

# Dealing with new guys in experienced teams - the old guys might also have to adapt

Jörg Denzinger

Department of Computer Science  
University of Calgary, Canada  
denzinge@cpsc.ucalgary.ca

Sean Ennis

Department of Computer Science  
University of Calgary, Canada  
ennis@cpsc.ucalgary.ca

## ABSTRACT

This paper deals with handling the situation when in an experienced team of agents one team member is replaced by a new agent with different abilities. We extend earlier work that focused on how the new agent can make use of the strategy of the old agent as seed strategy in an evolutionary learning approach with prototypical situation-action pairs and the nearest-neighbor rule as agent architecture. We now allow also the remaining team members to adapt their behavior using their old strategies as seed strategies in evolutionary on-line learning. After giving the new agent some time for a basic adaptation, the remaining team members take turns to adapt themselves.

Our experimental evaluation shows that this extension allows us to deal with situations where the difference in abilities is too big for the previous approach to succeed. We also can report examples in which additional abilities of the new agent are made use of to get an overall better team performance than with the old agent.

## KEY WORDS

Intelligent agents, learning/adaptation, cooperative behavior, Genetic Algorithms

## 1 Introduction

Cooperation of agents offers a way to perform tasks no individual agent can perform alone and to enhance the efficiency of problem solving in general. But developing a cooperation scheme suitable for a task and individual agent strategies that fit into the scheme is often very difficult and costly. Learning of cooperative behavior by agents offers a solution to this problem that shifts the development effort away from human developers and into the computer agents. But learning, be it in advance in form of off-line learning or interleaved with actions towards fulfilling the given task as on-line learning, still requires a lot of experiences, and hence time, before an acceptable cooperation scheme and appropriate agent strategies are learned.

If a scheme and strategies are learned and nothing changes with regard to task and agents, then the effort put into learning is definitely well spent. But if we look at human teams that went through learning together, then we often see that nothing is more constant than change. Even if

the task to do does not change, team members leave, either only temporarily or permanently. So, human teams very often have to deal with the situation of having an experienced team in which one of the members is replaced by a “new guy”, while the team is still expected to perform the task it is experienced in. And human teams in most cases can cope with such a situation without going through an intensive and costly retraining starting from scratch. Instead, the new guy is given the strategy used by the member he/she replaces and he/she is expected to modify this strategy according to his/her abilities, while already participating in performing the team task, i.e. he/she *trains of the job*. If this is not successful enough, because the abilities of the new guy are too different from those of the previous team member, then other team members adjust their strategies until the team is successful, again.

As in human teams, teams of (pure computer-based) agents face situations in which an experienced team has to cope with a new agent replacing an old one with different abilities. Examples are new software versions for individual agents or a new robot replacing a robot that was not repairable anymore. Especially with robots, it may not even be a new robot that replaces an old one, the “old” one might just lose one of several abilities (due to some hardware or low-level software failure) and therefore can be seen as a “new” agent with different abilities (especially if the next repair shop is on another planet, a situation NASA will face in the future). But also new software versions for agents are of quite some interest. If the new version is an improvement, naturally we would like to make good use of new features and abilities, which often requires other agents to adapt to allow for the usage of these features.

In [2], we presented a method how evolutionary learning of cooperative behavior based on an agent architecture using situation-action pairs and the nearest-neighbor rule can be modified to deal with a new agent in an experienced team without having to learn from scratch. But this method, so far, was only used by the new agent, i.e. the other experienced team members did not change their behavior at all. In this paper, we examine how we can deal with cases where it is not sufficient to just have the new guy adapt the strategy of the agent it replaces. If the difference in abilities is too big or if there are new abilities, then the other team members also have to adapt.

Our idea is to use the same method we used in [2], namely combining parts of a seed strategy with newly learned parts, but now not only have the new agent doing this. After giving the new agent some time to do its adaptation, the other “old” team members also try to adapt, one after the other. This way, the team as a whole does not become too unbalanced, which could cause a breakdown of the cooperative effort with a high probability of not achieving the given goals.

