

On Customizing Evolutionary Learning of Agent Behavior

Jörg Denzinger and Alvin Schur

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

Abstract. The fitness function of an evolutionary algorithm is one of the few possible spots where application knowledge can be made available to the algorithm. But the representation and use of knowledge in the fitness function is rather indirect and therefore not easy to achieve. In this paper, we present several case studies encoding application specific features into fitness functions for learning cooperative behavior of agents, an application that already requires complex and difficult to manipulate fitness functions. Our experiments with different variants of the Pursuit Game show that refining a knowledge feature already in the fitness function usually does not result in much difference in performance, while adding new application knowledge features to the fitness function improves the learning performance significantly.

1 Introduction

Over the last few years, research into learning behavior of agents has intensified, resulting in a large variety of learning concepts for different agent architectures and different application areas. While research into reinforcement-based approaches has focused a lot on possible solutions to the credit assignment problem and consequently the learning function, evolutionary learning approaches covered a large variety of goals related to agent, resp. multi-agent, specific features. These goals included the development of evolutionary algorithms for rather different agent architectures, like the use of genetic programming to learn real agent control programs (see [8]), co-evolutionary approaches on the level of nodes in neural networks (see [10]), or genetic algorithms for sets to learn prototypical situation-action pairs that are used together with the nearest-neighbor rule for action selection (see [3]).

Additionally, other research looked at how various kinds of knowledge can be incorporated into the learning process, ranging from using knowledge that allows a hierarchical learning approach (see [12]), to knowledge defining tasks of growing complexity (see [7]), to transferring knowledge from one agent to another with different abilities (see [2]). Also the ability to communicate and in fact to learn communications to express appropriate knowledge has been explored (see [9]). But one of the key ways to include knowledge into an evolutionary search has not been looked at in more detail: the fitness function. The reported research either just stated a fitness function that could do the job, or the researchers used

very general fitness measure ideas that can be instantiated rather differently and then provided a single instantiation. While outside of learning of behavior there has been quite some research into fitness functions (see [6] for a rather general method), for learning of cooperative behavior there is still little advice available for incorporating application specific knowledge into the fitness function.

In this paper, we present case studies regarding adding more knowledge into the fitness function used within the OLEMAS system (see [4]) that has as application domain a huge variety of Pursuit Games (see [3]). The fitness function in OLEMAS started out with the goal to be applicable for all game variants OLEMAS provided. However, learning complex behaviour was slow. Previous work concentrated on evaluating the influence of the control parameters (see [5]) or on improving the genetic operators in certain team scenarios (see [2]). But these works still showed room for improvements.

A lot of the problems that were observed deal with not representing certain knowledge, resp. features, specific to the game variant: orientation of agents, the blocking effect of obstacles and other agents or the particulars of the goal of the game. Our experiments show that representing such key game variant features in the fitness function improves the learning performance compared to a standard general purpose fitness function. While this seems to indicate that learning of cooperative behavior for a human developer just transfers the task of developing good cooperating strategies into the –more obscure– task of developing a fitness function containing the appropriate guiding knowledge, we can also report that the way feature knowledge is represented does not produce big changes, as demonstrated by exploring different ways to measure the base distance between two points. And the knowledge we integrate into the fitness in our case studies still applies to a rather large set of game variants, so that the gains by its use are wide spread.

2 Learning with SAPs and the NN Rule

In this section, we present our evolutionary learning method. To do this, we first present the agent architecture we are using and then we discuss learning using a GA on sets.

2.1 Our Agent Architecture

In general, an agent can be seen as a triple (Sit, Act, Dat) with a set Sit of situations, a set Act of actions, and a set Dat of values of internal data areas. The agent uses its perception of the actual situation and the values of the internal data areas to decide upon the next action. Therefore an agent Ag can be seen as a function $f_{Ag}: Sit \times Dat \rightarrow Act$. For an agent in a multi-agent system we can divide the sets into parts concerning the agent itself and parts concerning the other agents. More formally, an element s of Sit has the form $s = s_{Env}s_{Ag}$, where s_{Env} describes the environment the agent acts in without any information about other agents and s_{Ag} provides this information about other agents. Act can be divided into the sets Act_{own} of the agent's actions not dealing with other

agents and Act_{co} of its communication and cooperation actions. The internal data areas of the agent can belong to three different sets, namely the set Dat_{own} of values of data areas concerning information about the agent *itself*, the set Dat_{sag} of values of data areas concerning *sure* knowledge about other agents, and the set Dat_{usag} of values of data areas containing *unsure* knowledge about the other agents.

