Combining Rule Induction and Reinforcement Learning

An Agent-based Vehicle Routing

Bartłomiej Śnieżyński
Department of Computer Science
AGH University of Science and Technology
Krakow, Poland
sniezyn@agh.edu.pl

Wojciech Wójcik
Department of Computer Science
AGH University of Science and Technology
Krakow, Poland
wwojcik@student.agh.edu.pl

Jan D. Gehrke
Center for Computing and Communication Technologies – TZI
Universität Bremen
Bremen, Germany
jgehrke@tzi.de

Janusz Wojtusiak
Machine Learning and Inference Laboratory,
George Mason University
Fairfax, VA, USA
jwojtusi@gmu.edu

***Abstract*—Reinforcement learning suffers from inefficiency when the number of potential solutions to be searched is large. This paper describes a method of improving reinforcement learning by applying rule induction in multi-agent systems. Knowledge captured by learned rules is used to reduce search space in reinforcement learning, allowing it to shorten learning time. The method is particularly suitable for agents operating in dynamically changing environments, in which fast response to changes is required. The method has been tested in transportation logistics domain in which agents represent vehicles being routed in a simple road network. Experimental results indicate that in this domain the method performs better than traditional Q-learning, as indicated by statistical comparison.**

Keywords-intelligent agents, simulation, rule learning, reinforcement learning

I. INTRODUCTION

An attractive approach to handling complex and computationally hard situations is through their decentralization. Decentralized systems tend to be more robust and adaptive in changing environments. One important class of decentralized problem solving is based on multi-agent systems. In complex or dynamically changing environments it is difficult, or sometimes even impossible, to design all system details a priori. To overcome this problem one can apply machine learning methods, which allow adapting the system to the environment. Application of learning in a multi-agent system requires choosing a learning method that fits to the problem. Among many machine learning algorithms used in multi-agent systems, the most popular are those based on reinforcement learning or evolutionary computation.

In both, reinforcement learning and evolutionary learning, feedback from the environment about quality of the last action executed is sufficient to learn the strategy. In contrast, rule induction is a supervised learning method that requires training data in a form of labeled examples. In many cases learning agents are able to generate training data using their experience. Symbolic knowledge representation has several advantages, particularly unordered sets of attributional rules which correspond to the human way of expressing knowledge [10]. Therefore, knowledge in such forms generated by intelligent agents can be understood and verified by human experts. This feature, often ignored in autonomous systems, is particularly important in critical applications in which humans need to audit agents' behavior. Rule-based representations also provide modularity that allows for easier exchange of knowledge between agents and human experts.

The presented research is to combine two learning methods: reinforcement learning and rule induction. In the next section the problem definition is presented, followed by sections with simulation model description, details of learning agents, combining of reinforcement learning and rule induction, experiments and related research.

II. PROBLEM DESCRIPTION

In dynamically changing environments intelligent agents require the ability to adapt in order to efficiently perform their tasks. The example application area considered in this paper is adaptation and learning in agents representing vehicles. The agents need to make decisions about which routes to select to arrive at the destination in the shortest time. Agents predict or are provided with information about traffic, time, and road network, based on which their routes can be planned. Agents travel within a simple road network represented by a grid (see Section III B), and need to make decisions on which roads to select.

This application area is well known in the literature and many solutions have been proposed. It has been selected not to compete with existing routing or planning algorithms, but rather to study the effect of combining reinforcement and rule learning.

The specific problem considered here is how quickly different learning strategies (Q-learning, rule learning + Q-learning) converge to an optimal or near optimal routing strategy based on agents' experience and how that strategy compares to optimal choices. Faster convergence to a near-optimal solution is important when the environment dynamically changes and there is a need to update existing strategies.

III. Simulation Model

A. PlaSMA System

In order to evaluate the impact of environmental knowledge and driving time prediction abilities, experiments were set up within the simulation system PlaSMA (*Platform for Simulation with Multiple Agents*). PlaSMA [13] is a multiagent-based simulation (MABS) system, i.e., distributed simulation with discrete events, using software agents as logical simulation processes. MABS provides a natural way to evaluate software agents' behavior and interaction.

