Experiments in Learning Prototypical Situations for Variants of the Pursuit Game

Jörg Denzinger, Matthias Fuchs
Center for Learning Systems and Applications (LSA)
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: {denzinge|fuchs}@informatik.uni-kl.de

Abstract

We present an approach to learning cooperative behavior of agents. Our approach is based on classifying situations with the help of the nearest-neighbor rule. In this context, learning amounts to evolving a set of good prototypical situations. With each prototypical situation an action is associated that should be executed in that situation. A set of prototypical situation/action pairs together with the nearest-neighbor rule represent the behavior of an agent. We demonstrate the utility of our approach in the light of variants of the well-known pursuit game. To this end, we present a classification of variants of the pursuit game, and we report on the results of our approach obtained for variants regarding several aspects of the classification. A first implementation of our approach that utilizes a genetic algorithm to conduct the search for a set of suitable prototypical situation/action pairs was able to handle many different variants.

1 Introduction

Designing a set of agents and an organization and interaction scheme for them in order to solve a given problem cooperatively is not an easy task. Even if the designer does not have to cope with additional restrictions like the use of already existing software, interface restrictions, or also involved fellow designers, the task remains difficult and its realization is often very expensive. Especially in cases where the designer is not familiar with the given problem, the first design typically follows well-known and general principles, and has to be evolved over several versions until a satisfactory version is found. These versions reflect the learning process of the designer. The more versions are needed, the more expensive the task becomes.

An idea that has become more and more interesting and therefore a research goal of many computer scientists is to integrate learning and adaptation capabilities into the first design of agents in order to let them evolve the intended behavior automatically (and at much cheaper costs). The idea of providing an agent with learning capabilities, and using it as a start design for a range of problems has a strong potential, although some basic methods and representations concerning the problem domain should—not only in our opinion- be provided by a specialist of the domain. This concept is a basis for the growing field of genetic programming ([10]), but also to some degree applies to areas like software engineering in form of software generators, or multi-agent systems as a means for the reuse of agents or multi-agent platforms.

In this paper we present an agent architecture based on the classification of situations with the nearest-neighbor rule (NNR), and a learning mechanism based on the generation of prototypical situations that demonstrate the utility of the idea to have agents adapt automatically to problems in order to evolve cooperative behavior. The agents in our approach ground their behavior on a set of pairs of a situation and an action (sequence). When an agent is confronted with a situation it determines the situation/action pair (S/A pair) in its set of S/A pairs whose situation is the most similar to the given situation according to the NNR. Then it applies the action (sequence) associated with the selected pair. Since the behavior of such an agent can be easily changed by modifying, adding, or removing S/A pairs, this architecture provides a suitable basis for learning and adaptation.

Learning cooperative behavior in this context means searching for an appropriate set of prototypical S/A pairs. In order to realize this search, we have chosen a genetic algorithm (GA; [7], [9]) that allows us to proceed without much knowledge besides an apt representation of situations and a comparison procedure for sets of S/A pairs in order to determine the fitter set. We show the utility of our approach by learning agents for variants of the pursuit game.

2 The Basic Agent Architecture

An agent exposed to some (multi-agent) environment has to repeat the process of taking some action A when being confronted with a certain situation S in order to continuously modify its situation so as to achieve some goal (as efficiently as possible). Commonly, a situation S is represented by a vector of n variables from the set $\Re$ of real numbers. Hence, $\Re^n$ is the set of all situations respectively the *situation space*. Each component x_i of a situation $S = (x_1, \ldots, x_n) \in \Re^n$ essentially expresses a certain feature of S. Given $\mathcal{A}$ as the set of all possible actions (respectively action sequences) an agent may take, the behavior of an agent is reflected by its strategy $\Psi : \Re^n \to \mathcal{A}$ that allows the agent to select an action $\Psi(S) = A \in \mathcal{A}$ when being confronted with situation S.

There are many ways to realize the behavior of an agent, i.e., a mapping from $\Re^n$ to $\mathcal{A}$. We propose here to utilize a finite, non-empty set $\mathcal{I} = \{(S_i, A_i) \in \Re^n \times \mathcal{A} \mid 1 \leq i \leq m_\mathcal{I}\}$ of prototypical S/A pairs as basis for an agent's behavior. Given a situation $S \in \Re^n$, an agent takes the action A_j which is associated with the situation S_j—i.e., $(S_j, A_j) \in \mathcal{I}$—that is closest (respectively the most similar) to S among all situations in $\mathcal{I}$. More precisely, we employ a distance measure $\mathcal{D} : \Re^n \times \Re^n \to \Re$ (typically the Euclidean distance measure that we also employed for our experiments). Given the minimal distance $\delta_{min} = \min(\{\mathcal{D}(S_i, S) \mid 1 \leq i \leq m_\mathcal{I}\})$ between situations S_i in $\mathcal{I}$ and the given situation S, the action selected by strategy $\Psi_\mathcal{I}$ is $\Psi_\mathcal{I}(S) = A_j$, where $j = \min(\{1 \leq i \leq m_\mathcal{I} \mid \mathcal{D}(S_i, S) = \delta_{min}\})$ is the smallest index of situations in $\mathcal{I}$ that are closest to S. (The smallest index is chosen merely to resolve ambiguities.)