The agent architecture we will use for our agents is rather simple. The data areas (Dat_{own}) of an agent holds a set of prototypical *situation-action-pairs*, SAPs, (i.e. $Dat_{own} = 2^{Sit \times Act}$) and its decision function f_{Ag} uses the nearest-neighbor rule to determine the action to the actual situation. More precisely, for all SAPs (s_i, a_i) in the actual value of Dat_{own} the distance of s_i to the actual situation s with respect to a distance measure f_{dist} is computed (in our case f_{dist} computes the Euclidean distance of two number vectors). Then the action of the pair with the smallest distance (or greatest similarity to the current situation) is chosen (i.e. $a = a_j$ with $f_{dist}(s_j, s)$ is minimal). Note that we also can use *enhanced* situation descriptions, in which a situation is not just an element of Sit , but can contain additional data about the agent. In general, this agent architecture allows only for rather reactive agents.

2.2 Evolutionary Learning of SAPs

The evolutionary learning approach used in [3] and [4] focuses on evolving suitable SAP-sets. For all learning agents, a genetic algorithm evolves their set of SAPs, their *strategy*. An individual of the population maintained by the GA consists of several sets of SAPs, one set for each agent. As usual, we use crossover and mutation as genetic operators. The mutation operator just deletes an arbitrary number of SAPs in a strategy and then adds some randomly generated new SAPs. For each agent within an individual, the crossover operator randomly selects SAPs from both parents and puts them together to form the offspring.

The fitness computation for the evolutionary learning of cooperative behavior is based on a restricted simulation of the multi-agent system which measures the agent's behavior. The simulation is restricted because the number of steps that the agents are allowed to do is limited. In all other regards the simulation is identical to the task. The result of a single simulation run is called the behavior B . Starting from situation s_0 for a set of strategies for the learning agents Ag_1, \dots, Ag_k we get $B(Ag_1, \dots, Ag_k, s_0) = s_0, s_1, \dots, s_{i-1}, s_i, \dots$. If the task can be solved by the agents represented by the individual, let's say with situation s_i , then the number of steps, i , is the fitness measure. Else, a measure of how near the agents came to doing the task is used. Since we are measuring the behavior of the agent strategies, each situation produced during a run should have some influence on the fitness. Naturally, the measure has to be based on the problem to solve and therefore can only be given for a concrete application. We will do this in the next section. If there are random effects occurring during simulation runs, the fitness measure should take into account several simulation runs, to deal with these random effects. And as usual, the genetic algorithm favors fitter individuals for use by the genetic operators.

3 Pursuit Games and the OLEMAS System

In order to evaluate the evolutionary learning method presented in the last section, we build the OLEMAS system, that uses Pursuit Games as the application domain. Pursuit Games as a testbed for multi-agent systems and the cooperation of agents were first suggested in [1]. Since then, many authors have used variants for testing their approaches. The basic idea of all variants is that one or several hunter agents have to catch one or several prey agents within a world that usually is represented as a collection of connected grids. Time in the world is expressed as a sequence of turns. When used as a testbed for learning agents, usually the hunters (or some of them) are the learning agents, although some authors also looked at learning prey agents.

The following features of a Pursuit Game can be varied in the OLEMAS system (see [3]):

1. The form of the grid

In the basic scenario described in [1] the grid-world has no boundaries and there are no obstacles in it. OLEMAS allows for variants by introducing boundaries (e.g., a $N \times N$ grid) or obstacles (with varying shapes). Obstacles are treated as agents (a kind of innocent bystanders) and they are even allowed to move.

2. The individual hunter

In the basic scenario a hunter agent does not have many features. But we can obtain variants with respect to the following sub-features:

a) *Shape and size*

A hunter may occupy one or more grid squares. It may be rectangular, but it can also have other shapes. OLEMAS allows any set of connected points (occupying a grid) for the agent's shape.

b) *Possible moves and actions*

Besides moving only in the vertical or horizontal direction (or not moving at all) variants include diagonal moves or rotations, if rotations actually have an effect. In theory, there can also be communication actions but so far we do not have a communication action in OLEMAS.

c) *Speed*

Hunters, like all agents, have with each action associated a number of game turns that are needed to perform the action.

d) *Perception and communication capabilities*

An aspect that greatly influences the strategy of a hunter (and therefore each solution attempt for the game) are its perception and communication capabilities. (Note that being able to see the other hunters is a kind of visual communication.) The spectrum of this aspect ranges from hunters with no or very limited communication capabilities to hunters utilizing sophisticated communication methods. In OLEMAS, we allow for hunters to either see only the prey agents or all agents.

e) *Memory capabilities*

An aspect that can become important if the hunters are able to communicate with each other is the memory of an agent. Memory allows

an agent to remember plans and intentions of other hunters. There may be no memory, a restricted size for the memory, or an arbitrary amount of memory. In OLEMAS, memory can only be introduced in form of extended situations.

