about half the runtime of VE. While MATS with VE and MATS with RILS (with 15 trials) reach about the



(a) Regret vs, timesteps



(b) Run run time in milliseconds VE, HULSA and IHULSA

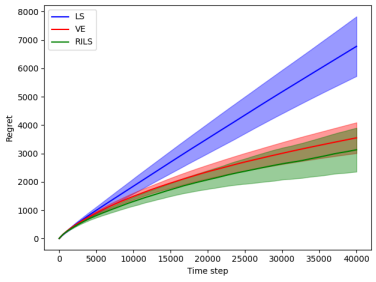
Fig. 2. Results when running MATS with a graph size of 20 villages and 1-3 mines per village

same regret, MATS using RILS has a slightly higher variance in its regret than when using VE. This is expected as RILS is a randomised heuristic algorithm.

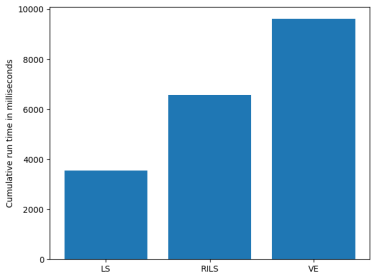
As the induced with increased we observe interesting patterns in regret (Figures 3(a)–5(a)). Firstly, the regret of using LS gets closer to that of using VE and RILS. This is probably because, as the number of neighbouring agents per agent increases, there are possibilities for gradual improvements for an hill-climbing algorithm like LS, i.e., it takes longer to run into a local optimum. Secondly, the regret of using RILS seems to dive under the regret of using VE. This can probably be explained by its reuse – even though the graph sampling of MATS might lead to a new joint action, the joint action of the previous timestep, as reused by RILS, may very well still be a local optimal. Therefore, while VE is guaranteed to follow the exploration mechanism of MATS, RILS is not. While this may lead to better in practice performance, this lack of exploration does break the regret guarantees of MATS [Verstraeten et al., 2020]. For the intent of this paper however, we are mainly interested in scalability and performance.

In terms of runtime (Figures 3(b)–5(b)), we observe a different pattern. Firstly, the runtimes of LS and RILS do increase with the complexity of the graphs. This can be explained from the observation that in some complex graphs it also takes longer to find a local optimum (even though the quality of that local optimum is likely to be higher). However, ultimately, as can be seen in Figure 5(b), for ever more complex graphs, VE has a much larger increase in runtime than LS and RILS.

Another key observation in terms of runtime is that for low induced width (Figure 2(b), LS has a much lower runtime than RILS. RILS uses 15 trials to find new local optima, and its efficiency gain due to reuse is clearly not able to compensate for the multiple trials yet. However, as the induced width increases, and finding a local optimum from a completely random solution takes more time, the runtime of MATS with LS overtakes the runtime of MATS with RILS, even if RILS is using 15 trials instead of the 1 for LS. This indicated that reuse is being effective; the reused initial joint action (i.e., the best joint action found

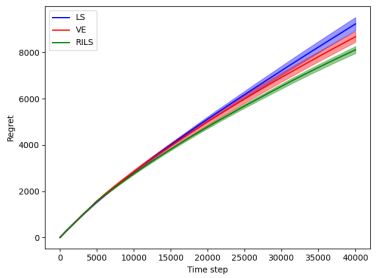


(a) Regret vs. timesteps

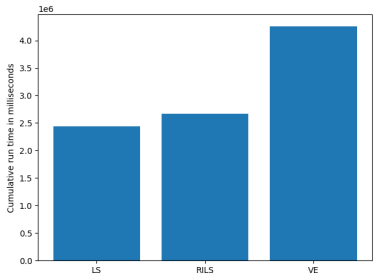


(b) Runtime in milliseconds for MATS in combination with VE, LS and RILS

Fig. 3. Results when running MATS with a graph size of 20 villages and 2-4 mines per village

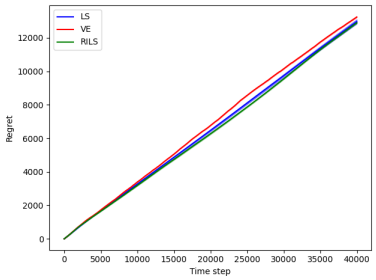


(a) Regret vs. timesteps

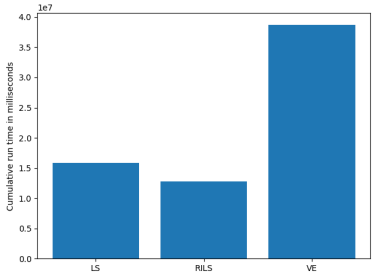


(b) Runtime in milliseconds for MATS in combination with VE, LS and RILS

Fig. 4. Results when running MATS with a graph size of 20 villages and a 3-5 mines per village



(a) Regret vs, timesteps



(b) Runtime in milliseconds for MATS in combination with VE, LS and RILS

Fig. 5. Results when running MATS with a graph size of 20 villages and 4-6 mines per village

for the previous timestep), is much closer to the ultimately selected joint action, and therefore takes considerably less time to find.

