Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 1890–1897

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

Farmer-Pest Problem: A Multidimensional Problem Domain for Comparison of Agent Learning Methods

Bartłomiej Śnieżyński, Jacek Dajda, Marcin Mlostek, Michał Pulchny

AGH University of Science and Technology, Department of Computer Science, Krakow, Poland

Abstract

Learning is often utilized by multi-agent systems which can deal with complex problems by means of their decentralized approach. With a number of learning methods available, a need for their comparison arises. This paper presents initial comparison results for selected algorithms (SARSA, Naïve Bayes, C4.5 and Ripper), which are obtained based on the new multi-dimensional Farmer-Pest problem domain, which is suitable for benchmarking learning algorithms. The results show that supervised learning algorithms can be used to generate agent strategy. It appears that for simple environment reinforcement learning algorithm together with Naïve Bayes learning gives best results. Although, in a difficult environment, C4.5 and Ripper are the best.

Keywords: multi-agent systems, machine learning, supervised learning, reinforcement learning.

1. Introduction

Decentralized problem solving is a method to deal with problem complexity. One of the architectures, which can be used in such a manner is a multi-agent system. In complex or changing environments it is very difficult, sometimes even impossible, to design all system details a priori. To overcome this problem one can apply a learning algorithm, which allows to adapt the system to the environment.

To apply learning in a multi-agent system, one should choose a method of learning, which fits well to the problem. There are many algorithms developed so far. However, in multi-agent systems most applications use reinforcement learning or evolutionary computations.

Because various methods can be applied to generate agent strategy, they should be compared in a controlled environment to choose the best method. This paper presents a new domain called Farmer-Pest problem, which is designed to compare various agent learning methods. It is an optimization with feedback problem. It has multiple dimensions and therefore can be easily configured to better adapt to specific needs. This allows to compare learning methods in various conditions and settings, without violating the core ideas and rules of the problem.

Email addresses: Bartlomiej.Sniezynski@agh.edu.pl (Bartłomiej Śnieżyński), dajda@agh.edu.pl (Jacek Dajda)

Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.205

In this research, we make the following contributions to the state of the art: we propose the multi-dimensional domain allowing comparison leaning algorithms; we show that methods other than reinforcement learning can be used for strategy generation; we show that in specific conditions, supervised learning can be much faster that reinforcement learning.

In the following sections we overview existing environments for learning agents and describe the proposed problem domain. Next, the learning agent architecture is described followed by the presentation of the initial experimental results. These results show performance of learning agents using reinforcement learning, and three methods of supervised learning (Naïve Bayes classifier, C4.5 and Ripper). Finally, conclusions and the further work are outlined.

2. Environments for Learning Agents

Good survey of learning in multi-agent systems working in various domains can be found in [1] and [2]. Below three example problems are presented.

Very popular in multi-agent systems is a soccer domain. The environment consist of a soccer field with two goals and a ball. Two teams of simulated or real robots are controlled by the agents. The performance is measured by the difference of scored goals. In [3] genetic programming is utilized to learn behavior-based team coordination. In [4] C4.5 algorithm is used to classify the current opponent into predefined adversary classes. Reinforcement learning can be also applied. Good example is [5], where a "keep away" version of the domain is used. Agents learn when to hold the ball an when to pass it. It appears that this domain is difficult for the learning algorithms. The state space is large, uncertainty and long delays of action effects should be taken into account.

One of environments, which is often used in research is Predator-Prey domain. It is a simple simulation with two types of agents: predators, and preys. The aim of a predator is to hunt for a prey. Prey is captured if predator (or several predators if cooperation is tested) is close enough. In [6] predator agents use reinforcement learning to learn a strategy minimizing time to catch a prey. Additionally, agents can cooperate by exchanging sensor data, strategies or episodes. Experimental results show that cooperation is beneficial. Other researchers working on this domain successfully apply genetic programming [7] and evolutionary computation [8]. Results of rule induction application in this domain can be found in [9, 10].

