Being the new guy in an experienced team – enhancing training on the job

Jörg Denzinger
denzinge@cpsc.ucalgary.ca

Sean Ennis
ennis@cpsc.ucalgary.ca

Department of Computer Science
University of Calgary, Canada

ABSTRACT

We present a modification to evolutionary on-line learning of cooperative behavior, based on a special action "learn", that allows the performance of a new agent replacing an old agent with somewhat different abilities in an experienced team to be improved. The general idea is to make us of the strategy of the old agent, obtained either directly or by the model the other team members made of it. This would be used as a seed strategy in the on-line learning process providing a focus to the process. This way, the flexibility of on-line learning remains, and the new agent is much less prone to making "beginner" mistakes that may prevent achievement of the team goal. Experiments with rather different variants of the pursuit game show that our method allows new agents to overcome the difference in abilities rather quickly. Thus, team performance is much better than when the new agent starts learning from scratch.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Developing cooperation schemes for multi-agent systems, and developing individual agent strategies that fit and implement such schemes is an important and unfortunately difficult and costly problem to solve in building a multi-agent system. The idea of using learning techniques to enable agents to develop cooperative behavior to solve given tasks has become more and more popular, because it provides a solution to this problem. Off-line learning approaches, as presented, for example, in [9], were able to reduce the human involvement in the solution process drastically, usually at the cost of long learning periods. In most cases, agent strategies learned off-line are not very flexible. As a result of this, on-line learning approaches have been developed that provide much more flexibility, but at the cost of having to interweave both working on the given task, and learning. Aside from the general problem of losing the possibility to perform the required task due to incorrect decisions made earlier, another consequence is that when using on-line learning, it is very seldom that solutions to the given task are produced very quickly. The known on-line approaches, whether they be based on reinforcement learning (see, for example, [10] for reinforcement learning in general and [14] or [11] for on-line learning with reinforcement learning) or evolutionary off-line approaches and an action learn (see [6], [7]), require a lot of (explorative) experiences before a satisfactory cooperative behavior is achieved.

If we look at human teams cooperating to perform given tasks and achieve given goals, very often the teams exist for a longer period than a certain person is a member of them. This means that the teams do not always have to start from scratch each time they tackle a task. In fact, on-line learned behavior survives the one time problem solving solution and becomes like off-line learned behavior the next time the task has to be performed (as was indicated in [6]). On-line learning only "kicks in", if its flexibility is needed. In such long existing teams, a situation that has to be dealt with is the exchange of an agent by another one and the consequent problem of fitting the new agent into the team during the execution of a task with the least amount of inefficiency as possible. In human teams, this is solved by *training the new guy on the job*.

There are a lot of applications of (pure computer-based) multi-agent systems that will face a similar problem, for example, if a new robot is brought into a robot team, or the software of an agent representing a human being or a company in auctions or other E-commerce applications is updated. Usually, the new agent replacing the old agent will have different abilities and the strategy of the old agent will not be sufficient for the new agent. Perhaps sometimes some action cannot even be performed perfectly by it. Therefore modifying on-line learning approaches so that they allow, on the one hand side, an old agent to "teach" the new one its strategy while on the other side still adapt it very quickly to the needs of the new agent, is an interesting and necessary research task. The goal must be to solve the team task with the new agent faster than in the case that the new agent starts its learning from scratch.

It should be noted that some people think that learning in this context is not necessary, because planning based on

AAMAS'02, July 15-19, 2002, Bologna, Italy.

some min-max search (to deal with having several agents acting) should do the job. But first of all, this means that each time an agent team is given a task, this planning method has to be applied (since no learning takes place). Even with only a few possible actions and a few agents, the search spaces are so large (we are talking about action sequences of at least 20 to 30 steps of each agent involved) that it takes a lot of time to traverse them, and storing the final state within an agent to make use of information from previous runs is not feasible. Finally, most heuristics for min-max search assume rational behavior of opponents, which cannot always be assumed in multi-agent systems, if agents do not have absolutely sure knowledge about other agents.

In this paper, we present a modification to the on-line learning approach of [6], as it is realized in the OLEMAS system, to deal with the situation of a new agent joining an experienced team. Learning agents in OLEMAS control their actions by comparing the situation they are in with the situations of a set of prototypical situation-action pairs (SAPs). They perform the action of the pair that is the most similar to the given situation. Learning is an evolutionary process, generating new sets of SAPs (so-called strategies) mostly out of strategies that performed very well in simulations of the team of agents at work. On-line learning is invoked via a special action "learn", and performs off-line learning by starting with the current situation and using simulations that are not going too far into the future (thus limiting the time an agent does not contribute to the given task).

Normally, if a new agent is put into an existing team, substituting an old team member, on-line learning would start with random strategies and would try to evolve an agent strategy that cooperates well with the strategies of the other agents. Even with having exploration of possibilities taken care of in simulations however, this usually takes quite some time and, due to the fact that from time to time actions have to be performed, might even result in missing all opportunities to solve the task given to the team. Our approach is to first determine how similar important characteristics of the old agent and its replacement are. If both agents are sufficiently similar, we extract from the old agent all SAPs of its strategy with actions that can be performed by the new agent. This is called its **seed strategy**.