Approaches based on the NNR fascinate with their simplicity. Nevertheless (or because of that) they have shown considerable success in connection with instance-based learning ([1]) and classification problems (e.g. [14]). Applications of the NNR commonly involve a set $\mathcal{I}$ which is (a subset of) a set of *given* training examples. We intend here to *evolve* the set $\mathcal{I}$ with the help of a GA in order to optimize the performance of an agent employing strategy $\Psi_\mathcal{I}$. Since evolution can be equated with search, the GA is only one among many search methods that can be utilized. We chose the GA, because it has practical and theoretical properties that suit our approach. Alternative search methods can be investigated in future research.

Regardless of the search method employed, the basic architecture or design of a strategy determines (a) how good a strategy we can expect to find and (b) how difficult the search is going to be. Point (a) surely favors the use of neural nets, because a neural net can realize (almost) any mapping provided that it is equipped with an apt topology and the connection weights are chosen appropriately. But evolving weight configurations and possibly also evolving net topology is a strenuous task even for the GA (cp. [21]). Considering the fact that neural nets are "over-sophisticated" for many applications, simpler designs entailing more manageable search spaces are called for. Similar considerations apply to genetic programming ([10]).

Classifier systems ([7]) can—by design—profit from the GA much more easily. Also, they are a very powerful methodology. But the rather complex architecture of a classifier system and the sophisticated techniques involved (e.g. credit assignment, bidding, etc.) make it worthwhile thinking about simpler methods, in particular in connection with problems that do not require the full power of a classifier system.

The use of condition/action rules is a very popular approach in this context (e.g., [6]). Commonly, such a rule is of the form "`if` $a_1 \leq x_1 \leq b_1 \wedge \ldots \wedge a_n \leq x_n \leq b_n$ `then` *execute* $A \in \mathcal{A}$", where each condition $a_i \leq x_i \leq b_i$ specifies the range of values for variable x_i (w.r.t. this rule). In a set of rules, the rules are (implicitly) or-connected. Hence, such a set of rules can subdivide the situation space $\Re^n$ into hyper-rectangles which are aligned with dimension axes. These hyper-rectangles may be partially overlapping, which calls for disambiguation techniques, and there may be gaps that necessitate some kind of default rules.

The major limitation of these condition/action rules is the restricted way they can subdivide the situation space. An approach based on a set of prototypical S/A pairs and the NNR as described above allows for subdividing the situation space without overlapping and gaps, where the possible sub-spaces include, but are not limited to hyper-rectangles. As a matter of fact, such an approach can piecewise-linearly approximate arbitrary sub-space boundaries.

In [20], both approaches are combined. But the GA is only applied to evolve sets of condition/action rules, which then provide the NNR based component with S/A pairs. We couple here the GA and the NNR approach tightly, meaning that the GA is immediately responsible for the set $\mathcal{I}$ of S/A pairs.

Before presenting the details of the GA designed for our approach, we would like to emphasize the simplicity of our approach: Once the situation space is known, the fundamental "data structure" is a situation respectively a point in the situation space and an associated action. Hence, the basic architecture of a strategy—namely a set of such S/A pairs—is at least as easy to handle (by a GA) as a set of condition/action rules, though more expressive.

3 Learning with the GA

The preceding sections repeatedly pointed out that we intend to evolve respectively search for a strategy (behavior) $\Psi_\mathcal{I}$ respectively a finite set $\mathcal{I}$ of S/A pairs. So, the search space we have to cope with is the set of all finite sets of S/A pairs. Even if we limit the number of S/A pairs of each set $\mathcal{I}$ to some arbitrary but fixed $1 \leq M \in \mathbb{N}$, i.e., $m_\mathcal{I} \leq M$ for all $\mathcal{I} = \{(S_i, A_i) \mid 1 \leq i \leq m_\mathcal{I}\}$, the search space in general remains enormous (and unstructured). The use of a GA appears to be appropriate under these circumstances, because the GA has the potential to cope with intricate search spaces in the absence of any knowledge about their structure. Furthermore, a GA is less prone to getting trapped in a local optimum. Both properties are highly valuable for our purpose. In the sequel, we describe the basics of the GA in the light of our application.