We extended our OLEMAS system (see [4], [2]) to include this idea. In its application domain, Pursuit Games, we tested interesting and rather different variants and the suggested adaptation of not only the new agent but also the old team members was either able to solve game variants with replacement scenarios consistently faster than having just the new agent adapt or to deal successfully with replacements that with adaptation by just the new agent could not be solved at all, due to a too big difference in abilities between old and new agent. Even more, we can report game variants with replacement scenarios in which the new agent has additional or better abilities and where adaptation of the other agents allows to make use of those new abilities, thus *enhancing* the team performance (compared to the experienced old team).

## 2 Evolutionary learning with SAPs

Our evolutionary learning approach, as used in OLEMAS (On-Line Evolution of Multi-Agent Systems), uses as agent architecture prototypical situation pairs (SAPs) and the nearest-neighbor rule (NNR) (see [3]). Since usually agents are characterized as triple  $Ag = (Sit, Act, Dat)$ , in which  $Sit$  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, we can see  $Ag$  as realizing a function  $f_{Ag}: Sit \times Dat \rightarrow Act$ . Our agent architecture assumes that in  $Dat$  we have a set  $strat \subseteq Sit \times Act$ ,  $strat = \{(s_1, a_1), \dots, (s_n, a_n)\}$ , called a strategy, and we also have a similarity measure (or distance)  $sim$  on situations (i.e.  $sim: Sit \times Sit \rightarrow \mathbb{R}$ ). If our agent is then confronted with a situation  $s$ , it computes  $sim$  for  $s$  and all  $s_i$  occurring in  $strat$  and performs the action  $a_j$  associated with  $s_j$ , where  $s_j$  is the situation with minimal  $sim$ -value (with respect to  $s$  and an appropriate tie-breaker, if necessary). In OLEMAS, we describe situations as a vector of numerical values that report the distances of all other agents to the agent (see [4]). Then a good similarity measure is the Euclidean distance of two vectors.

Off-line learning is achieved by applying a Genetic Algorithm that works on strategies, i.e. sets of SAPs (see [3]). Each strategy is evaluated resulting in its fitness value. New strategies are generated out of existing ones by applying Genetic Operators, namely crossover and mutation. The “parents” of a new strategy are selected by use of *fitness-influenced randomness*. New strategies replace the old ones with the lowest fitness.

In OLEMAS, the fitness of a strategy is determined

by applying the strategy (resp. a strategy for each learning agent) to the task that is to be learned (for a certain number of steps) and in most cases measuring success of the strategy after each step. The fitness is then the sum of the success measures after each step, resp. the average fitness of several such runs, if random factors can influence the task fulfillment. Only if the simulation leads to fulfilling the task completely, we use as fitness measure the number of steps until success.

The crossover operator takes two strategies and generates a new one by randomly picking up to a maximal number of SAPs out of the union of the two parent strategies (duplicates are not allowed). There are three kinds of mutation operators, each of which needs one parent strategy, copies it and then the new strategy is either the copy plus a randomly generated SAP (if the maximal number of SAPs has not been reached already), the copy without a randomly picked SAP or the copy with a randomly picked SAP replaced by a randomly generated SAP. The initial strategies at the start of the evolutionary process are totally generated at random.

The just described off-line learning can be used also for on-line learning by introducing a special action “learn” (see [4]). When “learn” is executed, the off-line learning approach is called with the situation the agent expects to be in after “learn” as start situation and a rather limited number of time steps as the length of the simulation runs performed for fitness evaluation. Since on-line learning is performed by a single agent (although several or all agents can do on-line learning on their own), the off-line learning approach needs a way to simulate the behavior of the other agents. This is achieved by having the on-line learning agent building models of the other agents. This modeling is done by either watching an other agent and protocolling situations and executed actions (and them using them as SAPs with our agent architecture) or by simply having the other agents communicate their strategies to the agent (which obviously should be done by cooperating agents).

