Evolutionary On-line Learning of Cooperative Behavior with Situation-Action-Pairs

Jörg Denzinger and Michael Kordt
Fachbereich Informatik, Universität Kaiserslautern, Germany
Email: {denzinge,kordt}@informatik.uni-kl.de

Abstract

We present a concept to use off-line learning approaches to achieve on-line learning of cooperative behavior of agents and instantiate this concept for evolutionary learning with agents based on prototype situation-action-pairs and the nearest-neighbor rule. For such an agent model also modeling of other agents can be achieved using the agent's own architecture with situation-action-pairs derived from observations. We tested our on-line learning agents for different variants of the pursuit game and characterize the aspects of variants for which our on-line learning agents outperform off-line learning ones. Since our concept also allows a smooth transition from off-line learning to on-line learning and vice versa, the resulting system is able to win much more game variants than systems using either on- or off-line learning exclusively.

1 Introduction

The use of learning techniques offers a solution to one of the main problems of many multi-agent systems, namely how to achieve good *cooperation* between the different agents. In general, we can divide the known learning approaches used in multi-agent systems into *off-line* and *on-line* learning approaches. While off-line learning approaches are based on the idea to separate the learning phase and the phase in which the multi-agent system really is used, on-line learning approaches try to adapt the multi-agent system while its agents are working on solving the given problem.

The big advantage of off-line learning approaches is that during execution of the learned cooperation concept for the agents no overhead for learning is necessary. In fact, the learning components in off-line learning can be mostly separated from the execution components of the agents. The advantage of on-line learning approaches is that they can cope with changing and/or difficult environments. But to do so the learning techniques have to be integrated into the agents. In addition, usually some modeling of the other agents of the system is necessary in order to achieve the intended adaptation of the agent to the given problem and its environment and consequently the cooperative behavior of the agent.

In this paper we present an approach to on-line learning that is based on off-line learning techniques, namely an evolutionary approach using so-called prototypical situation-action-pairs (SApairs) and the nearest-neighbor rule (NN rule) for decision making (see [DF96]). This agent architecture will be used both for a learning agent and to model other agents within it.

The general idea of our approach is to add to the possible actions of an agent also the action "learn". When this action is invoked the agent uses the off-line learning technique for a certain time period and then integrates the results of this learning into its own decision making. The off-line learning method uses a genetic algorithm to evolve sets of SApairs, what we call *strategies*. The fitness of such a set of SApairs is determined by simulating the behavior of the multi-agent system with the agent using this strategy. In our on-line learning approach the other agents are modeled using also SApairs and the NN rule, and their SApairs are obtained by observing their actions in the time before the learning takes place. If communication between the agents is possible then the other agents also can transfer their actual SApairs (if they use the same agent model) to the agent in order to give it more accurate information about their particular strategy.

Note that using the evolutionary learning approach for on-line learning a strategy for only one agent results in a smaller search space (than for the off-line learning approach) and therefore the possibility to learn more difficult strategies. Also, as in case of all on-line learning approaches, no complete strategies have to be learned but only parts of strategies that are good for the next few steps. This allows for the adaptation to the problem and to changing environments.

In this paper we will test our approach on variants of the pursuit game (see [Be+85]). There are many interesting questions with regards to the parameters of such a learn-

ing agent and its environment. We will concentrate here on comparing the pure off-line learning approach with our on-line approach. We will see that for certain game variants the on-line learning approach clearly outperforms the off-line variant both in success-ratio (if random factors are part of the game variant) and effort needed for learning (measured both in computing time and number of strategies generated).

2 Evolutionary on-line learning with SApairs

In the following we will firstly present our agent architecture and then recapitulate the off-line learning approach we will use. Then we will concentrate on our on-line learning approach, presenting firstly the general working cycle including learning and then the handling of the observations of other agents which also is some kind of learning.

In general, an agent can be seen as a triple (Sit,Act,Dat) of a set Sit of situations, a set Act of actions, and a set Dat of values of internal data areas. The agent uses its perception of the actual situation and the values of the internal data areas to decide upon the next action. Therefore an agent $\mathcal{A}g$ can be seen as a function $f_{\mathcal{A}g}$: $Sit \times Dat \rightarrow Act$. Note that from outside the agent looks like another function, namely a function from the set of situations into the set of actions. This can cause problems if other agents have to reason about an agent based solely on their observations of its actions.

