
On the influence of learning time on evolutionary online learning of cooperative behavior

Jörg Denzinger
Computer Science Department
University of Calgary, Canada

Michael Kordt
Fachbereich Informatik
Universität Kaiserslautern, Germany

Abstract

We present an online learning approach for learning cooperative behavior in multi-agent systems based on invoking an offline learning method as a special action “learn”. We apply this approach to evolutionary offline learning using situation-action-pairs and the nearest-neighbor rule as agent architecture. For the application Pursuit Games we show that the online approach using evolutionary offline learning allows for good success rates for rather different game variants. Particularly, we perform experiments highlighting the influence of the time needed for learning and of the parameters of the evolutionary offline method.

Our results show that even a duration of “learn” which is several times longer than the usual duration of an agent’s actions still achieves good success rates. The same applies to rather small values for the key parameters of the offline method. Together, this suggests that this evolutionary online learning approach is a very good alternative to the well-known online approaches based on reinforcement learning.

1 Introduction

Achieving cooperative behavior of a group of agents either requires a careful planning, design and implementation of the agents by human developers or can be tackled by using learning techniques. Learning can either be used in an offline manner, which means that the learning phase is separated from (i.e. before) the real application phase of the agents, or it can be online, in which case the agents try to adapt their be-

havior during their work on the real application. Evolutionary approaches have already proven to be quite successful for offline learning of cooperative behavior of agents, as has been demonstrated in Manela and Campbell (1993), Hayes et al. (1995) or Denzinger and Fuchs (1996). In these approaches the behavioral patterns of all cooperating agents were evolved together and the resulting agents were not capable of learning for themselves, they just consisted of the application specific strategies that then later were applied to solve the problem. The cited papers show that even a certain flexibility of these strategies, to deal with random effects, is achievable.

For online learning of agents, not many concepts involving evolutionary methods have been proposed so far. The dominating approach has been to employ reinforcement learning techniques, i.e. associating with each possible action in each possible situation a certain weight describing the usefulness of the action in the situation (see Watkins (1989), Weiß (1995) or Tan (1993)). As in case of evolutionary algorithms, some random decisions are needed in order to realize the necessary exploration of alternatives during solving the given task. From the point of view of online learning, at first glance the use of evolutionary methods with the need to evaluate a lot of different individuals that are the result of the evolutionary process seems to promise performance problems, because many individuals will not represent solutions and trying to evaluate their performance might gravely endanger the solving of the given task (that has to be interleaved with the learning process in order to be an online approach). We will see that this does not have to be the case.

In this paper we will use the offline learning approach presented in Denzinger and Fuchs (1996) together with the idea of an action “learn” and modeling of other agents to get an online learning approach for cooperative behavior of agents. An online learning agent will monitor and remember all occurring situations and the

actions the other agents performed in these situations in order to construct models of these other agents. It will also periodically perform the action “learn” that results in the agent invoking the offline learning approach. This offline learning approach evaluates the fitness of agent strategies based on a simulation of the real application and the success of the particular agent strategy in this simulation. For online learning, the simulation starts from the situation the learning agent anticipates to be in after learning and uses the models of the other agents to predict their behavior.

We applied this online learning approach to variants of the well-known Pursuit Game and showed that the approach is capable to solve instances that are not solvable by offline learning (see Denzinger and Kordt, 2000). But before we can apply the approach to more complex application areas, a more precise study of the influence of the time needed for performing “learn” relative to the execution time of “standard” actions is needed. And then some experimental evaluation is necessary to determine if the parameter values of the offline learning approach – like number of individuals and generations, or length of the simulation – that are needed to achieve sensible execution times for “learn” still result in a sufficient learning quality.

The experiments we present in this paper show that indeed the relation between the time needed for performing “learn” and the execution time of other actions plays an important role for success, and too much time needed for “learn” will lead to too much uncertainty so that learning is not sufficiently successful. On the other side, we can show that already relatively small values for the parameters mentioned above achieve fine learning results, so that the presented approach to evolutionary online learning is a good, even superior alternative to the non-evolutionary online learning approaches.