We modify the evolutionary learning process by requiring each new strategy to at least contain a given percentage of SAPs from the seed strategy and at most another given (higher) percentage of them. This use of a seed strategy, in addition to the usual random start SAPs and mutations, lets the learning start with much better strategies, i.e. the experience of the old agent, while still allowing for the flexibility needed to deal with the differences between the agents. As our experimental evaluation shows, on-line learning agents using a seed strategy fit themselves into the team much faster and thus enable the team to accomplish given tasks much faster than in the case of the new agent learning its strategy without the help of a seed strategy.

2. EVOLUTIONARY ON-LINE LEARNING IN OLEMAS

OLEMAS (**O**n-**L**ine **E**volution of **M**ulti-**A**gent **S**ystems) was developed as a testbed for learning cooperative behavior of agents. Although in general all types of agent architectures and learning concepts can be integrated into OLEMAS to be tested on a wide variety of pursuit game variants, so far only some fixed strategies and evolutionary learning for agents based on prototypical SAPs and the Nearest-Neighbor Rule (NNR, see [4]) have been implemented in it. In the following, we will briefly recapitulate the basics of OLEMAS (see [6] for more details) so that we can describe our extensions in the next section.

2.1 Agent Architecture

In general, an agent $\mathcal{A}g$ can be described as a triple $\mathcal{A}g = (Sit, Act, Dat)$. Sit is a set of situations the agent can be in, Act a set of actions the agent can perform, and Dat a set of possible values of the internal data areas of the agent. The agent then realizes a function $f_{\mathcal{A}g}$: $Sit \times Dat \rightarrow Act$. The set Dat can be especially structured in addition to take into account that an agent is acting within a multi-agent system, for example by distinguishing between Dat_{own} for information about the agent itself, Dat_{sag} for sure knowledge about other agents and Dat_{usag} for unsure knowledge.

Our agent architecture is based on SAPs and the NNR. A set of SAPs, the agent's strategy, is part of Dat_{own} of an agent (i.e. $Dat_{own} = 2^{Sit \times Act} \times Dat_{own,rest}$[1]). When an agent is confronted with a situation s, it computes the similarity (resp. distance) of all situations in its strategy and s, and performs the action of the SAP whose situation is most similar to s (i.e. it applies the nearest-neighbor rule). If a situation description just consists of a vector of numerical values, then the Euklidean distance of two vectors is a possible similarity measure. Otherwise, knowledge about the concrete application has to be used to come up with a measure.

Note that this agent architecture results in a very reactive behavior, but it is not very complex. The use of the NNR allows to cover a lot of possible situations with a rather small set of SAPs.

2.2 Off-line Learning

Off-line learning is characterized by a so-called learning phase in which the behavior of an agent or team is formed, later applying the learned knowledge to the task for which it was learned. During this later application phase there is only exploitation of what was learned, that is, no further learning takes place.

In OLEMAS, an evolutionary learning approach is taken, based on a Genetic Algorithm (GA) for sets as an individual. The general idea of a GA is to work on a set of so-called individuals. Each individual is evaluated resulting in its fitness value. New individuals are generated out of existing ones by applying Genetic Operators, namely crossover and mutation. The "parents" of a new individual are selected by use of *fitness-influenced randomness*. New individuals replace the old ones with the lowest fitness.

Applied to the agent architecture of OLEMAS, individuals consist of a set of SAPs for each agent whose strategy is to be learned. The fitness of such an individual is determined by applying the agent team to the task that is to be learned. If the team is successful, the fitness is the number of steps needed, otherwise, after each action, a measure for

[1]The situation of an SAP does not have to be limited to the set Sit, it is also possible to use extended situations that, for example, contain values of data areas that are part of $Dat_{own,rest}$

the task fulfillment by the whole team is computed and the fitness is the sum of these measures over a whole run. If the task includes dealing with random effects then several runs contribute to the overall fitness of an individual.

The crossover operator needs two parent individuals, generating a new individual by randomly picking SAPs from these parents (up to a maximal number of SAPs). The mutation operator only takes one parent, deletes a randomly chosen SAP and (sometimes) adds a randomly generated SAP. In the initial set of individuals all SAPs in an individual are randomly generated.

Since the idea of off-line learning is to produce a strategy that is then used whenever the team has to perform the task it was learned for, usually it is okay to put quite a lot of computational effort into the generation of such a strategy (or strategies, if we learn the behavior of a whole team). Although [5] showed, that in OLEMAS the learned strategies can deal with some small changes in the task (which was expressed, for example, by agents with some random behavior), this flexibility is rather limited. This led to the subsequent work of incorporating on-line learning. A good off-line learned strategy, however, must be seen as an investment that, if possible, should not be completely abandoned.