With the described input, the off-line learning approach is then capable to simulate the “real world” the agent is acting in for the given number of steps into the future, but it has to be noted that what really will happen in the real world can be different from the simulation. If the models of other agents are not totally accurate, then already the start situation for the simulations can be off the mark and the more the simulation tries to look into the future the larger the differences will get. Another reason for a simulation to differ from what will really happen can be random, resp. unpredictable events in the world. 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 performing “learn” are in the basic version flexible, depending on the performance that the current strategy achieves. In [4] we suggested that there are several ways how the strategy before “learn” and the one generated by “learn” can be combined, but in the basic version the new one just replaces the old one (which

will change for the old team members, now).

### 3 Using a seed strategy

While the rather detailed evaluation in [5] of the on-line learning approach sketched in the last section showed that already parameter settings of the system that lead to only short on-line learning times are very successful, it is also clear that an on-line learning agent needs some time (i.e. a number of steps, including several executions of “learn”) to start cooperating well with other agents. This also means that such an agent, and therefore also its team, may miss opportunities for fulfilling the given task that come up early in the run, thus potentially becoming unable to fulfill the task, if no other opportunities come up later. Note that this is a general weakness of on-line learning, not particular to our approach.

This problem can be solved by someone pointing out to the agent opportunities and suggesting appropriate actions. In the case of an agent coming newly to an experienced team, the strategy of the old team member replaced by the new agent (that usually is known to the other members) can accomplish this pointing out, provided that the strategy can be used by the new agent. If this is possible, then the rest of the team can do what they normally do and we will not see any change in performance. But even if old and new agent have slightly different abilities, the new agent can profit a lot from the strategy of the old agent, if it focuses its on-line learning around this strategy, as we showed in [2]. The following modification to the basic evolutionary learning method, to allow for the use of a seed strategy –the strategy of the old agent–, improves on-line learning by on the one hand evolving the usable parts of the seed strategy and on the other hand providing flexibility to deal with the differences in abilities by evolving new SAPs. More precisely, we suggested the following general procedure:

1. Compare the abilities of old and new agent.
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.

The key point in this procedure is 5, the improved version of the evolutionary learning. Let  $s_{seed}$  be the seed strategy from the old agent. When invoking “learn”, we first have to generate a start population of strategies. Let  $p_{seed,start}$

be the parameter determining the percentage of SAPs from  $s_{seed}$  in these initial strategies. 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 strategies in the current population that have been selected as the parents. Let  $seed\text{-pool} = seed(s_1) \cup seed(s_2)$  and  $new\text{-pool} = new(s_1) \cup new(s_2)$ . Then we generate a new strategy  $s$  consisting of at most  $k$  SAPs by randomly selecting  $\lceil \frac{k \times p_{seed,co}}{100} \rceil$  SAPs from  $seed\text{-pool}$ ,  $\lceil \frac{k \times p_{new,co}}{100} \rceil$  SAPs from  $new\text{-pool}$  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 a strategy  $s'$ . If  $s$  is the parent strategy, 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 general idea of using the seed strategy is to always have at least a certain percentage of SAPs in a strategy that are out of the seed strategy (that already has shown that it works well for the old agent). By requiring only a certain percentage of new SAPs, we can have a grey area between  $p_{seed,co}$  and  $p_{new,co}$  that is also supported by the mutations, so that we can be flexible with regard to how many SAPs from the seed strategy are really in a strategy. Note that the evolutionary learning process this way also takes care of identifying the SAPs in the seed strategy that are well suited for the new agent.

The experiments in [2] showed that the usage of the strategy of the old agent as seed strategy allows the new agent to get up to speed regarding the cooperation with the rest of the team much quicker than learning from scratch does. Naturally, depending on the particular difference in abilities between old and new agent and depending on the task the team has to fulfill, different settings for  $p_{seed,co}$  and  $p_{new,co}$  were best.

## 4 Adapting to a new agent as a team

While the approach of using a seed strategy for the new agent was quite successful, there have been two problems it could not deal with at all:

- If the difference in abilities is too big, it might be impossible for the new agent to fulfill the role in the team task that the other remaining team members require from it.
- If the new agent has more (usually better) abilities than the old agent, we do not make use of these abilities. Point 3 in the procedure already prevents this, because it might require the other agents to change their behavior to allow for using new abilities, which we could not do so far.