Usually, for an agent in a multi-agent system we can divide the sets into parts concerning the agent itself and parts concerning the other agents. More formally, an element s of Sit has the form $s = s_{Env}\, s_{Ag}$, where s_{Env} describes the environment the agent acts in without the other agents and s_{Ag} provides the information about other agents. Act can be divided into the sets Act_{own} of the agent's actions not dealing with other agents and Act_{co} of its communication and cooperation actions. The internal data areas of the agent can belong to three different sets, namely the set Dat_{own} of values of data areas concerning information about the agent *itself*, the set Dat_{sag} of values of data areas concerning *sure* knowledge about other agents, and the set Dat_{usag} for *unsure* knowledge about the other agents.

The architecture we will use for our agents is rather simple. The data areas (Dat_{own}) of an agent hold a set of so-called *situation-action-pairs* (i.e. $Dat_{own} = 2^{Sit \times Act}$), and its decision function $f_{\mathcal{A}g}$ uses the nearest-neighbor rule to determine the action to the actual situation. More precisely, for all SApairs (s_i,a_i) in the actual value of Dat_{own} the distance of s_i to the actual situation s with respect to a distance measure d is computed (in our case d computes the Euklidean distance of two number vectors). Then the action of the pair with the smallest distance is chosen (i.e. $a = a_j$ with $d(s_j,s)$ is minimal). This results in rather reactive agents.

The off-line learning approach presented in [DF96] uses this architecture as follows. For all agents in the system for which cooperative behavior has to be learned their set of SApairs is evolved using a genetic algorithm (GA, see [Ho92]). An individual of the population of the GA consists of several sets of SApairs, one set for each agent. As usual, we use crossover and mutation as genetic operators. The mutation operator just deletes an arbitrary number of SApairs in a strategy and then adds some randomly generated new SApairs. The crossover operator randomly selects SApairs from both parent individuals and puts them together to form the offspring individual. This selection is done for each strategy in an individual individually, because the agents may differ from each other in their sets Sit and Act.

The fitness computation for the evolutionary learning of cooperative behavior is based on a restricted simulation of the behavior of the agents when applied to the task the multi-agent system has to do. The simulation is restricted because the number of steps that the agents are allowed to do is limited. Naturally, the fitness measure has to be based on the problem to solve and therefore can only be given for a concrete application. We will do this in the next section. As usual, the GA favours fitter individuals for use by the genetic operators.

The problem of this approach is that with complex situation descriptions and larger numbers of agents an individual becomes rather large so that the resulting search space for the GA becomes immense. Nevertheless the approach showed in an application that it can be quite successful for tasks of various difficulty.

Our approach for on-line learning of cooperative behavior makes use of the off-line learning approach by adding an action "learn" to the set of possible actions Act. When "learn" is executed, the agent individually performs off-line learning (of its own strategy only) as described above but with a rather limited number of generations or a limited number of applications of genetic operators (depending on the concrete GA used). Usually, the number of steps in the simulations that are the basis for measuring the fitness of individuals is much smaller than in the pure off-line learning approach.

In order to be able to apply the off-line learning method the agent needs *models of the other agents* so that the cooperative behavior of itself (and the other agents) can be measured by the fitness obtained by the simulation. Naturally, this learning process takes time and therefore the other agents of the system will have done some steps towards doing the given task when the agent has finished executing the action "learn". Then the learned strategy and the strategy used by the agent so far have to be combined.

As already stated, a crucial part for the success of our on-line learning method are the models for the other agents in the system that an agent uses in the simulations. There

are various possibilities how this modeling can be achieved. The easiest way is to get the models directly from the other agents. This means that the models would be stored in the data areas with sure knowledge about the other agents and the simulations will be as accurate as they can be. The other possibilities are based on the agent observing the other agents for which it needs models. Building models out of observations is a learning task, again.

Observations already provide an agent with SApairs, so that using these pairs and the NN rule as agent model is a good idea, again. Note however that such models represent unsure knowledge about the other agents. As already stated, an agent also uses the values of its internal data areas for its decision making, and these values cannot be observed. Hence, we cannot be sure that later the agent will perform in the same situation the same action. Also, other agents might change their behavior. As a result, several different SApairs with the same situation might exist in the model for another agent and the number of pairs can become large. Therefore procedures for compacting the set and resolving conflicting information might be needed.