3. The hunting team

For most variants of the game more than one hunter (and cooperation between the hunters) are required for the team to succeed. Therefore, the composition of the team of hunters is also an aspect that can be varied.

a) *The number of hunters*

For each combination of the other aspects there is a minimal number of hunters needed to win the game. Deploying more hunters may help to win the game, but may also require different, possibly more sophisticated strategies and more effort in developing these strategies.

b) *The type of the hunters*

Since there can be different types of hunters, quite different strategies –depending on what kind of hunters form the team– are needed to cooperate in order to win.

4. The prey

The prey is an agent like the hunters. Therefore the same sub-features a) to e) apply with the exception that communication is only necessary if there are several prey agents. But there is an additional sub-feature:

f) *The strategy of the prey*

The (escape) strategy of the prey is the main factor determining the difficulty to win the game. Strategies range from simply moving in one direction to random moves to elaborate strategies like maximizing the distance from all hunters, obstacles and boundaries. OLEMAS also allows for using a set of SAPs and the nearest-neighbor rule as the strategy for a prey.

5. The start situation

The start positions of both hunters and prey can also influence both the possibility to win the game and the effort for learning a cooperative strategy for the hunters. If the game is always started from the same positions and no random element is introduced by other aspects, then a winning strategy will always win (and is easier to learn). Otherwise, different start situations will lead to different outcomes.

6. The game goal

Even for the definition of if and when the game is won there are different variants. The main question is to “capture” or to “kill”. The prey is captured if it cannot move to another grid anymore (i.e., it is totally surrounded by boundaries, obstacles and hunters). If a hunter occupies the same grid as the prey (at some point in time), the prey is killed. With several prey agents, the number of caught prey agents has to be defined within the game goal. Usually, the goal also includes resource limitations, i.e. the number of turns T_{Max} within which the goal has to be fulfilled.

With so many features to vary, there is an enormous number of game variants and an important goal in [3] was to define situation descriptions for SAPs and

a fitness measure that allows a hunter to learn winning strategies for many of them. For a human observer, the situations occurring during a game run are characterized by the positions and orientations (if an agent occupies more than one grid) of all agents on the grid field. The internal representation of a situation makes this more precise by stating the positions of the agents in a predefined order and by stating a position as the coordinates of the so-called centerpoint of each agent (that is a property of the agent). For a situation in a SAP in the strategy of a particular learning agent, we transform the objective game situation into the subjective view of the learning agent by presenting the coordinates of the other agents relative to the learning agent. This also allows us to filter out agents that the learning agent is not allowed to see according to the particular game variant.

As described in the last section, if the fitness fit of an individual $\mathcal{I} = Ag_1, \dots, Ag_k$ is based on evaluating b different runs r_1, \dots, r_b , then we have

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

The elemental fitness $efit$ of a run r_j with the behavior $s_{j0}, s_{j1}, \dots, s_{j, T_{Max}}$ is defined as

$$efit(r_j, \mathcal{I}) = \begin{cases} l, & \text{success in } l \leq T_{Max} \text{ steps} \\ \sum_{t=1}^{T_{Max}} success(s_t), & \text{in case of failure,} \end{cases}$$

and the function $success$ that measures how near a particular encountered game situation s_t is to a winning situation is approximated in OLEMAS by summing up the Manhattan distances of all hunters from all preys. The Manhattan distance $dist_{Manh}$ between two agents at coordinates (x_1, y_1) and (x_2, y_2) is defined as

$$dist_{Manh}((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

We also use the Manhattan distance in the nearest-neighbor rule to measure the similarity of two situations. The Manhattan distances between the coordinates of an agent in both situations is computed for all agents that are visible and these distances are summed up to produce the similarity measure. If the similarity to the current situation is the same for two or more situations in the set of SAPs forming the agent strategy, then the action of the first SAP is taken.