Target observation is another interesting problem domain. Good example is [11], where rules are evolved to control large area surveillance from the air. In [12] Parker et al. present cooperative observation task to test autonomous generating of cooperative behaviors in robot teams. Agents cooperate to keep targets within specific distance. Lazy learning based on reinforcement learning is used to generate strategy better than random, but worse than a manually developed one. Results of application of reinforcement learning mixed with state space generalization can be found in [13].

3. The Farmer-Pest Problem

The Farmer-Pest problem borrows the concept from the specific aspect of real world, in which farmers struggle to protect their fields and crops from pests. Each farmer (this is the only type of agent in the problem) can manage multiple fields. On each field, a multiple types of pests can appear. Each pest has a specific set of attributes e.g. number of legs, color. Values of these attributes depend on the pest type. To protect the field, the farmer can take the advantage of multiple means (e.g. pesticides) called *actions*. However, each pest type has different resistance to each farmer's action (hereinafter referred to as *resistance matrix*). Usually, the problem is time-limited to discrete number of turns. In every turn an agent can execute one action only. It makes simple strategies like applying all actions for every pest inefficient.

The key assumption here is that the farmer agent is not aware of the possible types of the pests nor the resistance matrix. What he can see are the pests' attributes. Based on them, he needs to learn how to recognize different pest types. To learn the resistance matrix, the agent needs to experiment with different actions and observe their effects (i.e. whether the pet dies or not). To make the problem more complicated, the effects are not always immediate and they depend on the resistance matrix. The resistance of the specific pest type to a specific action is valued by the time after which the pets dies (pest's immunity to the action is valued as infinite time). Pests can also have a maximum life-span after which they die regardless of the agent's actions. This maximum life span is called *alive time*.

The problem can be further extended by introduction of deviations to the observed values of the pests' attributes or limiting the number of attributes the farmer agent can see. Another interesting aspect is also the communication between farmer agents which allows them for exchanging obtained knowledge, in this way improving their efficiency in fighting the pests. Cooperation to eliminate some pests may be also necessary.

Based on the above description, the following problem dimensions can be identified:

- number of pest attributes and possible values (the value space can be either discreet or continuous),
- relation between pest type and attribute values,
- distribution of attribute values (e.g. random, Gauss),
- number of pest types,
- number of possible farmer actions,
- possible combination concerning relation between types and actions,
- the delay between action and its effect (so in most cases it won't be a Markov process),
- distribution of pest types regarding fields and farmer agents,
- communication schemes between farmer agents,
- deviations rate of attribute values,
- necessity of cooperation.

Beside various configuration dimensions, the problem offers a flexibility in defining the goals and end-conditions as well. For example, the typical goal will be to kill as many pests as possible or to maintain the highest sum of time periods, in which there were no pests on the fields. As for end-condition, it can be defined in terms of time (given time period or number of turns in case of discrete timing), results (reaching specific number of kills) or other properties (e.g. limited number of actions will enforce farmer agent to optimize his choices).

4. Architecture of Learning Agent

In this section we present the learning agent architecture, which is used in the experiments. It is presented in Fig. 1. The agent consists of four modules:

Processing Module is responsible for basic agent activities, storing training data, executing learning process, and using learned knowledge;

Learning Module is responsible for execution of learning algorithm and giving answers for problems with use of learned knowledge;

Training Data is a storage for examples used for learning;

Generated Knowledge is a storage for learned knowledge.

These components interact in the following way. *Processing Module* receives *Percepts* from the environment. It may process them and execute *Actions*. If during processing learned knowledge is needed, it formulates a *Problem* and sends it to the *Learning Module*, which generates an *Answer* for the *Problem* using *Generated Knowledge*. *Processing Module* decides also what data should be stored in the *Training Data* storage. When necessary (e.g. periodically, or when *Training Data* contains many new examples) it calls the *Learning Module* to execute the learning algorithm to generate new knowledge from *Training Data*. The learned knowledge is stored in the *Generated Knowledge* base.