The scenario of our on-line learning approach can also be seen as some kind of planning during the learning phase of an agent. This learning results in a strategy, which can also be called a *plan*, of how to proceed in the next steps. So again, the close connection of learning and planning in multi-agent systems becomes obvious. Learning approaches for agents are often analyzed with respect to the two terms *exploration* and *exploitation*. Obviously solving the given task (without the execution of the action "learn") is pure exploitation of learned knowledge. The idea of the learning action is exploration of the possibilities. And, as it is well known, genetic algorithms are rather good at this.

3 Pursuit games and the OLEMAS system

The pursuit game was originally presented in [Be+85]. In this original pursuit game four so-called hunter agents had to immobilize a prey agent on an infinite grid starting from random positions. All the agents could only perform four moves (forward, backward, left, right) and had the same speed, one square of the grid per time unit (and move).

Many variants of this pursuit game have been used in literature. In [DF96] the variants have been classified according to several features, namely

- the agents themselves,
- the start position, and
- the goal of the hunt.

Especially for the agents themselves there are several subfeatures than can be varied, as for example their form, their moves, their speed, and, in case of the prey agents, their strategies. From a multi-agent perspective the goal is to find cooperative strategies for the hunter agents to reach the goal of the hunt. Naturally, there is not one single strategy for all the variants of the game that achieves this multi-agent goal. In fact, there are variants for which there is no solution to this problem (i.e. the prey will always escape). But even for the solvable variants the strategies for the hunter agents that represent solutions differ very much from variant to variant.

While the original pursuit game and many slight variations of this original definitely have to be classified as "toy examples", varying features like form and speed of the agents and adding various obstacles to the world not only results in more difficulties to find good cooperative strategies for the agents. Such game variants also describe on a high level problems that teams of robots have to face (as coming near places they have work to do at, finding good relative positions around a working place, and so on). Therefore pursuit games are a very good testbed for learning approaches for multi-agent systems.

Our OLEMAS system (**O**n-**L**ine **E**volution of **M**ulti-**A**gent **S**ystems) is intended to use this testbed. It contains a component to simulate many variants of the pursuit game and it also has, in addition to components for (learning) agents and for evolutionary learning, a component to visualize game runs. OLEMAS is implemented in C++ and has a clearly defined interface so that different agents (using different models) can be used both as hunter or prey (and even as innocent bystander that just hinders or helps both hunters and preys). Whenever an agent can move it gets from OLEMAS the actual situation of the game and has to send its next move back to OLEMAS. If the move conflicts with moves of other agents, only one of the agents in conflict is allowed to move while the others have to stay put. After the time units it takes to execute a move (meaning the number of basic time steps that the move is supposed to take) the agent will be asked for its next move again. OLEMAS guards the rules of the particular variants of the pursuit game and detects situations in which the goal of the hunt is achieved. OLEMAS also protocols all moves (and their outcome), and this protocol is accessible by the agents, serving as a basis for observations.

Figure 1 depicts the architecture of OLEMAS and the data flow between the components. The agents component provides classes for different agent models that can be instantiated by setting various parameters (for example, form, color and possible moves of an agent). Although our main interest is in agents using SApairs, in OLEMAS there are also various agents with hardwired strategies to be used as either hunter, prey or obstacle.

The evolutionary learning component of OLEMAS can be used to learn strategies for one, several or all agents, thus enabling both on- and off-line learning. There are the usual