4 Putting More Knowledge into the Fitness Function: Case Studies

As already stated, the fitness function has always been considered as an appropriate place in an evolutionary algorithm to add knowledge that someone has about the problem he or she wants to solve. But many authors also reported on the difficulties of incorporating more knowledge into the fitness function. The knowledge has to be represented rather indirectly to be used to better distinguish

good individuals from bad ones. This brings the risk of focusing too narrowly, thus guiding the algorithm to only local optima or to deadends without solutions.

In case of our evolutionary learning, the fitness function does not measure individuals directly, it measures the cooperative behavior that is induced by the individuals, thus making any knowledge representation even more indirect and more prone to divert from the main objective. By defining the elemental fitness *efit* of a run of an individual based on the success evaluation of all encountered situations, we already introduced a more direct way to incorporate knowledge by concentrating on what is good or bad in a particular situation. But there are many aspects of a situation that might be of interest regarding how near to success we are and usually there are different ways to measure a particular aspect.

If we look at the base fitness function defined in the last section, the key aspect is the distance between agents. To measure distance, we used the Manhattan distance, which can be easily computed from the coordinates of the agents. But there are additional, more accurate measures of distance. Will their use improve our learning? Also, agents with shapes and grids with obstacles add an additional dimension to the concept of distance. If I am near to someone else if I ignore things like obstacles, then I can still be far away from the other agent if I cannot fly and have to walk around the obstacle. And, depending on my shape, spaces between obstacles might be too small to pass through or require rotations at the right times. And what about the goal of the pursuit game? If we have to catch several prey agents, should this not be reflected in how we measure being near to a goal-fulfilling situation? In the following, we will present modifications of our base fitness function that reflect these ideas of additional knowledge and we will evaluate the usefulness of these new fitness functions, compared to the base fitness, with pursuit game variants that benefit from using the additional knowledge.

4.1 General Remarks

All of the experiments we present in the following subsections report on changes we made to the function *success*, defined in Section 3, to reflect certain knowledge about game features. Often, these changes are adding a certain evaluation criterion to the base *success*-function.

Whenever we combined several criteria $C_1(s), \dots, C_m(s)$ for a situation s , we first normalized the individual criteria and then used weighting factors w_1, \dots, w_m to control the influence of the criteria (although, due to lack of space, we will report on one weight combination only). So, the general combination function *success-comb* we used is defined as follows:

$$\text{success-comb}(C_1, \dots, C_m)(s) = \sum_{i=1}^m w_i * \text{norm}(C_i(s))$$

A criterion is normalized by dividing its value by the maximum possible value for the criterion. However, for many of the criteria we developed, this maximum value is difficult to determine a priori or the values occurring during a run are

far from the theoretical maximum, thus requiring quite some fiddling with the criterion weight to get the influence of the criterion right. Therefore we decided to use the criterion value of the start situation as the value for normalization, if the criterion is not limited to only a few possible values.

For all of our experiments, we report on the success rates of the learner and average number of turns required by the successful strategy. Since an evolutionary algorithm uses random decisions, every run of the algorithm is different, so that we have to report a success rate (in percent). Naturally, the average number of turns needed by a strategy is computed for the successful runs only. We also provide the standard deviation we observed among the successful runs with regard to turns needed by the hunters using the learned strategies to win the game. We report on the average number of generations needed to find a winning strategy only if there was a substantial difference between the different fitness functions.

In all of the experiments, the agents can move in all eight directions and, if they have an irregular shape, they can rotate 90 degrees to the left and right, unless otherwise stated. Each action takes one turn to accomplish. All game variants are played on a 30 by 30 grid world and the hunters have 200 turns to win the game. If there is only one prey, then the game goal is to kill it. In all learning runs, we stop when the first strategy winning the game is found.

4.2 Does More Precision Help?

The base function used for function *success* is based on measuring the distance between hunters and preys using the Manhattan distance $dist_{Manh}$. This is not the most primitive way to determine a distance, but also not the most accurate way. To understand how different distance computations influence the results, we tested two other ways, namely $dist_{max}$ that uses the maximum of the coordinate differences in the x- and y-axis:

$$dist_{max}((x_1, y_1), (x_2, y_2)) = \max\{|x_1 - x_2|, |y_1 - y_2|\}$$

and $dist_{Eucl}$ that uses the Euclidean distance between the coordinates of the two agents:

$$dist_{Eucl}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The game scenario we used to come up with the results from Table 1 is as follows: We have a simple scenario with one hunter and one prey, see Variant 1 in Figure 1 for the start situation. The prey tries to avoid the hunter and all borders. The parameters for the GA were a population size of 100 for a maximum of 10 generations with a mutation rate of 15 percent and 20 individuals surviving from generation to generation. The strategies for the hunter could contain at most 20 SAPs. We did 100 learning runs per distance function.