The form of the knowledge stored in the *Generated Knowledge* depends on the utilized learning algorithm. It may have an explicit form, like rules or decision tree in the case of supervised learning. It can be also stored in a

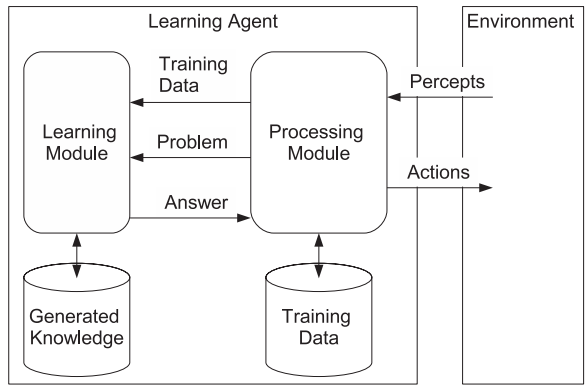

Figure 1: Learning agent architecture

low-level form, like parameters representing action value function or neural network approximator of such a function if reinforcement learning is applied.

Learning module may be defined as a four-tuple: (*Learning Algorithm*, *Training Data*, *Problem*, *Answer*). Characteristics of the training data, the problem and the answer depend on the learning strategy used in the learning module.

Two types of learning modules were tested in experiments: reinforcement learning and supervised learning.

For reinforcement learning we used SARSA algorithm [14] in experiments. The *Problem* definition that is provided consists of the description of the current state (attributes of pests). The *Answer* is an action chosen using the current strategy. *Training data* consists of the next state description (after executing action returned by the module), a reward, and action, which should be executed next. The reward is 0 if pest is dead and -1 if it is not. *Generated Knowledge* is a Q function representation.

Supervised learning module gets a *Training data*, which is a set of examples consisting of attributes of a pest p: $(a_1(p), a_2(p), \ldots, a_n(p))$ labeled by actions executed. Learning agent stores a new example $(a_1(p), a_2(p), \ldots, a_n(p), a)$ in the *Training data* if it observes that execution of action a killed pest p.

As a consequence, *Generated Knowledge*, which is obtained from this data, is a classifier in which class variable corresponds to actions. In experiments we used Naïve Bayes classifier, decision tree and decision rules. *Problem* is a set of pest attributes, and the *Answer* is action, which should be executed.

Experimental results show, that for both types of learning exploration helps to avoid local optimum. Instead of executing action provided by the learned knowledge, random action is executed. Probability of exploration decreases along the time. Boltzmann selection [14] is used in experiments for this purpose.

5. Experimental results

To test the environment and show that various conditions favor different learning algorithms, three experiments with various settings were performed. The values of these settings defined the difficulty of the testing environments. As result, three types of experimental environments were prepared: simple, medium and complex.

Five agents take part in every experiment. Four of them use learning algorithm: SARSA, Naïve Bayes, C4.5 and Ripper. Fifth executes random actions. Supervised learning algorithms were provided by Weka [15], in which C4.5 implementation is called J48, and Ripper implementation is JRip. SARSA algorithm was provided by reinforcement learning library written by Pawel Cichosz [16]. Boltzmann temperature parameter for SARSA is $\tau = 0.1$, for Naïve Bayes $\tau = 0.2$, for J48 and JRip $\tau = 0.3$. τ values are chosen in the initial experiments to achieve the best results for every algorithm. There is no delay in action results. Every experiment consists of 100 simulations. Every simulation consists of 15-20 consecutive games. Knowledge of agents is preserved from game to game. Although, it is cleared between simulations. In the experiments we checked how the performance of agents changes during simulation. Figures present means of numbers of pests eliminated by every agent in consecutive games.