2 Online Learning by Offline Learning

Offline learning of cooperative behavior of a group of agents is intended to produce *strategies* for all or some of the agents involved in performing the task to achieve a certain common goal. Offline learning takes place before the agents actually perform this task and usually involves the exploration of many possible strategies, their evaluation and then their optimization. During the exploitation of these learned strategies, i.e. the actual tackling of the task, no more learning takes place. So, we can see offline learning as a kind of selection process to find the strategies for the individual agents that, if applied together, achieve the intended common goal. In this sense, the resulting agents themselves do

not have any learning capabilities.

In contrast, *online learning* agents are in the process of performing tasks towards achieving a certain goal. Therefore learning has to be integrated into these tasks and combined with the actions that are taken. As a consequence, timing of actions and learning becomes a very crucial issue, because there can be actions that are not reversible. And if later learning leads to the knowledge that such an action should not be performed in a certain situation, the goal might already be not reachable anymore. So, the balance between exploration and exploitation of the learned knowledge is very crucial and online learning of cooperative behavior of agents is a very difficult task (if the problem to be solved is not very simple).

2.1 Our General Method

In general, an agent can be described as a function f_{Ag} that maps situations and the internal state of the agent to an action (or action sequence) that the agent is capable to perform. The internal state of an agent contains all its knowledge about the environment, other agents and itself. This can include goals, plans or observations. Note that from outside the internal state usually cannot be observed, so that an observer sees an agent only as a function $f_{Ag,Obs}$ mapping situations to actions. Also actions might include components that manipulate the internal state of an agent, and these components also cannot be observed from outside.

The result of offline learning is such a function f_{Ag_i} for each agent Ag_i that was included into the learning process. In contrast, the function f_{Ag} of an online learning agent contains a component that is responsible for learning (and such a component cannot be identified in the f_{Ag_i} resulting from offline learning). Our idea is to realize this component by invoking the offline learning approach and to trigger the use of the component by introducing a new action “learn”. The offline learning approach is based on a simulation of the real environment the agent is acting in, its most important part being the simulations (or *models*) of the other agents. These models are either derived from the agent’s observations of their behavior or from information communicated by the other agents to the online learning agent.

More precisely, an online learning agent performs the following cycle:

1. acting in the environment (*real world*) according to its current strategy for a given amount of time or until its success is obviously not good enough
2. performing the action “learn”:

- (a) generate/update the models of the other agents
- (b) determine the situation s_{after} after performing “learn”
- (c) run offline learning approach with s_{after} as start situation and the models of the other agents
- (d) combine best strategy found by offline learning with current strategy to get a new one

There are several ways to realize each of these steps, even for a given agent architecture and a given offline learning approach. Problems to be solved are, for example, how to determine success during the application of the current strategy in the real world, or how to generate the models for the other agents. There are also many possibilities for generating a new strategy, from simply replacing the old strategy with the learned one over somehow putting old and learned strategy together to just keeping at the old strategy (if the learned one does not seem good enough).

In addition to the possibilities and parameters of this general online learning method, there are also the parameters of the offline learning approach that influence the online learning. Obviously, the time needed for offline learning will also influence online learning, because this time must be known before performing offline learning in order to determine s_{after} . Therefore the offline learning method should be realized as any-time algorithm (see Boddy and Dean, 1988). And among the parameters of the offline learning method should be the length (in time units) of the simulations used by it.

2.2 Evolutionary Learning with SAPairs

If we look at the requirements for the offline learning method in 2.1, then evolutionary algorithms in general already provide the any-time property. The other requirements are met by the approach presented in Denzinger and Fuchs (1996) that is based on prototypical situation-action-pairs (SAPs) and the nearest-neighbor rule (NNR) as agent architecture and a genetic algorithm with sets as individuals and evaluation of simulation runs as fitness measure.

More precisely, an agent consists of a function f_{A_g} that is based on a set of SAPs. A situation is represented by a vector of numerical values that describe the environment the agent is acting in (including information about the other agents) and also the necessary information about the internal state of the agent. This means that a situation in a SAP may contain more information than the situations mentioned in 2.1. The

actions in a SAP are the actions that the agent is able to perform (without the action “learn”) or sequences thereof.