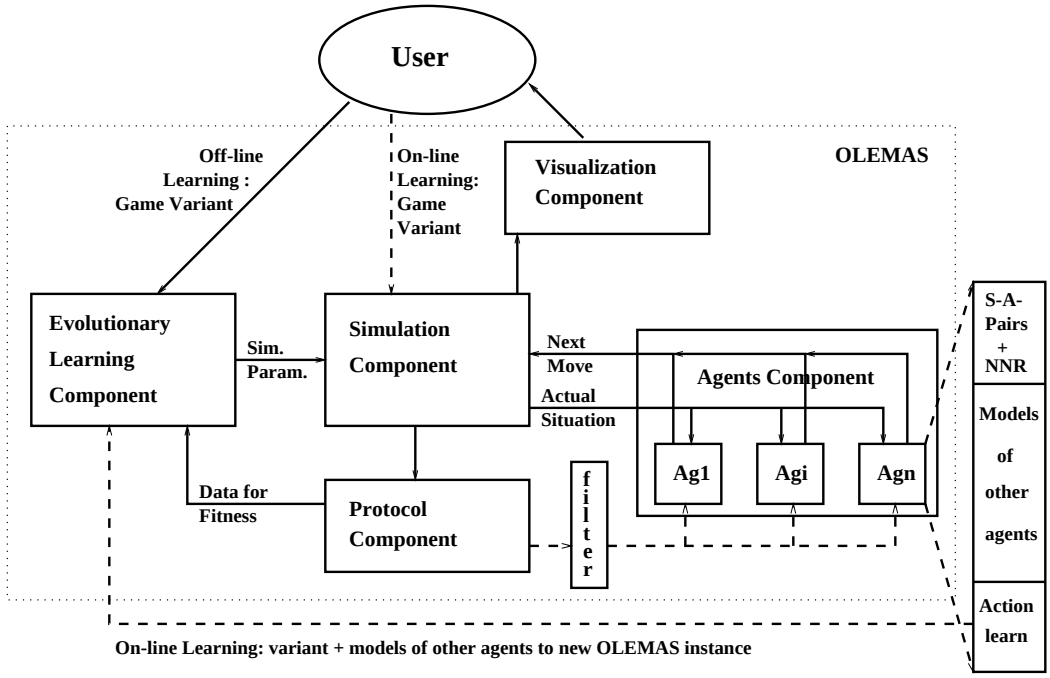

Figure 1. OLEMAS: system architecture

parameters of genetic algorithms (maximal number of individuals in the population, number of generations, basic probabilities used in the genetic operators) that can be set from outside. The learning component also needs a description of the pursuit game variant for which it has to learn, both in order to pass it to the simulation component (together with strategies for the learning agents) and in order to define the possible situation vectors and actions. The situation vector of an agent contains each other agent's coordinates relative to the position of the agent, its orientation (this is important if agents fill more than a square), and its type. The sequence of the other agents in this vector is fixed. Therefore, for measuring the distance between two situations the types of the agents are not important. If a situation j contains the coordinates x_{ij} and y_{ij} of agent i, $1 \leq i \leq$ n, and its orientation o_{ij} (one of 0 = north, 1 = east, 2 = south, 3 = west), then the distance d to situation k is computed as

$$d(j,k) = \sum_{i=1}^{n}((x_{ij}-x_{ik})^2 + (y_{ij}-y_{ik})^2 + ((o_{ij}-o_{ik})^2 \bmod 8)).$$

The contribution of the orientation of the agent has to be taken mod 8 in order to treat different orientations clockwise and anticlockwise the same. Without this modification the difference $o_{ij} - o_{ik}$ between o_{ij} = north and o_{ik} = east would not be the same as the difference between north and west, which is not our intention. Since we use the square of the distance, the results must be modified by mod 8. Finally, there is also a limit on the number of SApairs in the strategy of an agent.

The fitness function used by the learning component assigns to an individual winning the game variant the number of time units needed for the win as fitness. As in [DF96], the fitness of an individual not winning the game variant within the allowed number of time units T_{Max} is generated by summing up the sums of the distances of the hunter agents to the prey for each time unit. More precisely, this elementary fitness *efit* of an individual $\mathcal{I}$ for one simulation run is defined as

$$\mathit{efit}(\mathcal{I}) = \begin{cases} t_s, & \text{success in } t_s \text{ steps} \\ \sum_{t=1}^{T_{Max}} \sum_{i=1}^{n} \delta(i,t), & \text{in case of failure,} \end{cases}$$

where $\delta(i,t)$ is the Manhattan distance that separates hunter i from the prey at time step t (or the sum of the distances to all prey agents if there are more than one) and $t_s \leq T_{Max}$.

If we have to deal with either random effects or unsure data then one simulation run per individual is not sufficient. Thus, the overall fitness *fit* of an individual $\mathcal{I}$ is defined as

$$\mathit{fit}(\mathcal{I}) = \frac{1}{b} \cdot \sum_{i=1}^{b} \mathit{efit}_i(\mathcal{I}),$$

where b is the number of simulation runs, again a parameter of OLEMAS. All in all, the evolutionary learning compo-

nent is just an extension of the one used in [DF96] to handle more types of agents and a broader variety of variants of the pursuit game. To do so, it was necessary to switch to an "egocentric" agent view, i.e. all coordinates in a situation vector are with respect to the agent for which the vector is intended.