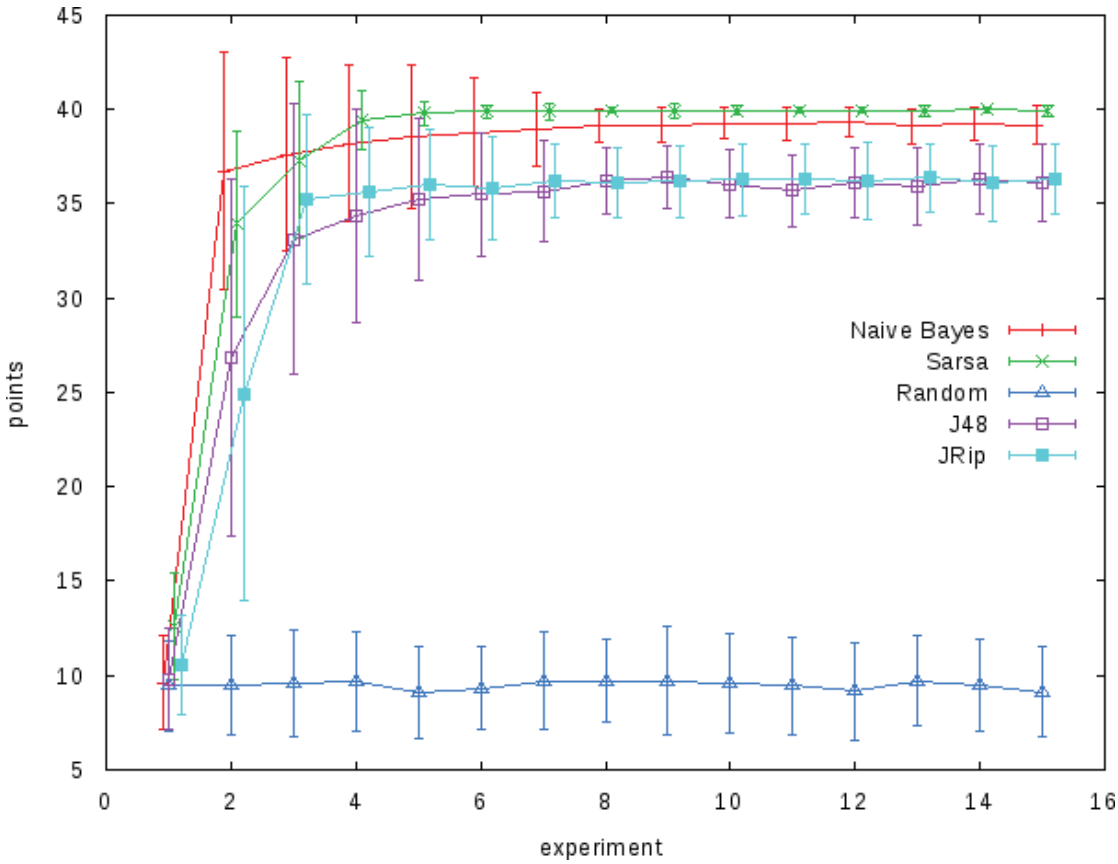

Figure 2: Means of numbers of pests eliminated by every agent in consecutive games in Experiment 1

5.1. Experiment 1 – Simple Environment

In this experiment environment is simple. There are 4 types of pests described by two nominal attributes from domains of size two. Every type has its unique combination of attribute values. It allows to recognize every type using one attribute only. 40 pests appear during the game. Results are presented in Fig. 2.

Learning agents perform much better than the random one. At the end of simulation SARSA and Naïve Bayes are better than C4.5 and Ripper and the difference is statistically significant (using t-test, at $p < 0.05$). Learning is very fast.

5.2. Experiment 2 – Medium Environment

Here situation is more complicated. There are 16 types of pests described by three nominal attributes from domains of size four. Again, every type has its unique combination of attribute values. 80 pests appear during the game. Results are presented in Fig. 3.

As above, learning agents perform much better than the random one. The supervised learning algorithms perform better at the beginning. But at the end, SARSA catches up Naïve Bayes and, finally, they are both significantly better (at $p < 0.05$) than C4.5 and Ripper. Learning is slower than in the simple case.

5.3. Experiment 3 – Complex Environment

Here situation is difficult. There are 8 types of pests described by four attributes from domains of size two or three. The attributes are discrete random variables. The distribution of values is presented in Tab. 1. As we can see, no single attribute can be used to recognize pest type – tests on several attributes are necessary. Therefore, this domain