In order to solve these problems, we propose the following improvements to our approach from the last section, that now allow also for the remaining team members to adapt using their old strategies as seed strategies. The new general procedure is as follows:

1. Filter out the SAPs in the strategy of the old agent that have actions the new agent cannot perform. This results in the seed strategy for the new agent.
2. Apply the improved on-line learning with this seed strategy for the new agent for a given amount of steps  $t_{initial}$  in the real world.
3. If the task is not fulfilled after  $t_{initial}$  steps, let the first remaining agent perform “learn” with its old strategy as seed strategy (and an additional condition on the combination of the result of this learning and the old strategy, see later).
4. As long as the task is not fulfilled, do
  - (a) perform  $t_{test}$  steps in the real world
  - (b) let the next remaining agent perform “learn” as described in 3.

This new procedure introduces two new parameters,  $t_{initial}$  and  $t_{test}$ , that define how long we try for the new agent to adapt and how long we try every new adaptation by one of the remaining team members in the real world. In our experiments, a  $t_{test}$ -value of 1 was already sufficient since communicating their own strategies by all cooperating agents together with the lead by the old strategies provided a good prediction of the quality of the adapted strategies, so that no big evaluation in the real world was necessary.

Obviously, with regard to the remaining old agents, we are taking a step by step approach having the adaptation attempts in a cascading fashion. This way, we have in the real world always at the most one agent that is inactive due to performing “learn” (which is important for our application area, because two many inactive agents kind of

guarantee that the prey can escape) and we avoid having too many changes in strategies at once (also, we keep the search spaces small by having to adapt at any time at most one agent). Naturally, we first require the new agent to do its best (within the given time limit of  $t_{initial}$ ) to adapt to the team requirements. If the difference in abilities is only small, this might already be enough and then our new procedure achieves exactly the same effect as the one of [2].

If the difference is too big, then the remaining old agents try to make adjustments. While they use for “learn” the approach involving as seed strategy their own current strategy, that the first time they execute “learn” is the strategy that worked well in the old team, we treat them differently than the new agent in one aspect. Since there is a certain chance that they should not change their strategy at all (only a few of them might have to adapt), we do not always use the result of “learn” as the new strategy of one of these agents. Instead we first make a simulation run with the old strategy and measure the fitness of it with this run (or runs, if necessary). Then we compare this fitness with the fitness of the strategy generated by “learn” and only use this strategy if the fitness is better than the fitness of the old strategy.

This additional criterion does not only help to detect agents that should not change their strategy at the moment (although they can change later, after some others have changed their strategies and the particular agent comes again to the point of learning), it also allows us to use smaller  $p_{seed,co}$ -values, if difference in abilities and task to fulfill require this. Naturally, different agents can use different  $p_{seed,co}$ - and  $p_{new,co}$ -values, we can even decrease the  $p_{seed,co}$ -value over time to introduce more and more flexibility, although this can have the team missing opportunities, again.

## 5 Experimental Evaluation

In OLEMAS, we have chosen to use Pursuit Games (see [1]) as application domain for testing our learning and adaptation approaches. The general idea of these games is rather simple: in a world consisting of connected grids, several hunter agents have to cooperate to catch one or several prey agents. A step of the game serves to establish a discrete time line and usually each agent performs one action in a step, except for actions that require several steps to be finished, in which case the action shows effect in the last step of the execution span. The game runs only for a limited number of steps, which puts pressure on the hunters (and the learning approaches) to get their job together.

Different versions of the game can be constructed by varying the world, the number and type of agents involved, by adding obstacles, having different start positions for the agents, or by having different definitions of catch, to name just the most important features. The type of an agent is of special interest to us, because it allows for different abilities of agents. The subfeatures leading to different types are agent shape and agent actions together with the speed with