2.3 On-line Learning by Off-line Learning

On-line learning does not have a learning phase and an application phase. Instead, learning is done while agents are already at work on the given task; they are trained on the job, but without advice up front. While this allows for more flexibility, especially in the beginning when not much has been learned, there is a chance that an agent performs actions that cannot be reversed. In some cases, this can result in not being able to complete the given task.

In OLEMAS, on-line learning is based on the off-line learning approach we just described by introducing a special action "learn". When "learn" is executed, the off-line approach is called with the situation the agent anticipates to be in after "learn" as start situation, models for the other agents to predict what they will be doing (on-line learning is performed within one agent, in contrast to off-line learning that develops strategies for all learning agents at once), and a rather limited number of actions as the length of the simulation as parameters. The off-line learning then uses these parameters to build simulations of the real world in which the agent is acting at the moment for evaluating strategies. Since learning takes time, it is necessary to start the simulations not with the current situation but with the anticipated situation after learning. Naturally, this anticipated situation might not be the real situation the agent will find itself in if random influences or models of other agents that are not very accurate are used (which requires to have a strategy to deal with it without having to start "learn" again immediately).

The models of other agents are either based on direct communication or observation. If a teammate communicates its strategy to the others, then this results in a very accurate model for it. If an agent only observes the agent to model and then uses the observed situation-action pairs as the modeled strategy, then this obviously cannot take into account any information about the internal data areas of the observed agent. As a result, the model might not be very accurate.

Since learning takes time, it somewhat interferes with fulfilling the task given to the team. Due to this, we cannot afford the long run times that off-line learning usually needs. By limiting the number of steps that are performed in a simulation run, a rather good run-time behavior in the real world can be achieved (see [7]). In OLEMAS, "learn" is performed periodically until either the given task is fulfilled or the allotted number of steps (in the real world) are used up. The intervals between executing "learn" are flexible depending on the performance the strategy that was learned last achieves. Whenever "learn" is executed, a new strategy is learned without any influence from the previous strategy (except that it led to the current situation).

3. EFFICIENT TRAINING ON THE JOB

While both on- and off-line learning in OLEMAS have proven to be successful, there are certain weaknesses:

- off-line learning is very inflexible, so rather small changes in the environment during the application often cannot be compensated for, and a whole new learning phase is needed (which is rather time consuming).
- on-line learning has to explore the possibilities, which can lead to missing opportunities, failing to fulfill the given task, and almost always will lead to poor efficiency in solving the task the first time.

These weaknesses are not particular to OLEMAS, they are general weaknesses of learning. While OLEMAS allows having a learning approach in between the two extremes (by, for example, executing "learn" as a start with a rather large number of steps in the simulations and then shorten this number drastically later; which kind of achieves moving from off-line to on-line learning), the efficiency in the real world and missed opportunities (due to not having sufficient experience) are still problems. If the task given to a team is completely new, even teams of human beings have these problems, so we cannot expect much improvement in this situation.

Very often teams perform tasks for quite some time. Human teams doing this are able to cope with changing team members without losing much efficiency. The reason for this is that new team members start out with a lot of advice on how to do their part of the team's task. Either the old team member provides this advice or the remaining team members communicate their model of how the new member should work to him or her. This even works, in the sense that the loss of efficiency is quite low, if the new team member has different abilities than the old one, as long as this difference is not too large.

In the following, we will present a modification to on-line learning in OLEMAS that achieves the same results when exchanging an agent, as we just described for human beings. We will first present the general approach and then concentrate on the key issue, namely on how to use a so-called seed strategy.

3.1 The General Approach

Before the modifications described in the following, dealing with the exchange of an agent in a team in OLEMAS required performance of on-line learning by the new agent without any use of either the strategy of the old agent or the model that the remaining agents had of the old agent. This assumed that there would be a strategy (resp. a sequence of

strategies to learn) for the new agent that would blend in with the strategies of the remaining agents. Naturally this would result in the performance of the given task in the real world to be quite ineffective.

Our new approach makes use of the strategy of the old agent (or the model of it of the remaining agents), resp. parts of it. The general procedure is as follows:

1. Compare the abilities of the old agent and its replacement.
2. If the abilities are too different then let the whole team re-learn the task.
3. Otherwise, if the abilities are identical (or the new agent has more abilities), use the strategy of the old agent.
4. Otherwise, filter out the SAPs in the strategy of the old agent that have actions the new agent cannot perform, which results in a seed strategy (i.e. all SAPs that are not filtered out).
5. Apply the improved on-line learning with the seed strategy.

While point 5 is obviously the most interesting one and will thusly be dealt with in its own subsection, we will now look more closely into points 1 through 4. Measuring the similarity with respect to abilties between the old and the new agent is also an interesting task that depends on the concrete application. This is because the application defines the situation descriptions, possible actions, and the possible agents in general. Note that ability is not the same as action. Two agents might be able to perform the same action in general, but they might do it at different speeds. Also, in special situations a certain action can be performed by one agent, while another cannot (because it needs more space or other preconditions). Therefore different applications will not only need different agent similarity measures, but also the threshold between sufficiently similar and too different (with regard to the applicability of on-line learning with a seed strategy) will vary. We will look into this issue in our experiments.