The results in Table 1 are somewhat surprising. While more precision in the distance evaluation allowed for 5 percent more success, producing solutions of approximately the same quality, being more crude in measuring the distance

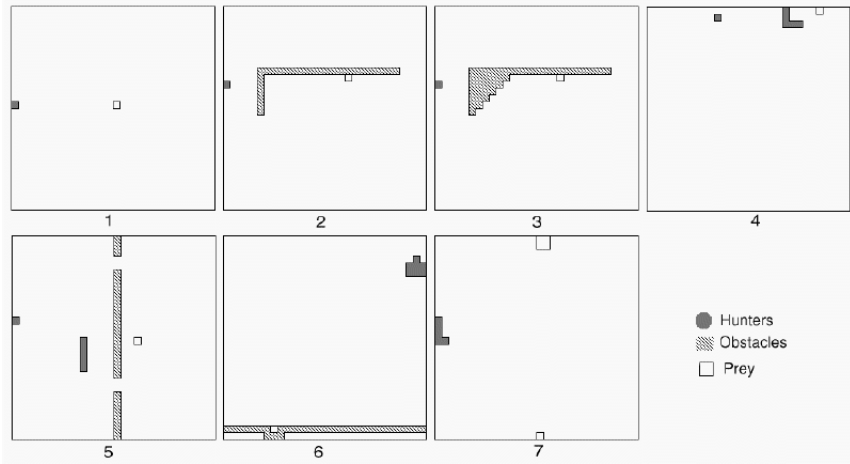


Fig. 1. Start situations for game scenarios

Table 1. Experimental results for different distance measures

Distance function	Success rate	Steps	Std. deviation (Steps)
$dist_{Manh}$	79%	58.6	23.6
$dist_{max}$	90%	56.2	19.2
$dist_{Eucl}$	84%	57.7	24.6

gives us an increase of 11 percent and more stability with regard to the quality of the solution. So, here we have an example where more precision in measuring knowledge about a feature does not really help.

4.3 Not Every Grid-Square Should Count the Same!

For Pursuit Game variants where agents occupy just one grid-square and where there are no obstacles, using any of the distance measures we looked at in the last subsection works well. The picture changes drastically if we have obstacles in the game variant and/or agents have shapes with a lot of blocking potential. Since getting by an obstacle or other agent often means having to go away from the preys, the base fitness of measuring how near the hunters are to the preys often leads the learning agents into a local optimum. Only with a lot of luck, such as a combination of several SAPs that together manage to get completely around the obstacle, can the learning still be successful. In [4], we showed that the problem can be reduced by using on-line learning, but obstacles in the way still is a basic problem in many game variants with regard to learning.

In order to deal with such game variants, the fitness function must account for obstacles and other agents between the learning agent and the prey, making

these situations less good than situations where the path to the prey is clear. Obviously, this means that we have to look at the paths from the hunters to the preys and have to measure grid-squares on these paths according to them being occupied or not. For a grid-square gs we define its *terrain value* v_{terr} as

$$v_{terr}(gs) = \begin{cases} H & \text{if } gs \text{ is occupied by a moving agent} \\ O & \text{if } gs \text{ is occupied by an obstacle} \\ E & \text{if } gs \text{ is not occupied} \end{cases}$$

In our experiments, we used $E = 1$, $H = 10$, and $O = 15$.

Then, the terrain distance $dist_{terr}$ for a sequence gs_1, \dots, gs_k of grid-squares is defined as

$$dist_{terr}(gs_1, \dots, gs_k) = \sum_{i=1}^k v_{terr}(gs_i).$$

Naturally, the question remains how to get the sequences of grid-squares between two agents, since there are several possibilities. In our experiments, we tested three possibilities related to the Manhattan distance:

$dist_{terr,x}((x_1, y_1), (x_2, y_2))$: starts at (x_1, y_1) moves along the x-axis to (x_2, y_1) and then moves along the y-axis to (x_2, y_2)

$dist_{terr,y}((x_1, y_1), (x_2, y_2))$: starts at (x_1, y_1) moves along the y-axis to (x_1, y_2) and then moves along the x-axis to (x_2, y_2)