The agent determines its next action by measuring the distance between the actual situation and the situations of all its SAPs. A possible distance measure is the Euclidean distance, but also other measures can be used. Then it performs the action of the SAP with the smallest distance. This is the well-known nearest-neighbor rule.

This agent architecture is very well suited for learning, because the strategy of an agent can be easily changed by adding or deleting SAPs or just changing components of an SAP. From the point of view of evolutionary algorithms, the agent architecture also has many advantages, as we will see. In Denzinger and Fuchs (1996), a genetic algorithm was used to evolve sets of agent strategies, i.e. a strategy (which is a set of SAPs) for each agent that needed to learn one. These strategies, when applied together, should achieve a good cooperative behavior of the agents. The used genetic operators were picking random SAPs out of the strategy for one agent of two individuals as crossover and generating a random SAP or deleting a SAP out of a set as mutation.

For determining the fitness of an individual (which describes a team of agents), the agents are put to solving their task (in a limited simulation). If the agents are able to succeed, the fitness is the number of time units needed. If the task is not accomplished then for each time unit t of the simulation a measure of task fulfillment $m_{task}(t)$ is computed and these measures are summed up. If there are random factors involved in the simulations then the mean fitness value over several different simulation runs is taken as the fitness of an individual. Important parameters of this offline learning method are the maximal number sap_{max} of SAPs of an agent, the maximal number T_{max} of time units in a simulation, the number b of simulation runs that go into determining the fitness of an individual and, as usual for a genetic algorithm, the number of individuals in a population and the maximal number of generations.

Using this offline learning approach for online learning, we have to learn a strategy for one agent only. Also, the length of a simulation run can be much shorter than in the offline case (as we will see in our experiments). We might also use a lower sap_{max} value. The measures of task fulfillment $m_{task}(t)$ can be used to determine the success of a strategy in the real world, by assuming that the strategy is successful as long as $m_{task}(t_i)$ gets better from some t_i to some t_{i+k} . Com-

binning strategies realized by SAPs is rather easy by just combining the sets. Finally, SAPs can also be easily generated out of an agent’s observations of the other agents. Therefore SAPs (together with NNR) provide a very good way to model the other agents.

3 The OLEMAS System

The OLEMAS system (OnLine Evolution of Multi-Agent Systems) instantiates the methods described in Section 2 for the application area Pursuit Games. In the following, we will first take a closer look on this application area and then we will describe OLEMAS in more detail.

3.1 Pursuit Games

The original Pursuit Game was described in Benda et al. (1985) as four dot-shaped hunter agents trying to surround and immobilize a dot-shaped prey agent on an infinite grid world without any obstacles. All agents could only perform movements in the horizontal and vertical directions at the same speed (one square per time unit). Every agent could see all the other agents and this was the only kind of “communication” between the agents. The goal of the game was to develop a set of hunter strategies to win the game against a prey choosing its moves randomly.

While this original game is considered a toy problem these days (although a difficult one), many variants have been developed that vary numerous features of the game. A large collection of these features and possible instantiations can be found in Denzinger and Fuchs (1996). We implemented the possibility to vary all the features mentioned there in OLEMAS and added some additional possible instances. For example, we do not only have fixed obstacles, but also agents that play the role of innocent bystanders (or moving obstacles).

Among the features that raise Pursuit Games from simple toy problems to abstractions of real world high-level robot control problems are the shape, speed, and possible actions of an agent. By letting agents occupy several squares of the grid representing the world we introduce additional actions, namely turns, and the difficulty of not only being in a certain position but also needing a certain orientation. Assigning to each action of each agent an execution time (as multiples of a basic time unit) also accounts for more reality and often more difficulties in achieving the common goal. The same applies to having the bystanders that accidentally can help or hinder both hunters and prey (and that allow for game variants in which predicting

their behavior is essential for winning the game). Naturally, bystanders and prey have their own strategies and the different possibilities here already produce an infinite number of variants. Other features include the number of agents of all types, the start situation and the definition of what is a winning situation.