When using on-line learning agents in OLEMAS, at least two instances of OLEMAS are needed: the instance that handles the game in which the on-line learning agents have to act (the "*real world*") and for each of these agents its own instance that is used when the action "learn" is executed. The additional data flow in OLEMAS for on-line learning is depicted in Figure 1 by the dashed lines. The description of the game variant that is given to the simulation component contains, besides the usual data, for each on-line learning agent an initial strategy (in form of a set of SApairs).

In Section 2 we already sketched alternatives with respect to some aspects of on-line learning agents. In OLEMAS we tried to cover many alternatives with respect to various design decisions. Therefore, in addition to the game variant and the parameters of the evolutionary learning component, OLEMAS offers many variants of on-line learning that are determined by several additional parameters and their value sets. In the following, we will only describe the alternative(s) we used in our experiments.

The action "learn" is performed by an agent within at least learn_{min} and at most learn_{max} time units after its last execution. Within this interval it is chosen when the variable $lval$ is zero or less. After its initialization with a positive number, after each move $lval$ is incremented by 1, if the distance agent-prey decreased, and decremented by 2, if the distance increased. This achieves a success-oriented handling of the execution of "learn".

In order to perform "learn" a new instance of OLEMAS is generated. The game variant for it is mainly the same as that of the "real world", except for the f_{Ag}-functions of the other agents and the start situation. The start situation is the predicted situation after "learn" has been performed. It is computed by running the new instance of OLEMAS, starting with the actual situation, for the number of time units that the action "learn" is allowed to take (naturally without any actions by the learning agent). The prediction of the start situation as well as the whole simulation runs naturally depend on the actions that the other agents will do. We used two alternatives to get the f_{Ag}-functions of other agents. The first alternative is to use the f_{Ag}-functions of the original OLEMAS instance, which can be seen as the agents communicating their behavior to the learning agent. The other alternative is to use the models of the other agents the learning agent has.

In OLEMAS, learning agents model other agents by prototypical SApairs and the NN rule (as can be seen for agent n in Figure 1). The SApairs are generated out of the observations the agent can make. In principle, the observations are taken from the protocol component, but there is the additional possibility to filter the protocol. Such a filtering can be used to simulate limited observation capabilities (hiding agents, for example, or imprecise observations) and is intended for future use. In addition to filtering, also a compactification of the observed data could be performed. In our experiments we took all the observed SApairs and, in case of several SApairs matching a situation, the action of the first pair in the list of SApairs was chosen (which reflects the most recent observation).

The execution of the action "learn" results in the evolutionary learning component of the new OLEMAS instance generating a fixed number of generations starting from an initial population that was randomly generated in our experiments. The result of learning is the strategy out of all generated individuals with the best fitness, as usual. In our experiments, we deleted the former strategy of the agent and replaced it by the newly learned one.

With the generation of this new strategy the execution of "learn" ends, the Dat_{own}-area of the agent is updated and the new OLEMAS instance is discarded. Then the agent has to act in the "real world" using this new strategy until "learn" has to be performed again.

4 Experimental results

We have performed various experiments with different variants of the pursuit game and both the off-line learning approach and our new on-line learning one. Due to the enormous number of different game variants and the large number of parameters and their values in OLEMAS, many more experiments can be done, but our experiments so far already allow us to give an impression of the strengths and weaknesses of our new approach. In the following, we will present a few selected experiments that highlight what on-line learning can do better than off-line learning.

For all comparisons and experiments both approaches used the described fitness measure. In all game variants horizontal, vertical, and diagonal translations, rotations (if sensible) and staying put were allowed moves, each of them taking one time unit. In the evolutionary learning process the best 20 individuals of a generation were always also part of the next one. The first strategy to win the game (or 9 out of 10 runs, if random factors were included) was selected as new strategy of a learning agent. If no such strategy was evolved then the one with the best fitness was selected. If not stated otherwise, "learn" took 3 time units.