Figure 1. Start situations of the variants

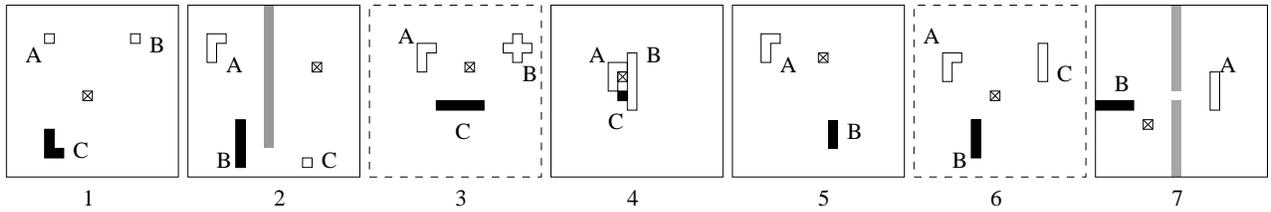
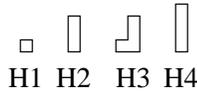


Figure 2. Replacement agents' shapes



which an agent can perform an agent (i.e. the execution span). By changing the shape, adding or removing actions or by just changing the speed for one or several actions we can generate a lot of different replacement scenarios.

In our experiments, we first defined a variant, learned a good set of strategies for the initial agents by applying the basic off-line learning approach from Section 2 and then defined a replacement of one agent by a new one. For these replacements, we looked at two basic types of difference in abilities: in one type (variants 1 to 4), we substituted an agent by a new agent missing at least one ability needed by the old agent to fulfill its role in the team (and not providing any new abilities that could make up for this). For the other type (variants 5 to 7), we either added new abilities or substituted one ability by another that would allow for a better team performance, if appropriately used. Before we look more closely at our results, we first present the variants and replacements. The start situations of the initial variants are depicted in Figure 1. Figure 2 presents agent shapes we used as replacements.

In all variants, the goal of the hunters was to immobilize the prey. If not stated otherwise, all agents could move in all 8 directions and agents that have a shape different from occupying just one grid can turn left and right. Also, if not stated otherwise, each action, including “learn”, takes 1 step. In variant 1, we substituted agent C by an agent of shape H1. This results in the team not being able to catch the prey on just a single border, which is the usual end situation the experienced team achieves. The new agent requires to make the catch in one of the corners, which is not within the original strategies of the remaining members of the team. In variant 2, we replaced hunter B by a hunter with the same shape but without the turn actions. As a result, the new agent cannot pass the obstacle. So, again, the usual end situation of B and C blocking the prey into the corner is not possible anymore. In fact, now agent A has to take a role in this and the new agent has to move out of the way.

In variant 3, we replaced hunter C with an agent of type H2 that required 3 steps for a turn action. As a result, the new agent was not fast enough to allow for the usual cooperation scheme of A and B “scaring” the prey and driving it towards C to make the catch. Letting A and B adapt resulted in first having them pull the prey up and then driving it down when the new agent is ready. But, due to being on an infinite grid world and the use of random decisions in evolutionary learning, only 80 percent of the runs were successful (which explains the high mean number of steps in Table 1). For variant 4, the original team had it easy, agents A and B just had to move down with the prey. Our change regarding C was rather radical, because we totally removed this agent (so, we were losing all abilities, even the possibility to block the prey). By allowing A and B to adapt, they still managed to catch the prey (much faster than starting from scratch would achieve).

In variant 5, the experienced team corrals the prey into the top right corner. After replacing B by an agent of shape H3, which allows the two agents to catch the prey everywhere, letting A adapt allows for catching the prey earlier and rather close to the middle, thus speeding up fulfilling the task. Variant 6 is a more complex variant than 5 with the same basic idea: the experienced team needs all 3 agents to catch the prey near the middle. By exchanging agent B by an agent of shape H3, C is not necessary anymore and A and the new agent can close the trap earlier. Letting A and C adapt leads to this improvement. Finally, in variant 7, agent B drives the prey through the obstacle and catches it with A in the top right corner, because the additional action of a turn slows it down too much, so that the experienced team did not see this as an option. If we replace B by an agent that already is turned (shape H4), the hunters can catch the prey between the obstacles. While Table 1 shows that this is not always happening, nevertheless it happens often enough to improve the average number of steps needed.