We therefore conclude that even though MATS using RILS as a subroutine does lose its theoretical regret bounds due to the heuristic nature of the RILS algorithm, the in practice regret can be good, whilst scaling much better in the complexity, i.e., induced width, of the graphs than VE, and even LS.

5 Related work

In this paper, we have proposed RILS – an approximate subroutine for optimising the joint action in coordination graphs for multi-agent reinforcement learning in a multi-agent multi-armed bandit (MOMAB) setting. For this setting we have integrated RILS with MATS [Verstraeten et al., 2020], the state-of-the-art in MOMABs. Other algorithms also apply to this setting however, and RILS can be used in those algorithms as well. For example, sparse cooperative Q-learning [Kok and Vlassis, 2004, Kok and Vlassis, 2006, Bargiacchi et al., 2018, Verstraeten et al., 2020] can be used in this setting, and RILS can directly replace the joint action selector subroutine there as well. Furthermore, RILS can also be easily adapted for usage in multi-agent upper confidence exploration (MAUCE) [Bargiacchi et al., 2018]. Specifically, as MAUCE keeps vector-valued rewards, and uses an adapted variant of VE that scalarises these to determine which vector is, RILS also should keep vector-valued rewards, and be aware of the value of the whole joint action to determine whether the difference in vector-valued rewards while running LS (Algorithm 1) are indeed improvements.

We note that other than local-search based algorithms. There are also other classes of approximate algorithms that seem promising, such as, a.o., Max-plus [Kok and Vlassis, 2005], AND/OR tree search methods [Marinescu and Dechter, 2005], variational methods [Liu and Ihler, 2013, Roijers et al., 2015a]. We note though that these have not been adapted for the multi-agent RL in MOMABs, and it would be interesting to investigate whether reuse schemes that exploit information from the previous iteration work for those algorithms as well. There may also be potential to use different initialisation schemes that leverage previous observations from interaction with the environment as well. For example, one may consider deep learning for coordination graphs [Böhmer et al., 2020], in order to determine the initial solution before running local search.

Finally, we note that this work may be extended to use in factored or multi-agent MDP settings [Boutilier, 1996]. In such settings, the coordination graph would depend not only on the actions of the agents, but also on state variables, that are provided by the environment. Therefore, multi-agent RL algorithms for this setting (e.g., [Kok and Vlassis, 2004, Kok and Vlassis, 2006, Bargiacchi et al., 2021]) are faced with different coordination graphs at every timestep, but can still use subroutines like VE to find the joint actions. In this context, RILS would have to be adapted, e.g., by finding the last joint action for the most similar state previously observed. Initialisation using deep learning [Böhmer et al., 2020], might be especially promising in this context.

6 Conclusion

In this paper, we proposed the heuristic *reusing iterative local search (RILS)* algorithm, as an alternative to exact joint action finders for multi-agent cooperative reinforcement learning in MOMABs, and specifically in combination with the multi-agent Thompson sampling (MATS) [Verstraeten et al., 2020] algorithm. RILS reuses the joint action found at the previous timestep to initialise its search for a new joint action. This is effective as, as the information accrued through interaction with the environment accumulates, the new information gained at each timestep impacts the learned reward structure (i.e., coordination graph) for the next timestep less and less. This makes the graphs for subsequent timesteps increasingly similar, and therefore the joint action of the previous timestep increasingly likely to be a good initialisation. We have shown experimentally that using RILS is able to closely match the regret for an exact subroutine, while using significantly less runtime. Moreover, its runtime scales better in the complexity of the graphs. We therefore believe RILS can be key to keep multi-agent reinforcement learning in MOMABs scalable for complex graphs.

In future work, we aim to investigate the combination of RILS with different algorithms such as sparse cooperative Q-learning [Kok and Vlassis, 2006] and MAUCE [Bargiacchi et al., 2018]. Furthermore, we aim to investigate larger, and real-world inspired problems, such as wind farms [Verstraeten et al., 2021]. Finally, we aim to investigate how a reusing iterative local search scheme can be applied in reinforcement learning in multi-agent Markov decision processes (MMDPs) [Boutilier, 1996], and multi-objective multi-agent reinforcement learning settings [Rădulescu et al., 2020].

Acknowledgements

The authors would like to acknowledge FWO (Fonds Wetenschappelijk Onderzoek) for their support through the SB grant of Eugenio Bargiacchi (#1SA2820N). This research was supported by funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. This project was supported by the KIEM project "Exploration towards an intelligent Scoliosis brace" that was funded by the Stichting Innovatie Alliantie in the Netherlands.