$dist_{terr,xy}((x_1, y_1), (x_2, y_2))$: $dist_{terr,x}((x_1, y_1), (x_2, y_2)) + dist_{terr,y}((x_1, y_1), (x_2, y_2))$

The reason for the different ways of generating the grid sequence becomes clear if we take a look at the start situations depicted in Figure 1 for variants 2 to 5. If we compare variants 2 and 3, we can see that the distance values differ quite a bit already in the start situations depending on whether we go first right or first up. The game variants 2 to 5 are as follows. In variants 2 and 3 we want to focus on getting the hunter around the obstacle. Therefore the prey does not move at all. Note that the hunter will not be successful within the given number of turns if it tries to pass above the obstacle, since it has to go a very long way (and away from the prey for quite a number of steps). For both variants the GA used 100 individuals for a maximum of 10 generations. We used a mutation rate of 25 percent and a strategy had at most 20 SAPs. The results in Table 2 are based on 100 learning runs for each fitness function.

In variant 4, we have two hunters that both learn. The L-shaped hunter is not allowed to rotate. The prey evades the nearest hunter and thus ends up in the upper right corner of the world. The problem of the hunters is to figure out that the small hunter has to pass the L-shaped one, because otherwise the L-shaped hunter ends up “protecting” the prey in the corner. This is not so easy to learn and therefore we used 200 individuals per generation and allowed for a maximum of 30 generations. Also, we used a mutation rate of 30 percent and increased the

Table 2. Experimental results with terrain fitness

Terrain fitness		Var. 2	Var. 3	Var. 4	Var. 5
$dist_{Manh}$	Success rate	65%	55%	77%	10%
	Steps	29.2	28.7	34.6	56.7
	Std. deviation (Steps)	8.3	9.1	21.7	16.7
$dist_{terr,x}$	Success rate	86%	82%	100%	37%
	Steps	29.1	29.8	39.0	46.8
	Std. deviation (Steps)	7.9	7.3	26.9	15.8
$dist_{terr,y}$	Success rate	84%	82%	100%	27%
	Steps	29.2	27.4	28.6	41.8
	Std. deviation (Steps)	9.6	9.5	13.6	9.3
$dist_{terr,xy}$	Success rate	88%	92%	100%	37%
	Steps	30.0	29.0	30.6	54.9
	Std. deviation (Steps)	9.6	9.5	17.0	19.2

maximum number of SAPs per individual to 30. As a consequence, the runtime of the learner went up and we report only on the results of 30 learning runs per function.

Variant 5 uses the same parameters for the GA as variant 4. Also the prey strategy is the same. The bigger hunter is not learning, it just moves towards the prey. It is not allowed to rotate. The small hunter is the learning hunter and the problem it has to deal with is that the larger hunter blocks the upper passage through the obstacle while moving towards the prey. Therefore it has to learn to pass through the lower passage. So, variant 5 combines the difficulties of the other 3 variants.

As Table 2 shows, using all 3 terrain distance measures improves the success rate of the learning drastically. The combination of the two ways to generate paths, $dist_{terr,xy}$, always has the highest success. For variants 2 and 3, the average number of steps and its standard deviation are approximately the same. For variant 4, using $dist_{terr,x}$ results in worse strategies than the default measure, while $dist_{terr,y}$ finds consistently shorter solutions. But the combined terrain fitness is only a little bit worse than that. By using $dist_{terr,y}$, the cooperative strategy of the two hunters that lets the bigger hunter move down to let the smaller one pass is more encouraged than by $dist_{terr,x}$, that requires the big hunter to move down more before the advantage becomes obvious to the fitness measure. Even the standard distance measure can take better advantage of that. But in our experiments with variant 4 we also observed that for all terrain measures the number of generations necessary to find a solution was less than half of the number required by the default measure, which shows that the awareness of the blocking potential also can improve the learning time. For variant 5, the success rate of $dist_{terr,y}$ is 10 percent less than the other terrain measures, but those strategies that were found are consistently better than the ones generated by all other measures.

All in all, using a distance measure that distinguishes terrain features to indicate blocking potential increases the chance for success of the learner. The path we use to calculate the terrain value has some influence on both success rate and the quality of the found solutions. If we are willing to combine the measures for different paths, the problem of influence of the path can be overcome by spending more computing time.