If the difference between the old and the new agent is too large, then we cannot expect the new agent to fit into the role that the old agent had in the exisitng team. Consequently, the whole team has to change its solution strategy for the given task and therefore has to learn a new one. There might be the possibility that using the old strategy of each team member as seed strategy for its newly to learn strategy might improve the learning process, but this will be the subject of future research.

If the new agent can follow the exact same strategy as the old one did, then there is obviously no need to learn a new strategy, if keeping up the good cooperative behavior of the whole team is our goal. If the new agent has additional abilities (because it is supposed to be an improvement over the old one), then not changing the strategy will not make use of these additional abilities (for example, being able to move faster). Again, using our concept of a seed strategy might allow to try out the new abilities in small steps. As a result of the on-line learning, a detoriation of the performance (compared to just using the old strategy) has to be expected and it might prove necessary to change the strategies of other team members in order to make use of the improved abilities. Therefore, looking into these possibilities will also be a future research topic after we established the basic advantages of using a seed strategy.

Point 4 of our general procedure does not address abilities anymore, but concentrates on the actions that the new agent is able to perform. Since our agent architecture with SAPs and the NNR results in rather reactive agent behavior, a complicated analysis of situations, actions and agent characteristics for each possible situation is much too much effort and is also something that should not be necessary due to the on-line learning capability of the agent. Therefore we just eliminate those SAPs of the old strategy that obviously cannot be of use to the new agent and use all others as the seed strategy that will guide the learning process of the new agent.

3.2 Using the Seed Strategy

The general idea behind using a seed strategy is to already have a frame for a good strategy in place that only contains a few holes that need to be filled by the on-line learning process. This way, the learning process is much more focused and there is a much lower probability that opportunities are missed due to bad actions in the real world.

Applied to evolutionary on-line learning with SAPs this means that we have to modify the evolutionary process to include the seed strategy. This is done by requiring that a certain percentage of all SAPs in an individual (i.e. strategy) are from the seed strategy.

More precisely, let s_{seed} be the seed strategy. When "learn" is invoked, a start population must be generated first. Let $p_{seed,start}$ be the parameter determining the percentage of SAPs from s_{seed} in the initial individuals. If strategy $s_{init,i}$ is supposed to contain k SAPs, then it is generated by randomly selecting $\lceil \frac{k \times p_{seed,start}}{100} \rceil$ SAPs from s_{seed} and generating $\lceil \frac{k \times (100 - p_{seed,start})}{100} \rceil$ SAPs randomly.

In the following, for a strategy s let seed(s) be the SAPs that are in the intersection of s and s_{seed} (seed(s) = $s \cap s_{seed}$) and new(s) the remaining SAPs in s (new(s) = s - seed(s)). Then we can modify the genetic operators crossover and mutation in the following way, given parameters $p_{seed,co}$, $p_{new,co}$ and $p_{seed,mut}$ with $p_{seed,co} + p_{new,co} \leq 100$.

crossover: Let s_1 and s_2 be two individuals in the current population that have been selected as the parents. Let seed-pool = seed(s_1) $\cup$ seed(s_2) and new-pool = new(s_1) $\cup$ new(s_2). Then we generate a new individual s consisting of at most k SAPs by randomly selecting $\lceil \frac{k \times p_{seed,co}}{100} \rceil$ SAPs from seed-pool, $\lceil \frac{k \times p_{new,co}}{100} \rceil$ SAPs from new-pool and k - $\lceil \frac{k \times p_{seed,co}}{100} \rceil$ - $\lceil \frac{k \times p_{new,co}}{100} \rceil$ SAPs out of $s_1 \cup s_2$.

mutation: The use of s_{seed} results in having several mutation operators that can be applied, differing in whether they contribute to the seed(s') or new(s') part of an individual s'. If s is the parent individual, then we have the following possible mutations to get s':

- just delete a SAP of s
- replace a SAP of seed(s) by a randomly chosen SAP of s_{seed}
- replace a SAP of new(s) by a randomly generated SAP

- replace a SAP of seed(s) by a randomly generated SAP if $\frac{|seed(s)|}{|s|} > \frac{p_{seed,mut}}{100}$
- replace a SAP of new(s) by a randomly chosen SAP of s_{seed} if $\frac{|seed(s)|}{|s|} \leq \frac{p_{seed,mut}}{100}$

The use of two parameters in crossover allows some flexibility in the use of the seed strategy by allowing for some variability in the seed content of the evolved strategies. This is also reflected in the different mutations (that are applied in equal percentage during the learning process). Different values for $p_{seed,start}$, $p_{seed,co}$, and $p_{seed,mut}$ also give us added flexibility, although we will not make use of this flexibility in our experiments.