References

[Bargiacchi et al., 2020] Bargiacchi, E., Roijers, D. M., and Nowé, A. (2020). AI-Toolbox: A C++ library for reinforcement learning and planning (with python bindings). *Journal of Machine Learning Research*, 21(102):1–12.

[Bargiacchi et al., 2018] Bargiacchi, E., Verstraeten, T., Roijers, D., Nowé, A., and Hasselt, H. (2018). Learning to coordinate with coordination graphs in repeated single-stage multi-agent decision problems. In *International conference on machine learning*, pages 482–490. PMLR.

[Bargiacchi et al., 2021] Bargiacchi, E., Verstraeten, T., and Roijers, D. M. (2021). Cooperative prioritized sweeping. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 160–168.

[Bertele and Brioschi, 1972] Bertele, U. and Brioschi, F. (1972). *Nonserial dynamic Programming*. Academic Press, N.Y.

[Böhmer et al., 2020] Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *International Conference on Machine Learning*, pages 980–991. PMLR.

[Boutilier, 1996] Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *TARK*, volume 96, pages 195–210. Citeseer.

[Chalup et al., 2019] Chalup, S., Niemueller, T., Suthakorn, J., and Williams, M.-A., editors (2019). *RoboCup 2019: Robot World Cup XXIII*, Lecture Notes in Artificial Intelligence, Berlin, Germany. Springer.

[Guestrin et al., 2002] Guestrin, C., Lagoudakis, M., and Parr, R. (2002). Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234. Citeseer.

[Kok and Vlassis, 2004] Kok, J. R. and Vlassis, N. (2004). Sparse cooperative Q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 61.

[Kok and Vlassis, 2005] Kok, J. R. and Vlassis, N. (2005). Using the max-plus algorithm for multiagent decision making in coordination graphs. In *Robot Soccer World Cup*, pages 1–12. Springer.

[Kok and Vlassis, 2006] Kok, J. R. and Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828.

[Liu and Ihler, 2013] Liu, Q. and Ihler, A. (2013). Variational algorithms for marginal MAP. *The Journal of Machine Learning Research*, 14(1):3165–3200.

[Lourenço et al., 2003] Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer.

[Marinescu and Dechter, 2005] Marinescu, R. and Dechter, R. (2005). AND/OR branch-and-bound for graphical models. In *IJCAI*, pages 224–229.

[Rădulescu et al., 2020] Rădulescu, R., Mannion, P., Roijers, D. M., and Nowé, A. (2020). Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–52.

[Roijers, 2016] Roijers, D. M. (2016). *Multi-Objective Decision-Theoretic Planning*. PhD thesis, University of Amsterdam.

[Roijers et al., 2015a] Roijers, D. M., Whiteson, S., Ihler, A., and Oliehoek, F. A. (2015a). Variational multi-objective coordination. In *NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems*.

[Roijers et al., 2015b] Roijers, D. M., Whiteson, S., and Oliehoek, F. A. (2015b). Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 52:399–443.

[Rosenthal, 1977] Rosenthal, A. (1977). Nonserial dynamic programming is optimal. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 98–105.

[Russell and Norvig, 2005] Russell, S. and Norvig, P. (2005). AI a modern approach. *Learning*, 2(3):4.

[Scharpff et al., 2016] Scharpff, J., Roijers, D., Oliehoek, F., Spaan, M., and de Weerdt, M. (2016). Solving transition-independent multi-agent mdps with sparse interactions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

[Scharpff, 2020] Scharpff, J. C. D. (2020). *Collective Decision Making through Self-regulation: Mechanisms and Algorithms for Self-regulation in Decision-Theoretic Planning*. PhD thesis, Delft University of Technology.

[Verstraeten, 2021] Verstraeten, T. (2021). *A Multi-Agent Reinforcement Learning Approach to Wind Farm Control*. PhD thesis, Vrije Universiteit Brussel.

[Verstraeten et al., 2020] Verstraeten, T., Bargiacchi, E., Libin, P. J., Helsen, J., Roijers, D. M., and Nowé, A. (2020). Multi-agent Thompson sampling for bandit applications with sparse neighbourhood structures. *Scientific reports*, 10(1):1–13.

[Verstraeten et al., 2021] Verstraeten, T., Daems, P.-J., Bargiacchi, E., Roijers, D. M., Libin, P. J., and Helsen, J. (2021). Scalable optimization for wind farm control using coordination graphs. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1362–1370.

[Visser et al., 2014] Visser, A., Ito, N., and Kleiner, A. (2014). Robocup rescue simulation innovation strategy. In *Robot Soccer World Cup*, pages 661–672. Springer.