Unlike other search methods, the GA maintains a set of (sub-optimal) solutions, i.e., several points in the search space. In this context, a solution is preferably called an *individual*, and the whole set is referred to as a *population* or *generation*. Usually, the size of the population is fixed. In order to explore the search space, the GA applies so-called *genetic operators* to (a subset of the) individuals of its current population. This way, new individuals can be created and hence new points in the search space can be reached. In order to keep the population size fixed, it must be determined which individuals are to be eliminated in order to make room for the new ones. For this purpose a so-called *fitness measure* is employed which rates the fitness (i.e., the ability to solve the problem at hand) of each individual of the current population. The genetic operators are applied to the most fit individuals, this way producing offspring which then replaces the least fit individuals (*"survival of the fittest"*).

So, the GA basically proceeds as follows: Starting with a randomly generated initial population, the GA repeats the cycle comprising the rating of all individuals using the fitness measure, applying the genetic operators to the best individuals, and replacing the worst individuals with offspring of the best, until some termination condition is satisfied (e.g., an individual with a satisfactory fitness level has been created).

In our case an **individual** $\mathcal{I}$ corresponds to a strategy represented by a (finite) set of S/A pairs. The **fitness** of an individual is assessed in terms of the problem solving expertise of its associated strategy $\Psi_{\mathcal{I}}$. The fitness measure is the only connection between the GA as an underlying search method and the actual problem. Therefore, we cannot define a specific fitness measure at this point, and we have to postpone details until a concrete problem is specified (see section 5).

The **initial population** of the GA is (as usual) generated completely at random. That is, for each individual $\mathcal{I}$ of the initial population comprising n_{pop} individuals, both its size $m_{\mathcal{I}} \leq M$ and the $m_{\mathcal{I}}$ S/A pairs are determined at random.

Two **genetic operators** are employed here, namely *crossover* and *mutation*. ('Reproduction' as given by [10], for instance, is basically realized by the survival of the r% fittest individuals.) Offspring is produced through crossover, while mutation can modify this offspring. The **crossover operator** randomly selects two distinct parents from the pool of r% best (surviving) individuals. A subset of the S/A pairs of each parent individual is chosen at random, and the union of these two subsets yields the "child" individual. (Note that the surviving individuals here simply all have an equal chance to become reproductive.) The **mutation operator** modifies individuals stemming from crossover before they are admitted to the next generation. An individual $\mathcal{I}$ is subject to mutation with probability P_{mut}. If an individual $\mathcal{I}$ actually is selected for mutation, each S/A pair of $\mathcal{I}$ is—with probability P_{rnd}—replaced by a randomly generated pair.

4 The Pursuit Game

In order to demonstrate the potential of our approach to learning cooperating agents we need problems that require different degrees of cooperation and different behavior of agents. But these problems also have to be easy to understand (which does not imply that they can be solved easily as well), and should allow for comparisons regarding several criteria, e.g., complexity, necessary cooperation or solubility. Therefore it would be best to have one basic problem that allows for variants regarding several aspects.

Fortunately, there is such a basic problem, namely the so-called *pursuit game* (also called "hunter and prey") that was first presented in [2]. Since then, a number of quite different approaches to solving this basic problem (see, for example, [17], [12]) and also some variants (see [5], [18], [19]) have been proposed, so that the pursuit game can be called the "blocks world" of the DAI community.

The basic problem can be described as follows: There are four hunter agents (also called predators) and a prey agent on an infinite rectilinear grid. The game is played in discrete time steps. At each time step, an agent can move one square in a horizontal or vertical direction, or stay put. The prey selects its steps randomly. The hunters win the game, if they can surround the prey. No two agents may occupy the same square. The task is to develop a strategy for the hunters that enables them to win the game.

There are several aspects of this basic scenario that can be varied. In the following we will take a closer look at all aspects, and we will examine more closely some combinations of several aspects that either have been investigated in literature or will be investigated in section 5.

1. **The form of the grid**
   In the basic scenario the grid-world has no boundaries and there are no obstacles in it. Variants regarding this aspect can be created by introducing boundaries (e.g., a $N \times N$ grid) or obstacles (with varying shapes).
2. **The individual hunter**
   In the basic scenario a hunter agent does not have many features. Therefore one can obtain variants with respect to the following sub-aspects:
   a) *Shape and size*
      A hunter may not only occupy one square of the grid, but several of them. It may be quadratic, but it can also have other shapes.
   b) *Possible moves and actions*
      Besides moving only in the vertical or horizontal direction (or not moving at all) variants can include diagonal moves or turns (rotations), if turns actually have an effect. There can also be communication actions (see d)).

c) *Speed*
A hunter does not have to be as quick as the prey. It can be faster, but it can also move more slowly than the prey.