In general, our experiments showed that there are game variants that can only be solved using on-line learning and also variants that can only be solved using off-line learning (or on-line learning with settings of the learning component equivalent to the ones used for off-line learning, plus execu-

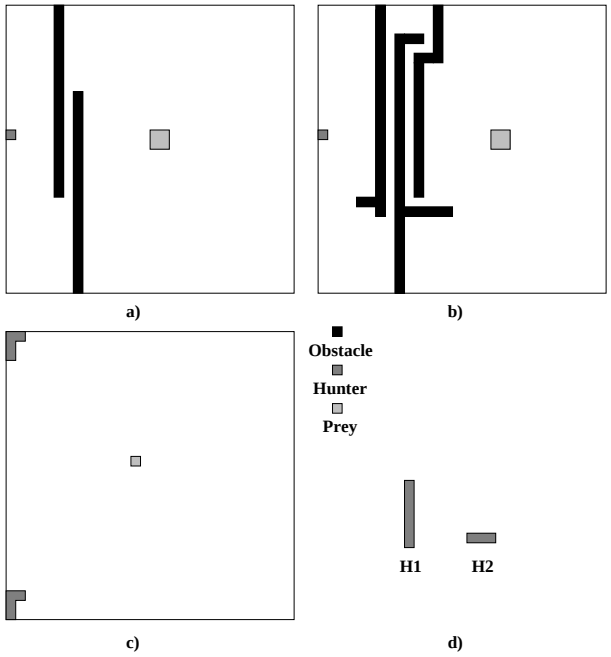

Figure 2. Tested variants (grid size 30 × 30 in all cases)

tion of "learn" within the first few moves). The reasons for the later are well-known in the area of real-time search (see [Ko90]). As in case of on-line learning, in real-time search the number of transitions is limited and after the limit is reached, an action has to be performed. Due to the limit it is possible that an action is chosen whose consequences cannot be negated by other actions afterwards. This might result in not being able to find a solution to the given problem although one exists.

On the other side, if a task to be done can be divided into a sequence of subtasks that are based on each other then the evolutionary off-line learning approach has to evolve prototypical SApairs for each of the subtasks together and without knowing about the existence of the subtasks. The on-line learning approach just can tackle each of the subtasks when they become obvious. In Figure 2 a) and b) we have depicted two variants of the pursuit game that obviously consist of subtasks, namely navigating through the obstacles and then catching the prey (i.e. moving on a square still occupied by it), while the strategy of the prey is to stay away from the hunter as far as possible.

Our on-line learning agent wins variant a) after 165 time units of the game (while needing appr. 4 minutes real time on an UltraSparc IIi). The length of the simulation runs was 13 units. An individual consisted of at most 20 SApairs and we allowed a population size of 100 and 20 generations for

var.	percentage solved		time off-line	mean time
	off-line	on-line	learning	on-line runs
1	40	64	62 min	58 sec
2	42	75	2633 min	358 sec
3	39	57	480 min	15 min

Table 1. Results of 100 games for each variant

the genetic algorithm. The whole game was limited to 300 time units. The off-line learning approach was not able to solve the game with these parameters (but setting the allowed time units in a simulation run to 300). By choosing some rather extreme parameters, for example 100 SApairs per individual and a population size of 5000, the off-line learning approach could solve the game variant using two days on the same machine. But even such extreme parameter settings do not allow the off-line learning hunter to solve the variant b) (that consists of more subtasks), while the on-line learning agent wins it after 249 time units (8 minutes run-time) using the same parameter settings as for variant a).

Another strength of on-line learning agents, we observed, is dealing with game variants including random factors. Although [DF96] demonstrated the ability of the off-line learning approach to cope with such variants, for more complex agents on-line learning achieves a greater percentage of wins. Table 1 compares the performance of both approaches for three game variants.

Variant 1 consists of the hunter H1 depicted in Figure 2 d) using a fixed strategy that tries to come as near as possible to the standard prey that occupies a single square and that itself tries to maximize the distance to the nearest hunter. Together with H1 we use an on-line learning agent occupying a single square. Randomness is introduced by randomly generating the start positions of all agents. Therefore we do not have a picture of the start positions in Figure 2. The goal of the game now and for the following variants is to immobilize the prey. For variants 1 and 2 the simulation runs were limited to 23 time units. The GA was allowed 15 generations of 300 individuals.

In variant 2 we made the task for the learning hunter more difficult by giving it the shape of H2 in Figure 2 d). In variant 3 we have the start situation as depicted in Figure 2 c). The hunter in the above left corner uses the strategy of H1 of the previous two variants, while the other hunter has to learn its strategy. Now the prey makes random moves which leads to a harder to solve game variant. Due to the unpredictability of the prey's moves, for performing "learn" we counted only 2 time units, but to make up for this we limited the simulation runs to only 13 time units. Also we allowed only 12 generations of 100 individuals.