4.4 If You Have a Shape Then Orientation Matters!

Another problem that comes up when we have agents with irregular shapes is that usually we require these agents to orient themselves into a certain direction (by rotating), so that their shape fits right into what is required for winning the game. If a hunter tries to rotate too late in the game, it might not be possible anymore due to obstacles blocking a rotation. Then we can fall into a local optimum without success. One could argue that making the fitness function aware about what the right orientation for a hunter is, is already giving away part of the solution to win the game and this is definitely true. But obstacles might require rotations to be passed by and, as our experiments show, it is not so easy to express the “right” orientation via the fitness function. In addition, our experiments also highlight that choosing the centerpoint of an agent and its initial orientation can have rather dramatic effects on the performance, regardless of what we do with the fitness function.

In OLEMAS, when defining an agent, we define its shape, indicate within the shape the centerpoint of the agent and we define a preferred *front* of the agent, which is the side of the agent initially given as the top of its shape. For indicating within a fitness function, resp. the *success*-function within the fitness function, how a hunter agent h is oriented with regards to a prey p , we define the orientation criterion C_{orient} as follows. We partition the whole grid world into 4 sections that are defined based on the centerpoint of the hunter by two diagonal lines passing through this centerpoint. The front section is the section of the world that is faced by the front of the agent, the sections to the right and left are obviously the right and left section and the final section is called the rear section. Then we have for a situation s :

$$C_{orient}(h, p, s) \begin{cases} 0 & \text{if centerpoint of } p \text{ is in front section} \\ 1 & \text{if centerpoint of } p \text{ is in right or left section} \\ 2 & \text{if centerpoint of } p \text{ is in rear section} \end{cases}$$

This criterion is normalized by dividing by 2 and we combine it with the criterion defined by using $dist_{Manh}$.

Variant 6 in Figure 1 shows the starting situation for the Pursuit Game variant we have chosen to illustrate the influence of the criterion C_{orient} . As in variants 2 and 3, the prey does not move, thus forcing the hunter to rotate to achieve the right orientation, so that its “nose” can kill the otherwise protected prey. The GA parameter settings were similar to variants 2 and 3, we only changed the maximum number of generations to 30. Since we measure the distance between two agents from centerpoint to centerpoint, already the placement

Table 3. Experimental results with orientation

Fitness		Var. 6a)	Var. 6b)	Var. 6c)
$dist_{Manh}$	Success rate	36%	73%	75%
	Steps	27.3	25.6	25.4
	Std. deviation (Steps)	4.1	2.3	4.0
	Nr. of generations	15.5	13.9	11.7
C_{orient} , with $w_{orient} = 1$	Success rate	68%	92%	77%
	Steps	30.8	29.0	26.1
	Std. deviation (Steps)	9.8	8.6	4.8
	Nr. of generations	15.7	11.1	12.2
C_{orient} , with $w_{orient} = 10$	Success rate	55%	79%	53%
	Steps	42.2	29.8	37.6
	Std. deviation (Steps)	17.6	9.5	22.4
	Nr. of generations	22.2	16.8	18.4
C_{orient} , with $w_{orient} = 100$	Success rate	45%	65%	48%
	Steps	39.2	30.9	35.8
	Std. deviation (Steps)	13.2	11.6	17.7
	Nr. of generations	20.1	17.9	19.5

of this centerpoint should have some influence on the outcome and therefore we tried out 3 different placements, namely

- a) in the hunter's nose
- b) in the center of the hunter's body and
- c) on the hunter's back directly opposite to the nose.

As Table 3 shows, the placement of the centerpoint has already quite some effect on the success rate and the learning time for the default fitness alone. The wrong placement results in half of the success rate and approximately 4 more generations needed to find the successful strategies. Adding the orientation criteria with equal weight as the distance criterion improves the success rate for each of the centerpoint placements, but helps especially in the case of a bad placement. While the learning times are comparable, the additional success is paid for by worse solution quality that varies much more than without the orientation criterion, except for the case c) where distance alone is already quite good. Increasing the influence of the orientation criterion does not improve the help by orientation, in fact it results in longer learning times and worse solution quality. Especially if the centerpoint placement is already good for the standard fitness, an increase of the influence of C_{orient} results in a deterioration of the learning performance.

All in all, placement of the centerpoint has shown to be potentially crucial for the success of learning. By including orientation into the fitness, we can make the influence of centerpoint placement less crucial and we can improve the success rate in general.