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.

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.

3. **The hunting team**
For most variants of the game more than one hunter (and cooperation between the hunters) are required so that there is a chance 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 (according to aspect 2), 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-aspects a) to e) apply with the exception that communication is only necessary if there are several prey agents (as was suggested in [13]). But there is an additional sub-aspect:

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 (which can be quite successful, see [8]) over random moves to elaborate strategies like maximizing the distance from the nearest or all hunters. Even learning strategies to counter those of the hunters was suggested and tried out (with perhaps too much success, see [8], again).

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 goal**
Even for the definition of if and when the game is won there are two variants. The main question is to "capture" or to "kill". The prey is captured if it cannot move to another square anymore (i.e., it is totally surrounded by boundaries, obstacles and hunters). It is killed if the prey and a hunter occupy the same square (at some point in time). The goal may also include resource limitations.

For describing an actual variant of the pursuit game it is necessary to choose one of the possible instantiations for each aspect and sub-aspect. Obviously, the possibilities to win are quite different for different variants.

Most of the work regarding the pursuit game so far centered on strategies for particular variants that were developed by human beings, naturally with some emphasis on the cooperation (using communication) of the hunters. In [18] and [19] the authors concentrated on communication-intensive strategies for the hunters. Diagonal moves were presented in [11]. (See also [15].)

But variants of the pursuit game were also used as a testbed for approaches to learning cooperative agents. In [8] a genetic programming approach was applied to variants originating from variations of the escape strategy of the prey. Even experimental data concerning a co-evolution of hunters and prey was provided. Genetic programming results in programs that are more expressive than our approach, but more expertise on the parts of the designer is required, because not only a suitable representation of situations must be found, but also the building blocks of programs (functions and terminals) must be chosen properly.

Another work dealing with learning cooperative behavior and the pursuit game is described in [13]. There, a genetic algorithm approach was chosen to learn strategies for variants with $n + 4$ hunters and n prey agents (where n was varying). In contrast to our approach, in [13] a genetic algorithm is used to search good parameters for a program that includes memory and planning functions. Hence, this approach does not provide as high a level of abstraction as our S/A pairs.

5 Experiments

This section documents the experimental setting and the results we obtained with our approach for several variants of the pursuit game. In order to describe the setting we have to describe how to represent situations, actions, and—for the learning part—we describe the fitness measure ϑ of the GA.

5.1 Representing Situations and Actions

The representation of situations is the main factor that determines whether suitable agent strategies can be learned, how good they will be, and how difficult the learning process will become. Clearly, there are different representations even for a single variant of the pursuit game, and also clearly we have to expect that different variants need quite different representations. In fact, the representation is the only possibility for a designer to influence the evolving strategy (which was exactly our goal).

In our experiments we have chosen to use representations for situations that differ as little as possible from variant to variant (which kind of emulates an unexperienced designer and puts quite a burden on our learning approach). Due to this reason (but also due to lack of space and due to the efficiency of our implementation that can only be seen as a first, very crude version) we restricted our experiments to only a few of the aspects of section 4. These aspects are the strategy of the prey, start situation, goal, possible moves, number of hunters, and perception capabilities. All other aspects were fixed: There is just one prey. All $k \geq 1$ hunters are of the same type, i.e., they occupy one square, have the same possible moves, move at the same speed as the prey (one move per time unit), and have no memory capabilities. The prey always knows the position of all hunters, and in most experiments we used a 30×30 grid. Furthermore, we decided to have all hunters use the same strategy $\Psi_{\mathcal{I}}$ (which is reasonable because the hunters are not distinguishable). Thus, learning amounts to searching for one suitable set $\mathcal{I}$ of prototypical S/A pairs.

Our basic representation of a situation is based on the positions of the agents regarding a distinguished reference square $(0,0)$ employing the usual Cartesian coordinate system. Hunter i occupies square $(x_i, y_i) \in \mathbb{Z} \times \mathbb{Z}$, and the prey occupies square $(x, y) \in \mathbb{Z} \times \mathbb{Z}$. ($\mathbb{Z}$ is the set of all integers.) We characterize the situation of hunter i with the help of its relative positions regarding prey and fellow hunters. That is, the situation S_i of hunter i is a vector consisting of $x - x_i$, $y - y_i$ and, if the hunters see their fellow hunters, $x_j - x_i$, $y_j - y_i$ for $1 \leq j \leq k$ $(i \neq j)$ sorted according to j. Hence, the situation space is $\mathbb{Z}^{2k}$, or $\mathbb{Z}^2$ if the hunters see only the prey. (On a $N \times N$ grid, $\mathbb{Z}$ can be restricted to $\{-N+1, \ldots, N-1\}$. See also [4].) This representation can be enhanced with hunter types, hunter orientation (if it is not quadratic), coordinates of obstacles, and memory fields (for more information on other hunters or for history information). The possible moves for both hunters and prey are either N, S, E, W or stay put, or these moves plus the diagonal moves NE, NW, SW, and SE. (If communication actions are possible, they also have to be added.) Hunters and prey execute their moves simultaneously. (See [4] for details on conflict resolution.)