3.2 The System

OLEMAS is implemented in C++ and uses Tcl/Tk and Tix for visualizing the pursuit games. The system architecture is depicted in Figure 1. Basis for the agent architecture are the situation vectors and the distance measure. A situation vector of an agent contains each other agent’s coordinates relative to the agent’s position (in a fixed order) and its orientation. If a situation s_j contains the coordinates x_{ij} and y_{ij} of agent i , $1 \leq i \leq n$, and its orientation o_{ij} (0 = north, 1 = east, 2 = south, 3 = west), then the distance d to situation s_k is computed as

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

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 $o_{ij} = \text{north}$ and $o_{ik} = \text{east}$ 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.

For the Evolutionary Learning Component, we have to define the fitness of an individual, more precisely, we have to define the function m_{task} :

$$m_{task}(t) = \sum_{i=1}^n \delta(i, t),$$

where $\delta(i, t)$ is the Manhattan distance that separates hunter i from the prey at time unit t (or the sum of the distances to all prey agents if there are more than one).

OLEMAS is intended as experimental system to evaluate all the possible influences on online-learning by offline learning. Therefore we tried to allow for changing as many parameters and components as possible. In addition to the learning time aspect that will be the subject of the experiments in the next section and the other parameters already mentioned, we allowed for the easy addition of other agent architectures and strategies, for filtering of all the information accumulated during a run (influencing the modeling of

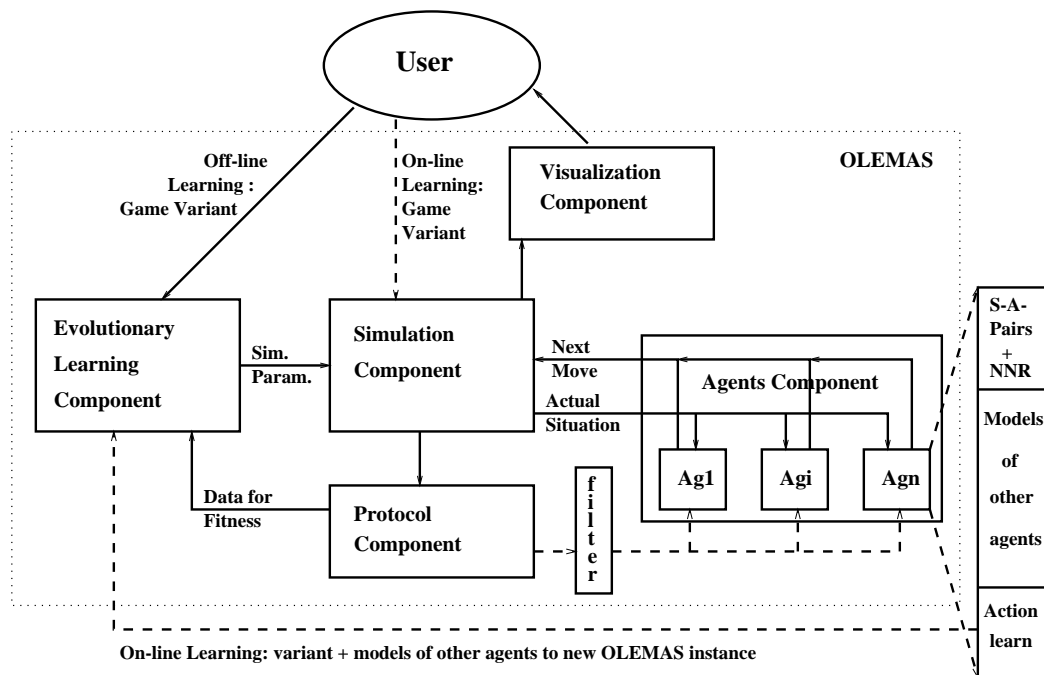


Figure 1: OLEMAS: system architecture

other agents) and even for changes in the Evolutionary Learning Component.