4. EXPERIMENTAL EVALUATION

The OLEMAS system offers pursuit games as the testbed for the learning concepts for cooperative behavior. As we will see in the first subsection, there are a wide variety of pursuit games of different difficulty and different realism that allow for agents of rather different abilities. In the second subsection, we will present our experiments with 3 different game variants (depicting the different problems to overcome) and the enhanced training on the job.

4.1 Pursuit Games

Pursuit games as a testbed for multi-agent systems were first introduced in [2] and since then have been used rather frequently in literature and in quite a number of variations. [5] introduced a large list of features of pursuit games that can be varied, and since then even more features, especially feature instantiations, have been introduced by various authors.

The general idea of pursuit games is rather simple: on a playing field, consisting of connected grids, several hunter agents have to work together to catch one or several prey agents. The game consists of turns that represent a discrete time line. Usually, the game is limited to a given number of turns. Among the features that can be varied in addition to this time limit are the field: infinite, with borders, with or without additional obstacles; the agents: number, shape, speed, possible moves; the strategy of the prey agents; or how we define catch. In the original version of the game, the playing field was infinite, all agents occupied only one grid and could move only to the north (N), west (W), south (S), and east (E) at the same speed of one grid per turn; the only prey agent moved randomly and catch was defined as the four hunters surrounding the prey so that it could not move.

For our purposes the features concerning a single agent, like its shape, the actions it can perform, and the speed for each action, are of special interest because these features define the abilities of an agent. This allows that a new agent might not be able to move in all of the directions that the old agent could, or it might move slower in one or several directions than the old one. The latter is a problem in multi-agent scenarios, because the timing between agents will be off. Another difference in abilities that can be difficult to overcome in multi-agent scenarios is a difference in shape. If the shape of the old agent is just a point occupying one grid and the new agent is a rectangle occupying 2 by 3 grids, then the old agent might go into places the new agent cannot and the new agent might block moves of the other hunters that were essential for the success of the old team. Generally, pursuit games allow a wide variety of difference in the abilities of agents and therefore are a very good testbed for our problem, the exchange of agents in an experienced team and how to overcome the consequences while working on the given task.

Before we look at our experiments, let us provide some important instantiations of the general learning methods presented in Section 2 (more details are provided in [6] and [7]). In OLEMAS, the situation vector seen by an agent consists of the coordinates relative to the agent and the orientation for each of the other agents in a fixed order. Consequently, the similarity of two situations just sums up the square of the difference in numbers for each field and an adjusted square of the difference in orientation (based on assigning numbers to the possible orientations). The fitness measure is based on computing the Manhattan distance between an agent and the prey at each turn. These distances are summed up for all agents and all turns, if the team was not able to catch the prey, else we just take the number of turns necessary to do the catching (as we already described earlier).

4.2 Results

In order to demonstrate the capabilities of our approach we have selected three variants of the pursuit game in which "newbie" agents can seriously endanger the success of a team. Naturally, there are many other variants we could use, but we hope that these variants demonstrate well the possible problems that a replacement agent has to face. Since we are also interested in how varying differences in abilities between old and new agent influence the success of our training on the job, we have provided for one variant, several replacements (and one alternative for each of the other variants) and their experimental results. Before we take a look at the individual variants and the results, let us first describe the general setting and the aspects all experiments have in common.

The experienced teams that are the basis of our experiments were obtained by letting OLEMAS learn them offline. For each variant this took more than 6 hours CPU time, whereas all runs reported in Table 1 took less than 5 minutes, including the time spent in on-line learning. All variants have as goal to surround and immobilize the prey agent. In order to clearly demonstrate the capabilities of our method and to allow for comparability of the different runs for a game variant, the variants do not include any random factors, like random moves by the prey or random start situations.

All on-line learning agents were allowed a maximum of 35 SAPs per strategy and the parameters of the GA for on-line learning were always a maximum of 20 generations, a population size of 100 and a mutation rate of 30 percent (i.e. 30 percent of the applications of genetic operators were mutations, 70 percent crossovers). The agents also are informed about the strategies of all other agents to be used in their simulation runs. The action "learn" takes 3 turns to perform and all other actions take just one turn (with one expection regarding variant 3). The action "learn" was performed after at least 10 and at most 30 turns. With regard to the parameters guiding the use of the seed strategy, we set $p_{seed,start} = p_{seed,mut} = p_{seed,co}$ for all experiments.

Table 1 reports the different variants (with the number of turns needed by the experienced team to solve them) and