In our experiments, the possible instantiations for the varying aspects are: The hunters either see each other and the prey, or only the prey. (Thus we have to establish cooperation without "proper" communication; see [16].) The start situation is either fixed (prey in the center of the 30×30 grid, hunters lined up along the whole western boundary with equal distance from each other; a single hunter starts from the middle of the western boundary), or chosen at random (prey in the center, each hunter at least 5 squares away from it in both horizontal and vertical direction). These start positions are also used in case of an infinite grid, but naturally the grid boundaries are ignored after setting up the initial situation. The goal situations are killing or capturing the prey. The number of hunters is the minimal number that is required to win the game (theoretically). (If hunters see fellow hunters, we always deploy $k \geq 2$ hunters.) There will also be some remarks on the effect of additional hunters in subsection 5.3.

Finally, there are five escape strategies $\Phi_1, \ldots, \Phi_5$. Note that, to our knowledge, Φ_4 and Φ_5 are novel. In particular Φ_5 proved to be quite challenging (cp. subsection 5.3). Besides precluding moves that violate some rules or restrictions, the prey also does not consider moves that lead to a square currently occupied by a hunter, although the hunter might move away from that square. The remaining moves are referred to as eligible moves. Each escape strategy (except for Φ_2) centers on a different vector of distance values $(d_1, \ldots, d_m)$, $m \geq 1$, that reflects the current situation of the prey from a certain point of view. Common to all these strategies is that the prey tries out (simulates) all eligible moves and then picks the best move (tested first). Distance vectors are compared using the usual lexicographic ordering $>_{lex}$ (left-to-right comparison with $>$), where "greater" means "better". If diagonal moves are allowed, then the basic distance measure is the Euclidean distance. The "Manhattan distance" is employed otherwise. In the sequel, let $\hat{d}_i$ be the basic distance between prey and hunter i.

- Φ_1: *Maximize distance from nearest hunter*: $m = 1$, $d_1 = \min(\{\hat{d}_1, \ldots, \hat{d}_k\})$.
- Φ_2: *Random movement*: Among all eligible moves the prey chooses one move at random. Staying put is not an eligible move here. Consequently, the prey does not stay put unless trapped.
- Φ_3: *Maximize sum of distances from all hunters*: $m = 1$, $d_1 = \hat{d}_1 + \cdots + \hat{d}_k$.
- Φ_4: *Maximize vector of sorted distance values*: $m = k$, $(d_1, \ldots, d_k)$ is obtained by sorting $\hat{d}_1, \ldots, \hat{d}_k$ so that $d_i \leq d_{i-1}$.
- Φ_5: Escape strategy Φ_5 is an extension of Φ_4. In addition to $\hat{d}_1, \ldots, \hat{d}_k$, the prey here also takes into account its smallest distance from each of the four grid boundaries. Consequently, $m = k + 4$. By also trying to maximize the distance from grid boundaries, escape strategy Φ_5 alleviates a "flaw"

of escape strategy Φ_4 which (like Φ_1 and Φ_3) essentially makes the prey run away to some grid boundary where it can be captured or killed much more easily than it can be when trying to stay in "open field" as long as possible. (Obviously, Φ_5 cannot be applied to an infinite grid-world.)

5.2 The Fitness Measure ϑ

The fitness measure ϑ is to rate the performance of an individual $\mathcal{I}$, i.e., a set of S/A pairs. The performance of $\mathcal{I}$ is given by the suitability of the associated strategy $\Psi_{\mathcal{I}}$ employed by each hunter. In case random effects can occur, it is clear that the fitness cannot be determined reasonably based on merely one trial (hunt). Consequently, several trials have to be executed, and the outcomes of these *elementary fitness values* must be combined to obtain the *overall fitness* ϑ.