When having online learning agents, the user defines a game variant for the Simulation Component (the “real world”). With this variant, also the online learning agents are identified and an initial strategy for each of them is given (either predetermined, totally at random, or the result of performing “learn” as first action). Then the agents act in the real world until the first online learning agent has to “learn”. This is either after a fixed number of time units or in between a minimum and maximum number of time units, determined by the performance (see Denzinger and Kordt, 2000, for details). The learning agent then generates SAP-based models for all other agents for which it does not have more accurate information (i.e. for all agents that do not communicate their real strategy to it). To do so, it generates a SAP for each situation observed so far from the perspective of the other agent (using the protocol generated by the Protocol Component; filtering will be used in future experiments). Then the models, the current situation and the other parts of the variant being the real world are passed as a game variant to a new instance of OLEMAS, more precisely to the Evolutionary Learning Component of it. There, first the simulation is started to determine the situation at the end of executing “learn” and then offline learning takes place. The best strategy found will then be the new strategy for the online learning agent.

4 Experiments

While the experiments described in Denzinger and Kordt (2000) concentrated on the suitability of the general method for online learning of cooperative behavior and established the advantages of this method, we want to concentrate in this paper on the suitability of evolutionary algorithms as basic underlying learning, resp. search, method. Since “learn” is an action, a learning agent performing it obviously cannot (and should not) do anything else, which kind of takes this agent out of the game during learning. So, a first important question is how much time an agent can spend on learning while still being able to cooperate with the other agents in achieving the goal of the game. Obviously, here the relation of (game) time units spent on learning and units necessary for the other actions is of interest. OLEMAS allows us to define the length of learning with respect to the real world game freely and independent from the computing time needed. We examine this relation in our first experimental series.

Although OLEMAS allows us to have two times for learning (game time and processing time needed), for later applications it is very important to know if the processing time can be limited sufficiently, so that, for example, a robot can have an action “learn” while acting in the real world. This leads to investigating the parameter values that mainly influence how much processing time is needed to perform “learn”, namely the maximal number sap_{max} of SAPs in an individual, the number of individuals in a generation, the number of

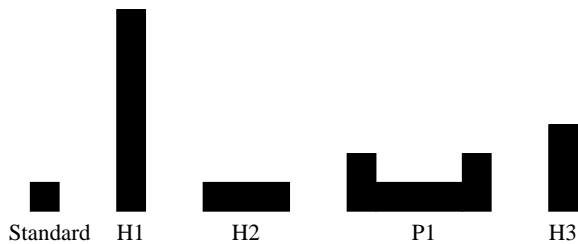


Figure 2: Shapes of the used agents

generations, and the maximal number T_{max} of time units to be simulated for determining the fitness of an individual. We examine the influence of these parameters in the experiments of the second series.

4.1 The game variants

The three game variants we selected for our experiments require rather different strategies from the hunters to be successful, and in all variants random factors (in the form of random start positions for the agents) are involved, so that two runs of OLEMAS with the same variant usually differ very much. In all variants we have only one online learning hunter, because we are interested in the learning time in general (for multiple learning agents, see Denzinger and Kordt, 2000). One single experiment for a variant always consisted of 100 real world runs and as success we will report the percentage of successful runs within the given time limit. In all variants, the time needed for any action except “learn” was one time unit and the possible moves of the agents are up, down, left, right, stay put, and, if the shape makes turns useful, turn left and turn right (for 90 degrees around the agent’s predefined center point). The grid size was 30×30 squares for all variants and there are no obstacle agents.

In variant 1 we have two hunters and one prey. The prey tries to maximize the distance to the nearest hunter and it occupies one square only. The first hunter has the shape of H1 in Figure 2 and uses a fixed strategy, namely trying to come as close to the prey as possible. The learning hunter has the form of H2 in Figure 2. The goal situation of this variant is to immobilize the prey.