For all three variants, the on-line learning agent clearly outperforms an agent using an off-line learned strategy (see

Table 1). If we are interested in one single run only, then the chance is higher that an on-line learning agent helps the other hunter to achieve success. In addition, the time needed for learning is much less than for off-line learning (the time needed for the "real-world" game for the off-line learned agent is neglectable). While by using an off-line learned strategy the agent has to hope that both certain critical situations occur and the subsequent actions of the other agents follow what occurred during learning, using on-line learning allows for starting from the actual situation and looking at shorter action sequences. This results in the observed better performance.

Although the learning agent in the variants of Table 1 has to learn a strategy that cooperates with the rather limited strategy of the other hunter, learning cooperative behavior often is thought of as involving all the agents, in our case all the hunters. Therefore we repeated the experiments of variant 2 of Table 1 but now let both hunters use on-line learning. This resulted in an improvement to winning 80 per cent of the games with a mean time for a game of 6 minutes. We also tried the off-line learning approach for this game variant, i.e. learning strategies for both hunters, but the system was not able to evolve an acceptable solution within 3 days of run-time.

So far, in all experiments the on-line learning agents used sure knowledge about the strategies of the other agents in the simulation runs while performing the action "learn". Naturally, if the prey makes random moves (as was the case for variant 3 of Table 1) this knowledge does not help much, since the moves in the simulation runs are likely to be different from the ones that the prey will do in the "real world". For such agents also modeling them using observations will not help a hunter agent much.

In our next series of experiments we used variant 2 from Table 1 with two learning hunters (as described above) and let them model the prey agent using all SApairs from their observations and the NN rule. So, the learning with respect to the behavior of the prey was learning by heart. As can be expected, the very unsure knowledge, to which this learning and modeling led, resulted in a worse percentage of won game runs, namely only 41 per cent. The mean time needed for a game run was 20 minutes, a result of the rather large SApair sets (for the prey) out of which the nearest situation had to be found. It should be noted that the resulting percentage of wins is nearly as good as the percentage of the off-line learning agent for variant 2 and better than the result of off-line learning for both hunters (which both did not model the prey but used its real strategy during the simulation runs). Note also that we did not allow the hunters more time units for winning the game than in these other variants.

Since the success of learning should improve with additional experience, our final series of experiments repeated the last experiment but now doubling the allowed number of time units. Then the agents were able to win 72 per cent of the game runs in a mean time of 38 minutes.

While these results indicate that using SApairs and the NN rule for modeling other agents when performing on-line learning can achieve some success, there is much room for improvements. Future work must be directed towards finding good filters (as indicated in Figure 1) and using some analysis of the observed behavior (resulting in better learned models).

Our results show that on-line learning allows OLEMAS to deal much better with more complex (game) tasks and games involving random factors. The price to pay are longer game run-times, because now learning is not separated from the runs anymore (a run using off-line learned strategies takes less then a second). But the differences in learning time between on- and off-line learning is very large so that even for several runs the combined times of the on-line learning runs are smaller than the off-line learning time.

5 Related work

The most often used concept for on-line learning is reinforcement learning (see [Wa89]). Its general idea is to associate with each action for each situation a weight describing its usefulness. Rewards of chains of actions are propagated back through the chains to adjust these weights. In order to achieve an exploration of the possibilities, not always the action with the best weight is chosen. Concepts like the bucket-brigade (see [We95]) try to catch the idea of reinforcement learning on the multi-agent level.

Our approach using SApairs does not use weighting of actions for situations and adjusting weights with experience over time. Instead, we have a clearly defined exploration phase and as a result exactly one action for a situation. Changing this action is the task of later executions of action "learn". A big advantage of our approach is that we do not need to store the weights for each action and each situation, a matrix that is usually very large. We even do not have to store one action for each situation, since we learn a set of prototypical situations.

In the domain of pursuit games, most of the published learning approaches are off-line ones (see [DF96] for an overview). Also, these approaches have only been tested for slight modifications of the original game (no obstacles, no other forms of agents than a single dot). The approaches mainly differ in the agent architectures used. In general, evolutionary methods to implement off-line learning are rather often used. In [MC93] a GA evolved good parameter settings of a given agent control program, while in [Ha+95] genetic programming was used to find good control programs.