Table 1: Experimental results

Variant	Exp. Team	Replacement	from scratch	with seed strategy			$p_{seed,co}$	$p_{new,co}$
				best	worst	mean		
1	41	A1 ← A11	–	487	514	500.8	80	20
		A1 ← A12	–	363	388	370.6	60	20
2	14	A2 ← A21	–	212	240	224.9	60	20
		C2 ← C21	–	144	163	153.4	90	5
3	70	A3 ← A31	870	147	165	154.4	90	5
		A3 ← A32	–	499	520	507.6	90	5
		A3 ← A33	532	75	93	82	60	20
		B3 ← B31	–	111	132	120.1	80	20

the replacements we performed, the best result on-line learning in OLEMAS produced without using the seed strategy (as number of turns until the prey was catched), the results of OLEMAS using the seed strategy and the values of $p_{seed,co}$ and $p_{new,co}$ that produced these results. Since on-line learning is based on an evolutionary algorithm, different runs usually lead to different results (due to the random influences that are an essential part of a GA). Therefore we report the best run, the worst run and the mean value of the 10 runs we performed (the success in 10 runs is used as fitness when dealing with random effects in a game, therefore we have chosen this as the value here as well). We tested several ratios of $p_{seed,co}$ to $p_{new,co}$ for each of the variants and replacements. Table 1 reports the best combination of these only (and what this best ratio was, see later for comments on this) to keep the table simple. For all replacements in all variants, just using the strategy of the old agent did not result in catching the prey within 1000 turns.

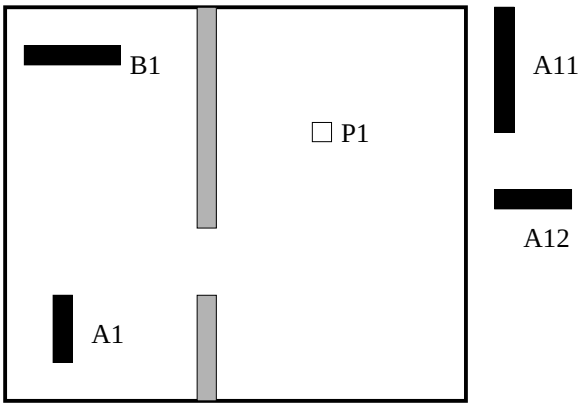
Variant 1

Pursuit game variant 1 in our experiments requires the two hunter agents to pass between two obstacles before they then have to corner the prey (the start situation is depicted in Figure 1). Potential problems that the experienced team had already solved where collisions of the agents while attempting to pass through the opening (like several people trying to pass through a door at the same time). After passing the obstacle (agent B1 passes first) there are other potential problems regarding getting the prey into a corner. The experienced team forms a wedge that brings the prey (that tries to stay away form the nearest hunter as far as possible) against a corner. All agents can move in all 8 directions and both hunters, in addition, can turn right and left (around a center point that is located in the middle of the agents).

Our first experiment replaces agent A1 by the larger agent A11, and therefore has to figure out that it must turn before being able to pass through the opening. Since agent B1's strategy (that is not changing) is to pass first through the opening, learning from scratch results in exactly the tangle at the opening we described above. In fact, the replacement agent blocks the opening and agent B1 then blocks all attempts of the replacement agent to turn.

By having the seed strategy providing the necessary order in passing through the opening, on-line learning (i.e. the new-part of the strategies) figuring out that a turn is necessary, then getting the prey into the corner guided by the seed strategy, and finally having on-line learning account for

Figure 1: Start of variant 1 and replacements



achieving the catch, the improved OLEMAS on-line learning wins the game.

In the second experiment to this variant we replaced agent A1 by A12 that does not have the problem of turning before the opening, but after succesfully making it through, it has a different orientation than A1 did. Consequently, this new agent has it easier than the one in the first experiment, which is also documented by the lower number of turns needed to win.

Variant 2

Our second game variant features an infinite grid together with a prey that tries to maximize the distance to the nearest hunter (see Figure 2 for the start situation). Consequently, whenever the prey gets past any one of the hunters, it will not be caught anymore (all agents have the same speed). That does not leave much room for errors by a new agent, proving that this scenario is an excellent example for the need to get the new agent up to speed very fast (or at least keep some status quo until the agent has figured out what to do). The three hunters in the experienced team are very specialized: A2 is only allowed to move E, SE, S, SW and W, B2 E, SE and S, and C2 can move N, NE, E, W and NW. Since they cannot turn, there is only one constellation for them to catch the prey, that is allowed to move in all directions, and the team must cooperate very well to achieve this (which this particular team does, as indicated by the only 14 turns necessary to win).

If we replace A2 by A21, which is allowed to move in all directions and may also turn left and right (the center is, again, the middle of the agent), then the problem it has to

Figure 2: Start of variant 2 and replacements

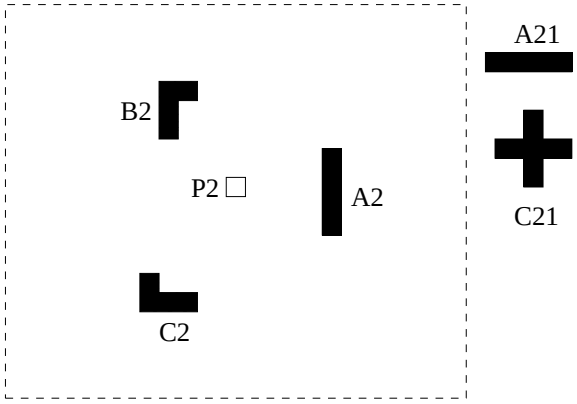
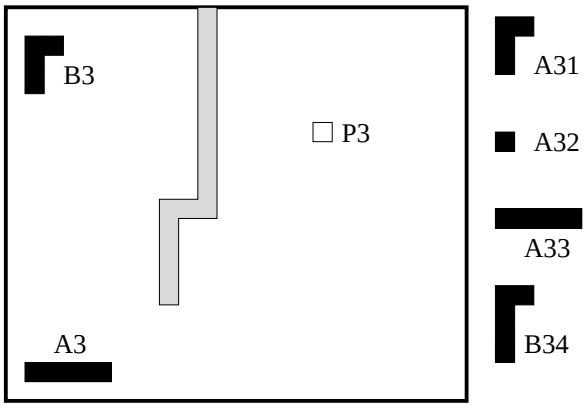