In variant 2 we have 4 hunters and 4 preys, all of them occupying one square only. The prey agents use the same strategy as the prey in variant 1. While three of the hunters use the strategy of the non-learning hunter in variant 1 (forming a pride), the fourth hunter is a learning one. The goal situation is that each hunter occupies the same square as a prey (no two prey agents are allowed to occupy the same square), and whenever

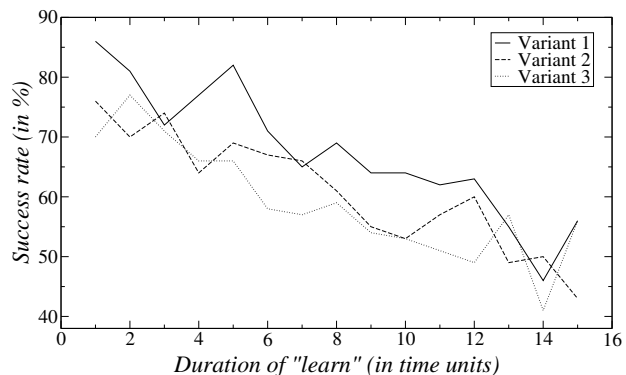


Figure 3: Influence of duration of “learn” on success

a prey is caught in this sense both prey and hunter do not move any longer.

In variant 3 we have 2 hunters and 1 prey, again. The prey has the shape of P1 in Figure 2. The non-learning hunter has standard form and uses the strategy of the non-learning hunter of variant 1. The learning hunter has the form of H3 of Figure 2. As in case of variant 1, the goal situation is to immobilize the prey which tries to stay away from both hunters and boundaries, thus complicating the hunters’ task.

4.2 Series 1

The number of time units spent on learning in contrast to the execution time of the other actions obviously should have some influence on the outcome of a game run. This becomes immediately clear if we are considering game variants in which agents move randomly or in which the learning agent relies on models of other agents based on observations, thus representing unsure knowledge. Since the simulation runs start with the situation that the learning agent expects to face after having executed “learn”, unsure knowledge or random factors result in an also unsure prediction of this situation. And the longer the learning takes the more unsure is the prediction.

But also if we have game variants for which we can reliably predict the start situation for the simulations, we expected the execution time for “learn” to have an impact on the success of hunter teams including an online learning agent. As Figure 3 shows, we were right to expect this. All three variants we described above allow a precise prediction of the situation the agent will face after learning. Our experiments show for all variants some decline in the success rate. But this decline is rather gracious – using 4 to 5 times the time for learning than for other actions achieves definitely still acceptable results. Even when needing 10 to 12 times the time we are still above 50 percent success

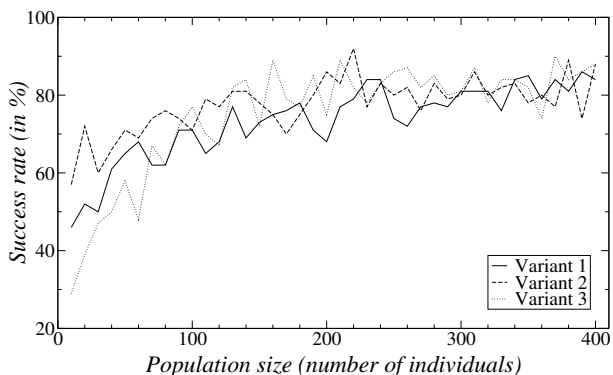


Figure 4: Influence of population size on success

rate in all variants. If we consider that allotting more time units for the real world game usually results in higher success rates (see Denzinger and Kordt, 2000), then these results indicate that our evolutionary online learning approach is feasible.

4.3 Series 2

The experiments of series 1 were performed with the following settings for the parameters that influence the processing time needed to execute “learn”: The population comprised 300 individuals in variant 1 and 100 individuals in all other cases. We defined $sap_{max} = 40$ for variant 1, $sap_{max} = 20$ otherwise. The number of generations was limited to 15 in variants 1 and 2 and restricted to 10 in variant 3. Furthermore, we defined $T_{max} = 35$ in variant 1, $T_{max} = 30$ in variant 2 and $T_{max} = 50$ in variant 3. Although our experiments so far show that learning can take some time without reducing the success rate very much, they also show that different variants can differ quite a lot in this regard (which we expected and which led to the different initial parameter values for the different variants). Therefore we examined the influence of the parameters mentioned above on the success rate in order to see how processing time can be reduced. Obviously, reducing the value of each of these parameters reduces the amount of computation necessary for “learn” and therefore the needed processing time.