Table 4. Experimental results with prey clustering

	Success rate	Steps	Std. deviation (Steps)
C_{prey} , with $w_{prey} = 0$	14%	43.9	14.9
C_{prey} , with $w_{prey} = -0.1$	20%	55.8	20.8
C_{prey} , with $w_{prey} = -1$	29%	48.5	13.3
C_{prey} , with $w_{prey} = -10$	41%	51.9	16.7

4.5 Complex Catches Require More Complex Evaluations!

With certain Pursuit Game variants it is possible to mimic some problem aspects of many other application areas for learning agent behavior. *Herding* is one of these application areas (as suggested in [11]). As a Pursuit Game variant, bringing prey agents together into a herd can be achieved by having as a game goal that the hunter(s) kill all the prey, which means that the hunter(s) must, in the same turn, occupy for each prey at least one of the grids occupied by the prey agents. If we have only one hunter (or less hunters than prey), then the knowledge we should convey to the learner is that it is good to have prey agents stay together.

Since the hunters have to “overlap” prey agents to achieve the game goal, we found a simple distance measure for preys, similar to what we used for variant 1, not sufficient. Instead we assigned to each prey a so-called *control radius* that extends around it from the farthest grid occupied by the prey to all sides. For two prey, p_1 and p_2 , we define their x-overlap and y-overlap in a situation s as follows:

x-overlap (p_1, p_2, s) = number of grid squares along the x-axis in which the control radii of p_1 and p_2 overlap,

y-overlap (p_1, p_2, s) = number of grid squares along the y-axis in which the control radii of p_1 and p_2 overlap.

And then we have as the overlap:

$$\mathbf{overlap}(p_1, p_2, s) = \mathbf{x-overlap}(p_1, p_2, s) \times \mathbf{y-overlap}(p_1, p_2, s)$$

With this we can define the prey criterion $C_{prey}(s)$ in a situation s by summing up the overlap of each pair of prey agents. This is normalized by dividing by $2 \times radius + 1$ with a *radius* of 15 used in our experiments. Since the more overlap the better, we use as w_{prey} negative values.

To evaluate C_{prey} , we used a game scenario with start situation 7 in Figure 1. The big prey evades just the hunter, while the smaller one in addition tries to stay away from the borders. The GA parameters are similar to variant 6, except that we allow for 40 SAPs in an individual.

As Table 4 shows, the additional C_{prey} criterion increases the success rate substantially, it nearly triples it, if we use $w_{prey} = -10$. This comes with the price of finding longer solutions, but this is definitely acceptable. So, taking knowledge about the game goal into account again results in an improvement.

5 Conclusion and Future Work

We presented several case studies showing how knowledge about application features can be integrated into an evolutionary learner for cooperative behavior of agents. Our experiments showed that awareness of a certain feature substantially increases the success of the learner (assuming limited resources for the learning), while increased precision of the knowledge already used is not guaranteed to improve the learning.

Obviously, there are many more features of groups of Pursuit Game scenarios that can be useful for an evolutionary learner and that we want to look into in the future. Our interest will be in finding features that are not typical for just a few game variants but useful in many of them, as has been the terrain fitness. Naturally, we also are interested in identifying additional ways to include knowledge into learning behavior. And it will be interesting to see if there are other “hidden” features, like the placement of the centerpoint, that influence learning performance.

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’00, 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. G. Eiben and J. van Hemert. SAW-ing EAs: Adapting the Fitness Function for Solving Constrained Problems, in: Corne, Dorigo, Glover (eds.): *New Ideas in Optimization*, McGraw-Hill, 1999, pp. 389–402.
7. F. Gomez and R. Miikkulainen. Incremental Evolution of Complex General Behavior, *Adaptive Behavior* 5, 1997, pp. 317–342.
8. T. Haynes, R. Wainwright, S. Sen and D. Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies, Proc. 6th GA, Morgan Kaufmann, 1995, pp. 271–278.
9. K.-C. Jim and C.L. Giles. Talking helps: Evolving communicating agents for the predator-prey pursuit problem, *Artificial Life* 6(3), 2000, pp. 237–254.
10. J.D. Schaffer, D. Whitley and L.J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art, Proc. COGANN-92, IEEE, 1992, pp. 1–37.
11. A. Schultz, J. Grefenstette and W. Adams. Robo-shepherd: Learning complex robotic behaviors, *Robotics and Manufacturing: Recent Trends in Research and Application*, Volume 6, ASME Press, 1996, pp. 763–768.
12. P. Stone. *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*, MIT Press, 2000.