The *elementary fitness measure* θ rates a single trial. θ should therefore reflect *if* the hunters were successful in that particular trial, and in that case, how fast they succeeded. In case of failure, θ should somehow express how close to success the hunters came. Naturally, the hunters cannot be granted an arbitrary amount of time. If they do not succeed within $T = 200$ time steps, then their hunt is judged a failure. The elementary fitness measure $\theta(\mathcal{I}) \in \mathbb{N}$ is defined as follows.

$$\theta(\mathcal{I}) = \begin{cases} t_s, & \text{success in } t_s \leq T \text{ steps} \\ \sum_{t=1}^{T} \sum_{i=1}^{k} \delta(i,t), & \text{in case of failure,} \end{cases}$$

where $\delta(i,t)$ is the Manhattan distance that separates hunter i from the prey at time step t. Hence $\theta(\mathcal{I})$ is the smaller the fitter $\mathcal{I}$ is considered to be.

The overall fitness $\vartheta(\mathcal{I})$ of $\mathcal{I}$ is computed in a straight forward way on the basis of $b \geq 1$ trials which result in b elementary fitness values $\theta_1(\mathcal{I}), \ldots, \theta_b(\mathcal{I})$:

$$\vartheta(\mathcal{I}) = \frac{1}{b} \cdot \sum_{i=1}^{b} \theta_i(\mathcal{I}).$$

If no random effects occur, we set $b = 1$, and $b = 20$ else. An individual $\mathcal{I}$ is considered as being successful only if *all* b trials were successful. If two individuals have the same overall fitness measure, then the more concise individual is preferred.

For our experiments, the parameters of the GA were chosen as follows (cp. section 3): $r = 30\%$, $P_{mut} = 50\%$, $P_{rnd} = 10\%$, $n_{pop} = 100$. The maximal number M of S/A pairs of an individual $\mathcal{I}$ was restricted to 30. (All these settings were determined after a few preparatory experiments and are in no sense "optimal".) The maximal number of generations (cycles) of the GA was limited to 100. The GA stopped before exceeding this limit as soon as a successful individual was found. For each variant of the pursuit game the GA was run 5 times, and the performance of the best individual that surfaced in (one of) these 5 runs is presented. This 'best-of-5' policy is reasonable, because experiments corroborated that several shorter runs are more promising than a single long run (cp. [10]). (We picked the number '5' more or less arbitrarily.) For details on the performance of the GA see [4].

5.3 Results

Tables 1–4 present the core of our experimental results for a 30×30 grid. (We will also report on other experiments that do not fit into these tables.) Each table and its caption specify the particular variant of the pursuit game. The column labeled with 'k' shows the number of hunters. The columns labeled with 200, 1000, 10000 display the performance of the 'best-of-5' individual when granted the respective number T of time steps. (Note that $T = 200$ during learning.) The performance is given as the number of steps necessary to succeed if no random effects can occur, and as a success rate that was determined based on 100 trials otherwise.

Being granted more time steps T only pays off in connection with random escape strategy Φ_2. There, the hunters essentially chase the prey, waiting for the prey to make a certain sequence of moves that allows them to succeed. Naturally, the probability for this to happen increases with the number of time steps. However, by allowing diagonal and hence more moves, this probability decreases, which is reflected by the sometimes significant worse performance of the hunters when dealing with a prey using Φ_2 and diagonal moves compared to the case where no diagonal moves are allowed. For all other strategies, failure is almost always caused by some kind of "deadlock", so that increasing T does not improve the success rate.

Tables 1 and 2 show that the theoretically minimal number of hunters almost always suffice to succeed respectively to achieve an acceptable success rate (if granted sufficient time) even though the hunters only focus on the prey (*emergent behavior*). The only significant failure (0% success rate) occurs when 2 hunters starting from random positions attempt to capture the prey that tries to escape using either Φ_3 or Φ_4, and diagonal moves are not allowed (see table 2). With 3 hunters the success rate is ca. 100% in both cases.

From a theoretical point of view it is clear that hunters that see both prey and fellow hunters are at least as versatile as hunters that see only the prey. But taking into account fellow hunters enlarges the situation space and thus complicates the search space. This is the main reason why we sometimes obtained worse results in practice although better results have to be expected in theory. Nonetheless, considering fellow hunters sometimes leads to significant improvements (possibly because it is indispensable). This happened, for instance, in connection with the case discussed above where 2 hunters focusing on the prey had a success rate of 0%; 2 hunters that also see each other performed significantly better (cp. tables 2 and 4). We could make this observation again when tackling a variant related to the original version of the pursuit game. Only 4 hunters hunting a prey using Φ_2 on an infinite

Table 1: Hunters see only the prey, and their objective is to "kill".