As Figure 4 shows, relatively small populations already achieve rather good success rates. And an increase in the number of individuals (beyond 80 to 100 individuals) does not increase the success very much. Since the number of individuals that are generated during “learn” has a very large influence on the needed processing time, this is already a very promising result with respect to using our learning approach in more realistic applications. The other parameter influencing the number of individuals generated during a simula-

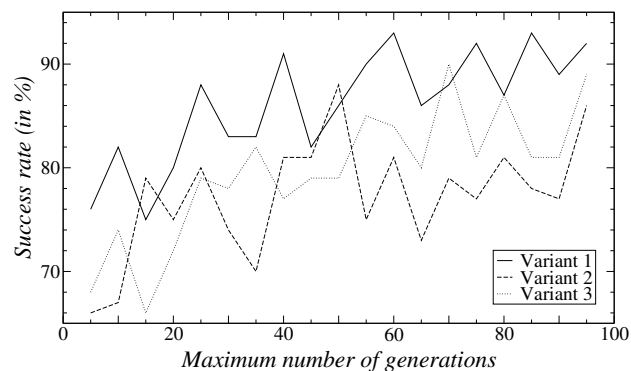


Figure 5: Influence of number of generations allowed on success

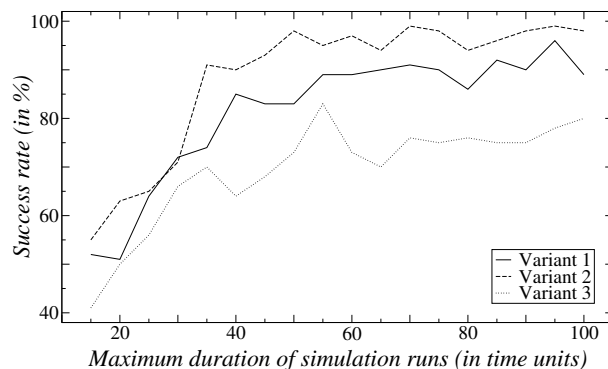


Figure 6: Influence of simulation length on success

tion run is the maximum number of generations. The influence of this parameter is depicted in Figure 5. Already with 25 generations all variants show a success rate above 70 percent. Then we have rather large variations for each variant, so that more than 40 generations seem to be redundant in either case.

The more time units are allowed in the simulation runs the more processing time is needed to evaluate the fitness of an individual, i.e. an agent strategy. Also, the further these simulations look into the future the more online learning changes towards offline learning. As Figure 6 shows, the longer we plan into the future the better is our chance for success (although we do not have a steady increase). But 30 time units already achieve a success rate of over 60 percent for all variants. And for variants 1 and 2 going above 40 time units does not increase the success much more. The relatively bad results for variant 3 are due to the fact that this variant allows for many situations in which both hunters are directly near the prey without having it immobilized, so that the m_{task} -values for these situations are rather good. While this affects all experiments with this variant, especially longer simulation runs suffer from the not very selective fitness values that are generated for this variant.