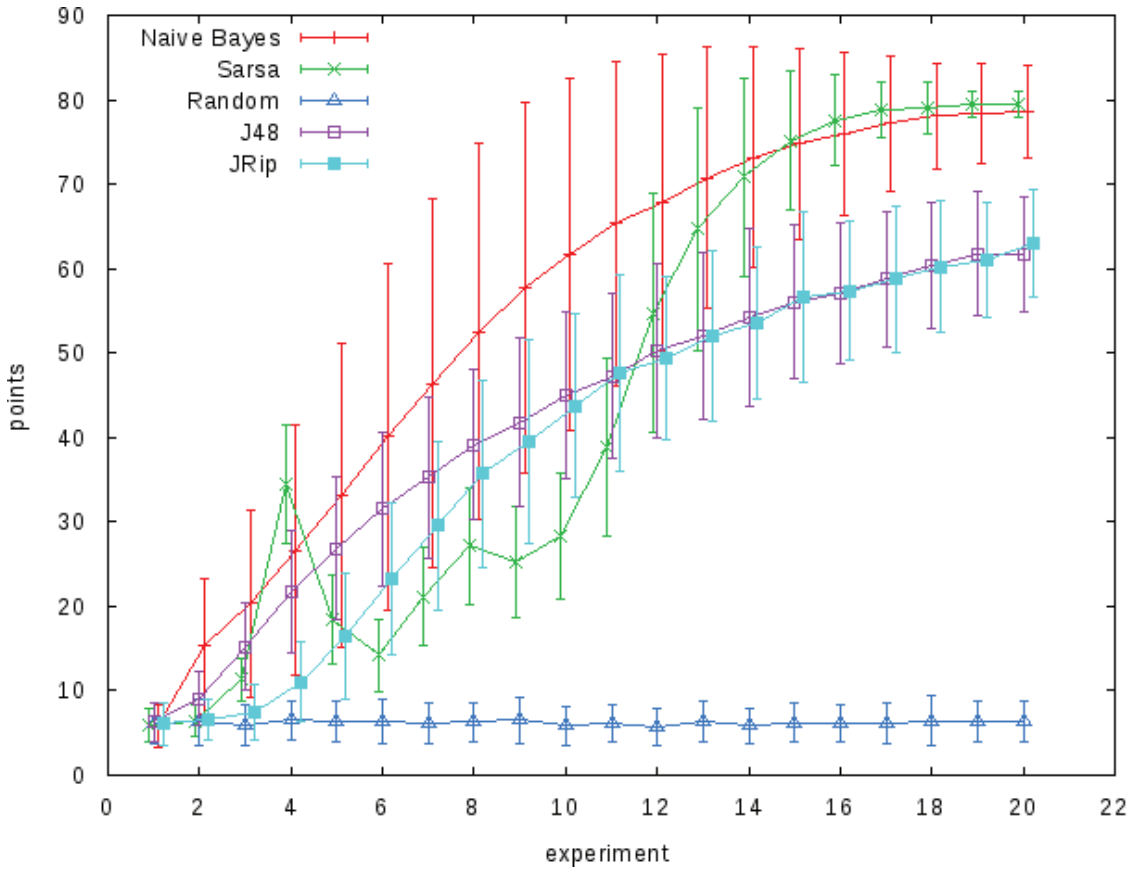

Figure 3: Means of numbers of pests eliminated by every agent in consecutive games in Experiment 2

Table 1: Pest's attribute probability distributions for experiment 3

Pest	size		legsno		speed			jump		
type	40	10	40	30	40	20	10	40	20	10
1	0.6	0.4	0.9	0.1	0.8	0.1	0.1	0.8	0.2	0.0
2	0.6	0.4	0.8	0.2	0.7	0.2	0.1	0.05	0.9	0.05
3	0.5	0.5	1.0	0.0	0.8	0.1	0.1	0.05	0.15	0.8
4	0.4	0.6	0.9	0.1	0.0	0.9	0.1	0.8	0.1	0.1
5	0.5	0.5	0.1	0.9	0.05	0.85	0.1	0.05	0.8	0.15
6	0.5	0.5	0.2	0.8	0.0	1.0	0.0	0.05	0.15	0.8
7	0.6	0.4	0.0	1.0	0.0	0.1	0.9	0.8	0.1	0.1
8	0.5	0.5	0.1	0.9	0.1	0.1	0.8	0.05	0.8	0.15

is more complex than the previous one, despite it has less pest types. 80 pests appear during the game. Results are presented in Fig. 4.