Figure 3: Start of variant 3 and replacements



face is to turn early enough so that prey and other hunters do not block such a turn. On the other side, if A21 stays too far away (or makes wrong moves), the prey can escape. As Table 1 shows, in order to achieve this, we have to allow for quite a bit of flexibility in the strategies (i.e. 20 to 40 percent SAPs not in the seed strategy in all individuals).

Agent C21 is also allowed moves into all directions, but its blocking behavior against the prey is different from C2, so that a slightly different approach towards the prey is necessary to achieve a catch. As our results show, this can be accomplished with a rather large percentage of seed-SAPs and faster than in the first experiment to this variant.

It should be noted, that for this particular scenario there might be a chance to employ min-max search, because of the small number of turns needed by the experienced team (although 14 turns still generate a very large search space and there is no guarantee that the new team does not need more turns to be successful). Using evolutionary on-line learning (with a seed strategy) allows to concentrate on what would be only a few of the paths in such a min-max search graph and, despite the shortness of simulation runs compared to what is done in off-line learning, an evaluation much deeper than everything possible by a heuristical control of a min-max search before committing to actions.

Variant 3

While variants 1 and 2 were rather unforgiving with regard to early errors by the replacement agent, with variant 3 we have a scenario that still can profit a lot from the seed strategy, but at least gives the on-line learning without seed strategy a chance, as Table 1 shows. As Figure 3 shows, the hunters A3 and B3 have to pass an obstacle, again, and then have to immobilize P3. This time, the obstacle does not force immediate changes against the seed strategy on replacements, but timing between agents is a factor, again (as is to be expected in experienced teams). In the experienced team, B3 passes the obstacle after A3, gets above P3 and then A3 comes from below, turns at the right time and achieves the catch against the right side of the field. P3 is the same prey as in the first two variants and A3 and B3 can move in all directions and turn left and right.

In the first experiment to this variant, agent A31 has a different shape that requires two turns to bring it into the catch position of A3. In the second experiment, agent A32 does not need the turn of A3, and in fact cannot perform turns at all. As the results show, eliminating the SAPs with turns as actions offsets the agents so much that their timing gets out of synchronization. This leads to a worse behavior than in the first experiment (that required additional actions). This shows that holes in strategies due to filtering out of SAPs can have rather negative effects (in cooperative multi-agent environments where agents rely on certain actions of other agents). Still, however, the result is better than in the case of learning from scratch.

In our third experiment, we substitute A3 by agent A33 that has the same form, but can only perform the moves N, E, S, SW, W, and NW and turn left and right. So, no moves to NE or SE are allowed. In the strategy of A3 there are not many SAPs with these two actions, but nevertheless they are performed by A3 and therefore A33 has to compensate for this. With a $p_{seed,co}$-value of 90 and a $p_{new,co}$-value of 5 we can already achieve success with around 108 steps, but catching the prey in the upper right corner, i.e. differently than the experienced team. The results reported in Table 1 use more variablility and achieve the same catching situation as the experienced team. So, the agent with this more variability learns how to compensate for the unavailable moves.

In our forth experiment, we substituted B3 by B34 which has a slightly different shape, can move N, E, SE, S, SW, W and NW and is slower in moving W (i.e. a move W takes two turns). This means that each backing up of B3 towards west will hamper the timing in the team considerably. As our results show, with some flexibility and the guidance by the seed strategy the new agent is able to avoid such situations and therefore stays on target, which allows the team to be very successful, despite the new agent.

General Remarks

The usage of a seed strategy allows our evolutionary on-line learning to overcome the exchange of an agent by a new agent with different abilities and trains the new agent on the job to achive the task given to the team. This way we can make use of the experience that the old agent had in working together with the other agents that was gained under rather high costs. In addition, this effect is rather stable, which is an important issue in evolutionary systems. As Table 1 shows, varying differences in abilities (and different scenarios) require different percentages of SAPs from the seed strategy in the evolved strategies. Sticking too much to the seed strategy may not allow the necessary flexibility, while allowing for too much flexibility results in loosing the focus and making mistakes that compromise fulfilling the

given task.