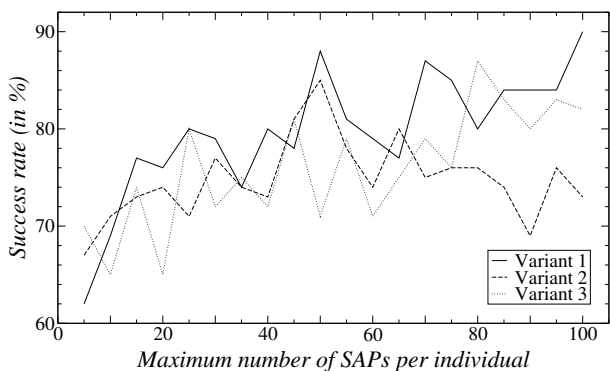


Figure 7: Influence of possible complexity of strategy on success

The last parameter we want to look at influences the processing time needed for executing “learn”, because the more SAPs are in a strategy, the longer it takes to decide what action to perform in a situation, since the similarity to each pair from the actual situation has to be computed. But the number of SAPs per strategy also determines how complex the strategies can be. If there are too few SAPs, then each possible strategy might be too simple. But if there are too many SAPs then the possibility is much higher to have unnecessary or even disturbing pairs. As Figure 7 shows, the influence of the complexity of the possible strategies on the success rate is not easy to describe and the different variants show rather different results. 25 to 35 SAPs achieve for all variants high success rates and already with only 5 SAPs we are above 60 percent success rate. But all variants show large variations in their results, with variant 2 definitely having decreasing success as a trend for larger numbers of SAPs. But this is not exactly surprising. In variant 2 the goal is to catch several preys and with a high number of SAPs our evolutionary approach leads to learning SAPs for catching each of these preys. In the simulations this is a good trait, because, due to the random factor involved, the fitness is evaluated by several runs. But in the real world the additional pairs make the hunter less efficient.

5 Conclusion

We presented an online learning approach for achieving cooperative behavior of agents. It is based on introducing an action “learn” that executes an offline learning method. In our case this method is the evolutionary approach presented in Denzinger and Fuchs (1996) based on SAPs and NNR. In this paper, our experimental evaluation concentrated on the relation of the execution times needed for “learn” and the other actions. In the area Pursuit Games we showed for

rather different game variants that learning times that are several times longer than the times needed for performing other actions still achieve good success rates (for fixed game lengths). We also investigated different values for the key parameters of the evolutionary offline learning method and found that already rather small values of these parameters achieve good success rates, again. Since small values for these parameters result in less processing time needed for “learn”, our experiments suggest that our evolutionary online learning approach can be successfully used in more realistic applications, like cooperating robot teams.

Before tackling such new application areas, in future work we want to investigate several other problems and phenomena with our Pursuit Games application. Among them are improving the evolutionary learning by getting more information out of the simulation runs, co-evolution of hunters and preys, and testing the adaptation capability of online learning agents when becoming member of already good cooperating teams. Other future work will be to improve on the modeling of other agents and to investigate how to better adapt the intervals between performing “learn”. The latter also focuses on making more use of seeing on- and off-line learning as extremes of a spectrum of possibilities instead of different kinds of learning tasks, which is suggested by our results.

6 References

- M. Boddy and T. Dean (1988). An Analysis of Time-Dependent Planning, Proc. 7th AAAI, pp. 49–54.
- M. Benda, V. Jagannathan and R. Dodhiawalla (1985). An Optimal Cooperation of Knowledge Sources, Technical Report BCS-G201e-28, Boeing AI Center.
- J. Denzinger and M. Fuchs (1996). Experiments in Learning Prototypical Situations for Variants of the Pursuit Game, Proc. ICMAS’96, pp. 48–55.
- J. Denzinger and M. Kordt (2000). Evolutionary On-line Learning of Cooperative Behavior with Situation-Action-Pairs, Proc. ICMAS-2000, IEEE Press, pp. 103–110.
- T. Haynes, R. Wainwright, S. Sen and D. Schoenefeld (1995). Strongly typed genetic programming in evolving cooperation strategies, Proc. 6th Intern. Conf. on Genetic Algorithms, Morgan Kaufmann, pp. 271–278.
- M. Manela and J.A. Campbell (1993). Designing good pursuit problems as testbeds for distributed AI: a novel application of genetic algorithms, Proc. 5th MAAMAW, pp. 231–252.
- M. Tan (1993). Multi-agent reinforcement learning: Independent vs cooperative agents, Proc. 10th Machine Learning, Morgan Kaufmann, pp. 330–337.
- C.J.C.H. Watkins (1989). Learning from Delayed Rewards, PhD thesis, University of Cambridge.
- G. Weiß (1995). Distributed Machine Learning, Infix-Verlag, Sankt Augustin.