	Fixed Start Positions								Random Start Positions							
	No Diagonal Moves				Diagonal Moves				No Diagonal Moves				Diagonal Moves			
Φ	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000
Φ_1	2	33	33	33	2	28	28	28	2	100%	100%	100%	2	100%	100%	100%
Φ_2	1	100%	100%	100%	1	100%	100%	100%	1	96%	99%	100%	1	99%	100%	100%
Φ_3	2	28	28	28	2	28	28	28	2	100%	100%	100%	2	100%	100%	100%
Φ_4	2	44	44	44	2	28	28	28	2	100%	100%	100%	2	99%	99%	99%
Φ_5	2	54	54	54	2	26	26	26	2	99%	99%	99%	2	100%	100%	100%

Table 2: Hunters see only the prey, and their objective is to "capture".

	Fixed Start Positions								Random Start Positions							
	No Diagonal Moves				Diagonal Moves				No Diagonal Moves				Diagonal Moves			
Φ	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000
Φ_1	2	56	56	56	3	28	28	28	2	100%	100%	100%	3	96%	96%	96%
Φ_2	2	55%	100%	100%	3	5%	22%	72%	2	10%	81%	100%	3	3%	37%	92%
Φ_3	2	56	56	56	3	28	28	28	2	0%	0%	0%	3	100%	100%	100%
Φ_4	2	59	59	59	3	33	33	33	2	0%	0%	0%	3	88%	88%	88%

grid that see each other achieved an acceptable success rate (55% when being granted 10000 time steps and starting from fixed positions).

Furthermore, a prey using the quite sophisticated escape strategy Φ_5 appears to be very hard to capture. First of all, a prey using Φ_5 tries to stay away from boundaries and corners. As a matter of fact, it cannot be forced to move into a corner or next to a boundary in any way. Consequently, 4 hunters are required to capture the prey if diagonal moves are not allowed (8 hunters else). In our experiments, 4 hunters only succeeded when they knew about their fellow hunters. (See [4] for more details.)

We also examined the effects a surplus of hunters can have. On the one hand, it is understandable that more hunters have a better chance to win the game. On the other hand, due to the fitness measure, all hunters will chase the prey and hence might hinder each other, in particular when they only focus on the prey. (In order to favor more sophisticated pursuit strategies involving "driving" and "ambushing" the fitness measure has to be adapted appropriately.) In our experiments, increasing the number of hunters (up to 7) did not have any significant negative effects (except for complicating the search space if fellow hunters were to be considered, of course).

6 Discussion

We have presented an approach for learning cooperative behavior of reactive agents that is based on the search for a set of prototypical situation/action pairs. An agent employs the nearest-neighbor rule to select that pair whose prototypical situation is closest (most similar) to the actual situation, and then performs the associated action. In our current implementation, the search is conducted by a genetic algorithm.

Our goal was to demonstrate that this approach is considerably versatile in that it allows a designer of multi-agent systems to specify requirements on a very high level in terms of a representation of situations and possible actions (and a comparison of strategies in our case), and then a satisfactory solution is evolved automatically. We could show that this goal can be achieved in many cases: We presented aspects of the well-known pursuit game that can be varied so as to obtain variants of the game, and for many of these (non-trivial) variants the first implementation of our approach succeeded in evolving apt strategies.

However, our experiments also suggested several improvements. Although it was surprising how well our ad hoc implementation of the genetic algorithm performed, for problems involving a more complex representation of situations improvements are necessary. The use of additional knowledge (apart from the knowledge integrated with the fitness measure), alternative search methods (possibly combined with the genetic algorithm), and enhanced efficiency through distributed search seem to be profitable. The TEAMWORK approach to distributed search ([3]) is capable of achieving these goals, nevertheless providing a high level of abstraction for the designer.

A comparison of our approach with other (learning) approaches in connection with the pursuit game is at the time rather difficult, because other approaches mostly have been examined with respect to a few particular variants of the game. Moreover, in most cases the experimental setting cannot be reproduced, because full information on all aspects of the game is not available. But this is essential in order to obtain a reliable comparison, because (as our experiments have shown) tiny variations of the setting can have significant effects.

Table 3: Hunters see both prey and fellow hunters, and their objective is to "kill".

	Fixed Start Positions								Random Start Positions							
	No Diagonal Moves				Diagonal Moves				No Diagonal Moves				Diagonal Moves			
Φ	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000
Φ_1	2	29	29	29	2	28	28	28	2	100%	100%	100%	2	100%	100%	100%
Φ_2	2	100%	100%	100%	2	99%	100%	100%	2	96%	100%	100%	2	100%	100%	100%
Φ_3	2	28	28	28	2	28	28	28	2	78%	78%	78%	2	98%	98%	98%
Φ_4	2	42	42	42	2	28	28	28	2	82%	82%	82%	2	99%	99%	99%
Φ_5	2	182	182	182	2	26	26	26	2	74%	74%	74%	2	100%	100%	100%

Table 4: Hunters see both prey and fellow hunters, and their objective is to "capture".