Here C4.5 and Ripper perform much better than SARSA and Naïve Bayes and the difference is significant (at $p < 0.05$). Learning is much slower than in the previous cases. C4.5 and Ripper are able to distinguish types using tests on several attributes, while both SARSA and Naïve Bayes are not able to achieve this.

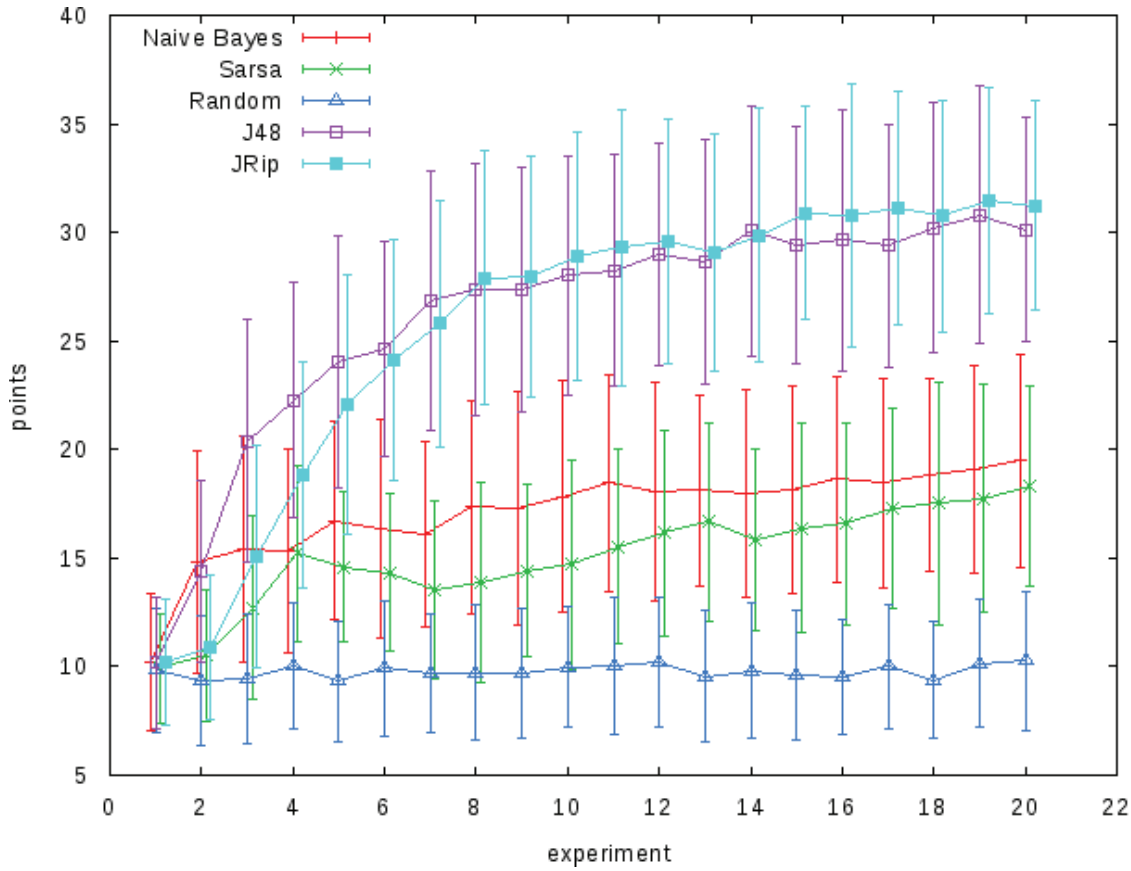

Figure 4: Means of numbers of pests eliminated by every agent in consecutive games in Experiment 3

6. Conclusion and Further Research

In this paper we proposed a scalable multi-dimensional problem domain for testing agent learning algorithms. The large number of configurable dimensions enables preparation of different testing conditions. This allows to test algorithms more thoroughly by examining their performance for various problem parameters. Initial experimental results show that the changes in distribution of pest attributes' values can completely reorganize the algorithms ranking. They also show that supervised learning algorithms can be used to generate agent strategy.