The results of Table 1 were produced by having the new agent use a seed strategy with  $p_{seed} := p_{seed,start} = p_{seed,co} = p_{seed,mul} = 60$  and  $p_{new,co} = 30$ . If the team is also allowed to adapt to the new agent, then this was started after 5 steps for variant 3 and 4, 10 steps for variant 1 and 25 steps else (these are the values for  $t_{initial}$  and we had to vary them due to the different variants and their requirements). As already mentioned, we kept  $t_{test}$  at 1. For each

Table 1. Experimental results

Var	new agent only			with team adapting			$p_{seed}$
	best	worst	mean	best	worst	mean	
1	–	–	–	20	37	33.3	25
2	–	–	–	49	113	77.7	25
3	–	–	–	18	–	211.9	85
4	–	–	–	22	27	24.3	60
5	31	55	40.8	21	43	31.3	60
6	14	29	24.1	12	25	21.5	85
7	61	89	83.4	61	81	75.1	40

variant, we performed 10 runs (without adaptation of the rest of the team and with adaptation, each) and we report the number of steps needed to fulfill the given task for the best and the worst run and the mean value over all runs (due to having random decisions in the GA, the learned strategies, resp. the achieved behavior, differ from run to run).

If we look at variants 1 to 4, then it is obvious that having the rest of the team adapt to the new agent after some time saves the day, if the new agent simply cannot fulfill the role the old agent had in the team. All  $p_{seed}$ -values we tried out (we always set  $p_{new,co} = 100 - p_{seed}$  and used the same values for all agents from the old team) allowed for this, but as can be seen, the best observed value is rather different for different game variants (as we expected from our experiences in [2]). Coming up with methods to suggest good values based on game variant and difference in abilities will be important future work.

In the cases of the replacements with agents offering new abilities, the picture is not as clear as for the first 4 variants. While the usage of a seed strategy is aimed at keeping the amount of “trying out” of strategies low, nevertheless learning always means that things have to be tried out. While in the runs without adaptation of the old agents we still had the new agent perform “learn” (although it usually realized that no change in strategy is necessary and therefore used the strategy of the old agent) so that in both kinds of runs we had the same loss of steps due to learning, having the old agents adapt still leads to trying out things and later having to do corrections. And this means normally more necessary steps to fulfill the given task. That nevertheless we achieved better mean values for the number of steps needed shows that the adaptation allows for using the improved abilities of the new agent, because only these abilities are making the difference. Again, the  $p_{seed}$ -values that were best varied and we had some values for each variant that did not show improvements.

All in all, our experiments show that using a seed strategy in on-line learning for agents that already performed well before a change to the team was made complements the use of the strategy of the replaced agent as seed strategy for the new agent. If the new agent alone cannot make the necessary adjustments to be successful, the adaptation by the remaining old agents steps in and achieves

the success. If the new agent has new, better abilities, the adaptation of the remaining team members offers a way to use these abilities, which having only the new agent adapt cannot achieve.

## 6 Conclusion

We presented a continuation/improvement of our work in [2] that deals with having a member of an experienced team replaced by a new agent with different abilities. Instead of having only the new agent trying to adapt the strategy of the old agent by performing on-line learning with a seed strategy, after some time also the remaining agents are allowed to adapt using this concept. Our experiments show that in case of new agents with less abilities (or agents losing abilities) having the old agents adapt allows for success where the new agent alone totally fails. If the new agent has additional abilities, then having the old agents adapt allows to really make use of these abilities instead of just sticking to the strategy of the old agent.

The general problem of agent replacements with different abilities still has not been addressed very much. In addition to our work, the closest approach was presented in [6], but this approach was not in a team setting and it was more interested in imitating an agent (and not replacing it and just getting its strategy). So, there are still many remaining issues that have to be addressed in the future, like finding measures for the difference in abilities to allow for suggestions of good parameter values for our method, or measuring the difference of tasks.

## References

- [1] M. Benda; V. Jagannathan and R. Dodhiawalla. An Optimal Cooperation of Knowledge Sources, Technical Report BCS-G201e-28, Boeing AI Center, 1985.
- [2] J. Denzinger and S. Ennis. Being the new guy in an experienced team – enhancing training on the job, Proc. AAMAS-02, ACM Press, 2002, pp. 1246–1253.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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.