Our results suggest that difference in abilities is not the only factor influencing the success. The game variant and the quality of the strategies of the agents in the experienced teams also have some influence on what combination of $p_{seed,co}$ and $p_{new,co}$ is best, resp. successful. We added to the seed strategies used in our experiments to version 2, a few SAPs from strategies that were predecessors of the strategies used by the experienced team. Not only did the performance deteriorate in general, but especially high values of $p_{seed,co}$ did not produce good results anymore. Here we will have to do more experiments to get an idea how an automatic selection of $p_{seed,co}$ and $p_{new,co}$ can be realized.

5. RELATED WORK

While there are lots of works concerned with on- and off-line learning in multi-agent systems, the problem of substituting an agent in an experienced team by another agent with different abilities has not been tackled so far (at least to our knowledge). There are, however, some works that can be seen as addressing a related problem, namely *learning by imitation* (see, for example, [8], [1], or [3]).

In contrast to the basic assumption in learning by imitation, we do not have an agent to observe and imitate, we just have its strategy (or its observed strategy). Also, in nearly all of the approaches to learning by imitation, there is no difference in abilities between old and new agent and the learning agent is not part of a team, i.e. no other agents rely on the new agent atleast doing an acceptable job.

Price and Boutilier presented in [13] a reinforcement learning approach that tackled the problem of difference in ability, but did not work on learning cooperative behavior, and the difference in abilities was only due to different action sets. They extended their approach of [12] by a limited search for repair action sequences that replace actions that become infeasible due to difference in abilities (here: actions that lead into deadends because the follow-up actions of the imitated agent cannot be performed). They provided no experiments in which timing, due to other agent's needs, was an issue, so that we cannot comment on the potential of their approach with this regard.

6. CONCLUSION

We presented the problem of how to deal with an agent replacing another agent in an experienced team that has different abilities. We also offered a first solution to this problem based on improving evolutionary on-line learning by using a seed strategy. We presented different variants of the pursuit game highlighting the different aspects of the problem and different replacement scenarios and showed with experiments that our improved on-line learning allows the new agent to avoid early mistakes that are the pitfall of just using on-line learning without a seed strategy. If the difference in abilities between old and new agent is small enough, our improved on-line learning allows the new agent to fit into the cooperative team strategy fast enough to allow the new team to fulfill its task. Too much of a difference throws off the timing in the team and consequently the cooperative behavior and as a result the team is not able to solve the given task without a rather expensive retraining of all agents.

Future work has to focus on a better understanding of what differences in abilities require what percentage of seed strategy SAPs. Such a better understanding should also help us to use seed startegies for off-line learning of whole teams, if the difference in abilities is too large so that the whole team has to adapt itself to the new agent. Hopefully, we will be able to speed-up the retraining process of the team. This should then also allow to speed up retraining if the new agent has additional abilities, that might allow for better task fulfillment if the team adjusts itself appropiately. Since learning of cooperative behavior of teams is a time consuming task requiring the agents to make lots of experiences, a re-use of at least some of the results of such a learning process after changes to agents are made is definitely an important goal in multi-agent systems.

7. REFERENCES

[1] C. Atkeson and S. Schaal. Robot learning from demonstration, Proc. ICML-97, 1997, pp. 12-20.

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

[3] A. Billard and G. Hayes. Learning to communicate through imitation in autonomous robots, Proc. ICANN-97, 1997, pp. 763–768.

[4] T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classification, IEEE Trans. on Information Theory, Vol. IT-13, 1967, pp. 21–27.

[5] J. Denzinger and M. Fuchs. Experiments in Learning Prototypical Situations for Variants of the Pursuit Game, Proc. ICMAS'96, AAAI Press, 1996, pp. 48–55.

[6] J. Denzinger and M. Kordt. Evolutionary On-line Learning of Cooperative Behavior with Situation-Action-Pairs, Proc. ICMAS-2000, IEEE Press, 2000, pp. 103–110.

[7] J. Denzinger and M. Kordt. On the influence of learning time on evolutionary online learning of cooperative behavior, Proc. GECCO-2001, Morgan Kaufmann, 2001, pp. 837–844.

[8] G. Hayes and J. Demiris. A robot controller using learning by imitation, Technical Report DAI No. 676, University of Edinburgh, 1994.

[9] T. Haynes, R. Wainwright, S. Sen and D. Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies, Proc. 6th Intern. Conf. on Genetic Algorithms, Morgan Kaufmann, 1995, pp. 271–278.

[10] J. Hu and M.P. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm, Proc. 15th Machine Learning, AAAI Press, 1998, pp. 242–250.

[11] N. Ono and K. Fukumoto. Multi-agent Reinforcement Learning: A Modular Approach, Proc. ICMAS'96, AAAI Press, 1996, pp. 252–258.

[12] B. Price and C. Boutilier. Implicit imitation in multiagent reinforcement learning, Proc. Machine Learning-99, 1999, pp. 325–334.

[13] B. Price and C. Boutilier. Imitation and Reinforcement Learning in Agents with Heterogeneous Actions, In Stroulia, Matwin (eds.): Advances in Artificial Intelligence, Springer LNAI 2056, 2001, pp. 111–120.

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