For simple environments, every learning algorithm give fast improvements, but finally, SARSA and Naïve Bayes algorithms give the best results. If environments is difficult, with probabilistic distributions of attribute values, when it happens that only one attribute differentiates the types, supervised learning algorithms (C4.5 and Ripper) perform much better than SARSA and Naïve Bayes.

In cases when the knowledge learned by agents should be interpreted by humans, supervised learning algorithms should be considered. Knowledge stored during the learning process can be interpreted by human. Even in the case of Naïve Bayes, it can be transformed into rules [17].

Future work will be concentrated on execution of number of experiments with more algorithms. The main goal of these experiments will be determining specific conditions for every algorithm in which it performs well. This will improve our understanding of these algorithms and allow for their classification with respect to problem types and parameters.

Another aspect of work will be the extension of testing environment to cover cooperation between agents and delays in action results: more than one agent will have to act to eliminate pests. Additionally, a work will be carried out to include disturbances in the agent measurements of the pests' attribute values.

Acknowledgments

This research was funded in part by the Polish Ministry of Science and Higher Education grant number N N516 366236.

References

[1] L. Panait, S. Luke, Cooperative multi-agent learning: The state of the art, Autonomous Agents and Multi-Agent Systems 11 (2005) 2005.
[2] S. Sen, G. Weiss, Learning in multiagent systems, MIT Press, Cambridge, MA, USA, 1999, pp. 259–298.
[3] S. Luke, C. Hohn, J. Farris, G. Jackson, J. Hendler, Co-evolving soccer softbot team coordination with genetic programming, in: H. Kitano (Ed.), RoboCup-97: Robot Soccer World Cup I, Springer-Verlag, 1997, pp. 398–411.
[4] P. Riley, M. Veloso, On behavior classification in adversarial environments, in: Distributed Autonomous Robotic Systems 4, Springer-Verlag, 2000, pp. 371–380.
[5] P. Stone, R. S. Sutton, G. Kuhlmann, Reinforcement learning for robocup-soccer keepaway, Adaptive Behavior 13 (2005) 2005.
[6] M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative agents, in: In Proceedings of the Tenth International Conference on Machine Learning, Morgan Kaufmann, 1993, pp. 330–337.
[7] T. Haynes, I. Sen, Evolving behavioral strategies in predators and prey, in: Adaptation and Learning in Multiagent Systems, Springer Verlag, 1996, pp. 113–126.
[8] C. L. Giles, K.-C. Jim, Learning communication for multi-agent systems, in: WRAC, 2002, pp. 377–392.
[9] B. Śnieżyński, Agent strategy generation by rule induction in predator-prey problem, in: ICCS 2009: Proceedings of the 9th International Conference on Computational Science, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 895–903.
[10] B. Śnieżyński, Two methods of agent strategy generation by rule induction, in: R. Tadeusiewicz, A. Ligeza, W. Mitkowski, M. Szymkat (Eds.), CMS'09 – 7th Conference Computer MEthods and Systems, Springer, 2009, pp. 141–146.
[11] A. S. Wu, A. C. Schultz, A. Agah, Evolving control for distributed micro air vehicles, in: In IEEE Conference on Computational Intelligence in Robotics and Automation, 1999, pp. 174–179.
[12] L. E. Parker, C. Touzet, Multi-robot learning in a cooperative observation task, in: In Distributed Autonomous Robotic Systems 4, Springer, 2000, pp. 391–401.
[13] F. Fernndez, D. Borrajo, L. E. Parker, A reinforcement learning algorithm in cooperative multirobot domains, Journal of Intelligent Robotics Systems.
[14] R. Sutton, A. Barto, Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning), The MIT Press, 1998.
[15] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 1999.
[16] P. Cichosz, Reinforcement learning in java.
URL `http://www.ise.pw.edu.pl/~cichosz/rl-java/`
[17] B. Śnieżyński, Converting a naive bayes models with multi-valued domains into sets of rules, in: S. Bressan, J. Küng, R. Wagner (Eds.), Database and Expert Systems Applications, Vol. 4080 of LNCS, Springer, 2006, pp. 634–643.