	Fixed Start Positions								Random Start Positions							
	No Diagonal Moves				Diagonal Moves				No Diagonal Moves				Diagonal Moves			
Φ	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000	k	200	1000	10000
Φ_1	2	56	56	56	3	30	30	30	2	83%	83%	83%	3	74%	74%	74%
Φ_2	2	53%	100%	100%	3	4%	18%	88%	2	50%	100%	100%	3	1%	3%	21%
Φ_3	2	57	57	57	3	28	28	28	2	11%	11%	11%	3	81%	81%	81%
Φ_4	2	61	61	61	3	28	28	28	2	99%	99%	99%	3	81%	81%	81%

References

[1] **Aha, D.W.; Kibler, D.; Albert, M.K.:** *Instance-Based Learning Algorithms*, Machine Learning 6, 1991

[2] **Benda, M.; Jagannathan, V.; Dodhiawalla, R.:** *An Optimal Cooperation of Knowledge Sources*, Technical Report BCS-G2010-28, Boeing AI Center, 1985

[3] **Denzinger, J.:** *Knowledge-Based Distributed Search Using Teamwork*, Proc. 1^{st} ICMAS, San Francisco, CA, USA, 1995, pp. 81–88

[4] **Denzinger, J.; Fuchs, M.:** *Experiments in Learning Prototypical Situations for Variants of the Pursuit Game*, Technical Report LSA-96-04E, University of Kaiserslautern, 1996, access via WWW at `http://www.uni-kl.de/AG-AvenhausMadlener/fuchs.html`

[5] **Gasser, L. ; Rouquette, N. ; Hill, R.W. ; Lieb, J.:** *Representing and using organizational knowledge in DAI systems*, in Distr. AI, vol 2 of Research Notes in AI, Pitman, 1989, pp. 55–78

[6] **Grefenstette, J.J.:** *Lamarckian Learning in Multiagent Environments*, Proc. 4^{th} International Conference on Genetic Algorithms, 1991, San Diego, CA, USA, pp. 303–310

[7] **Holland, J.H.:** *Adaptation in natural and artificial systems*, Ann Arbor: Univ. of Michigan Press, 2^{nd} edition, 1992

[8] **Haynes, T. ; Sen, S. ; Schoenefeld, D. ; Wainwright, R.:** *Evolving Multiagent Coordination Strategies with Genetic Programming*, submitted to Artificial Intelligence, access via WWW at `http://euler.mcs.utulsa.edu/~haynes/jp.ps`

[9] **De Jong, K.:** *Learning with Genetic Algorithms: An Overview*, Machine Learning **3**:121–138, 1988

[10] **Koza, J.R.:** *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1991

[11] **Korf, R.E.:** *A simple solution to pursuit games*, Working Papers of the 11^{th} Intern. WS on DAI, 1992, pp. 195–213

[12] **Levy, R. ; Rosenschein, J.S.:** *A Game Theoretic Approach to Distributed Artificial Intelligence and the Pursuit Problem*, in Decentralized AI III, Elsevier, 1992, pp. 129-146

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

[14] **Michie, D.; Spiegelhalter, D.J.; Taylor, C.C.:** *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, 1994

[15] **Osawa, E.-I.:** *A Metalevel Coordination Strategy for Reactive Cooperative Planning*, 1^{st} International Conference on Multi-Agent Systems, 1995, San Francisco, CA, USA, pp. 297–303

[16] **Rosenschein, J.S. ; Ginsburg, M. ; Genesereth, M.R.:** *Cooperation Without Communication*, Proc. AAAI-86, AAAI-Press, 1986, pp. 51-57

[17] **Singh, M.P.:** *The effect of agent control strategy on the performance of a DAI pursuit problem*, Working Papers of the 10^{th} Intern. WS on DAI, 1990

[18] **Stephens, L.M. ; Merx, M.B.:** *Agent organization as an effector of DAI system performance*, Working Papers of the 9^{th} Intern. WS on DAI, 1989

[19] **Stephens, L.M. ; Merx, M.B.:** *The effect of agent control strategy on the performance of a DAI pursuit problem*, Proc. 1990 Distributed AI Workshop, 1990

[20] **Sheppard, J.W.; Salzberg, S.L.:** *Combining the Genetic Algorithm with Memory-Based Reasoning*, Proc. 6^{th} International Conference on Genetic Algorithms, 1995, Pittsburgh, PA, USA

[21] **Zhang, B.-T.; Mühlenbein, H.:** *Genetic Programming of Minimal Neural Nets Using Occam's Razor*, Proc. 5^{th} International Conference on Genetic Algorithms, 1993, pp. 342–349