An on-line learning approach based on reinforcement learning for the pursuit game was first presented in [Ta93].

In [OF96] a modular reinforcement approach was evaluated for a slight modification of the game. An agent used several learning modules, each of them focusing on only a part of a situation vector, namely the part for one other agent and the prey. So, also some modeling of the behavior of the other agents was achieved, but the model was not explicit and not separated from the on-line learning of the agents.

Modeling other agents has been of quite some interest in the last years. Section 2 of [Ga+98] gives a good overview of the known approaches to model other agents, ranging from using BDI formulae as models to game- and decision-theoretic approaches centered around payoff matrixes and functions and needing rationality of the agents as premise. [Ga+98] itself is also concerned with learning models for other agents from observations. Instead of constructing a strategy, as in our case, an agent is modeled as a probability vector for a fixed set of given strategies. The entries in the vector describe the probability that the other agent follows the particular strategy, and incrementally updating the vector is done using the observations.

6 Conclusion and future work

We presented a concept for on-line learning of cooperative behavior of agents that is based on using off-line learning techniques while performing a special action "learn". We instantiated this concept for evolutionary learning of prototypical situation-action-pairs and demonstrated in experiments that the on-line learning approach allows to solve variants of the pursuit game that the pure off-line learning approach is not able to deal with. For other variants, a higher success rate and much less time spent for learning was achieved.

But there are also game variants for which off-line learning achieves better results. Since our on-line learning concept is based on off-line learning, by incrementing the look-ahead, represented by the allowed length of the simulation runs during learning, it is possible to achieve a smooth transition towards off-line learning. Therefore a system based on our concept can use the strengths of both on- and off-line learning.

So far, we have conducted a small part of the experiments that are possible and interesting with regards to the various parameters and alternatives for on-line learning implemented in OLEMAS. The length of simulation runs, the maximal number of SApairs in a strategy, and the interval between executions of "learn" obviously have quite some impact on the performance of the learning agents. Also the influence of the time units spent in the "real world" game is of interest. With regards to having several on-line learning agents and agents modeling other agents using observations our first experiments have been very promising, but more series of experiments should be made to evaluate the potential of these abilities of OLEMAS.

References

[Be+85] Benda, M.; Jagannathan, V.; Dodhiawalla, R.: An Optimal Cooperation of Knowledge Sources, Technical Report BCS-G201e-28, Boeing AI Center, 1985

[DF96] Denzinger, J.; Fuchs, M.: Experiments in Learning Prototypical Situations for Variants of the Pursuit Game, Proc. ICMAS'96, Kyoto, 1996, pp. 48–55.

[Ga+98] Garrido, L.; Brena, R.; Sycara, K.: Towards Modeling Other Agents: A Simulation-Based Study, Proc. MABS'98, Paris, Springer, 1998, pp. 210–225.

[Ha+95] Haynes, T.; Wainwright, R.; Sen, S.; Schoenefeld, D.: Strongly typed genetic programming in evolving cooperation strategies, Proc. 6th International Conference on Genetic Algorithms, Morgan Kaufmann, 1995, pp. 271–278.

[Ho92] Holland, J.H.: Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, Ann Arbor: Univ. of Michigan Press, 2^{nd} edition, 1992.

[Ko90] Korf, R.E.: Real-Time Heuristic Search, AI 42, 1990, pp. 189–211.

[MC93] Manela, M. ; Campbell, J.A.: Designing good pursuit problems as testbeds for distributed AI: a novel application of genetic algorithms, Proc. 5th MAAMAW, Neuchatel, 1993, pp. 231–252

[OF96] Ono, N.; Fukumoto, K.: Multi-agent Reinforcement Learning: A Modular Approach, Proc. ICMAS'96, Kyoto, 1996, pp. 252–258.

[Ta93] Tan, M.: Multi-agent reinforcement learning: Independent vs cooperative agents, Proc. 10th Machine Learning, Morgan Kaufmann, 1993, pp. 330–337.

[Wa89] Watkins, C.J.C.H.: Learning from Delayed Rewards, PhD thesis, University of Cambridge, 1989.

[We95] Weiß, G.: Distributed Machine Learning, Infix-Verlag, Sankt Augustin, 1995.