PlaSMA was developed to support experimenting with multi-agent systems in logistics scenarios. The simulation system provides ontology knowledge on logistic entities and transport infrastructure. The virtual environment is based on a directed graph spatially mapped to a geographic coordinate system. An extendable library of physical actions (e.g., driving or loading cargo) and events (e.g., occurring cargo) is available to ease scenario modeling. The PlaSMA system handles and synchronizes agent actions and simulation events as well as agent message passing in order to guarantee causality and reproducibility of simulations [13].

Additionally, the system offers means to record scenario evaluation metrics into a database and to archive simulation results. Simulations can be loaded, controlled, and tracked in a GUI client that enables OpenGL 3D Earth visualization.

B. Environment

For the logistics scenario a road network was set up as a 3×3 grid graph with a set E of directed edges (i.e., unidirectional roads) of various lengths (Fig. 1). Although this grid structure is very simple it allows the agent to adjust the path to the traffic density. In fact, a very small grid on which it is possible to illustrate how the described method of combining rule induction with reinforcement learning works was deliberately selected. Experiments with much larger road networks can be conducted within the PlaSMA system. The goal of the agents is to travel from bottom left corner of the grid to upper right one and return. For the return travel, the graph is symmetric according to the middle node.

Function $dens_g(T)$ provides a linear traffic density measure in the whole environment for given time T. It is used to simulate changes in traffic during various hours or days of week. It is normalized to range 0 to 1 and indicates the ability of a vehicle to drive at a reference speed v_{ref}. Every edge e has a traffic modifier assigned $tm(e)$. As a result, function *dens* can be extended to represent traffic density in a given time for a given edge:

$$dens(e, T) = dens_g(T) + tm(e) \quad (1)$$

Maximum possible speed on every edge e is given by (2).

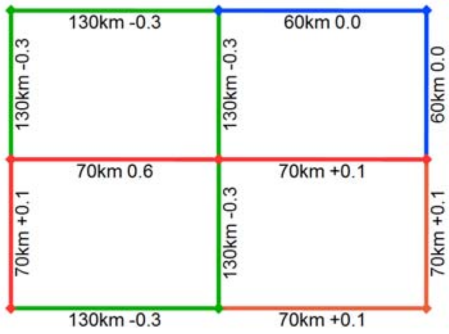

Fig. 1. Road network; edges are marked with the road length and traffic density modifier

$$v_{max}(e, T) = v_{ref}(e) \cdot (1 - dens(e,T)) \quad (2)$$

For our experiments $v_{ref}(e)$=100 km/h for all roads was used. Hence, if traffic density is 0.5 a vehicle is not able to drive faster than 50 km/h on average.

IV. Learning Agents

In the presented research four types of learning agents are considered. They use Q-learning to create an optimal or near optimal strategy based on input data consisting of different types of measurements. These data always contain information about current agent location $(x, y) \in \{0, 1, 2\}^2$, and optionally also traffic density and date/time.

Location learning agent (*LA*) uses only location information for learning (Fig. 2-A). The agent uses only the Q-learning algorithm to find an optimal route depending on the current location.

Location and traffic learning agent (*LTA*) gets information about locations and also about current traffic density $tr = dens(e, t)$ (Fig. 2-B). Traffic density is discretized into six ranges, from very low to very high. The agent uses only Q-learning to find an optimal route based on current location and complete information about traffic density.

Input data may also contain information about current day of the week ($d \in \{1, 2, \ldots, 7\}$) and current time expressed as an hour ($t \in \{0, 1, 2, \ldots, 23\}$). *Location, day and time learning agent* (*LDTA*) is using these data for learning applying the Q-learning algorithm (Fig. 2-C), but does not include explicit information about traffic density.

AQ-RL agent (Fig. 2-D) combines two methods of learning. The AQ rule induction algorithm is used to generate a classifier, which predicts traffic (its discretized value) from date and time data. Prediction given by the classifier, combined with location data, is used as an input to the reinforcement learning module. Next section briefly describes Q-learning and rule learning algorithms, and a method for combining these two learning algorithms.

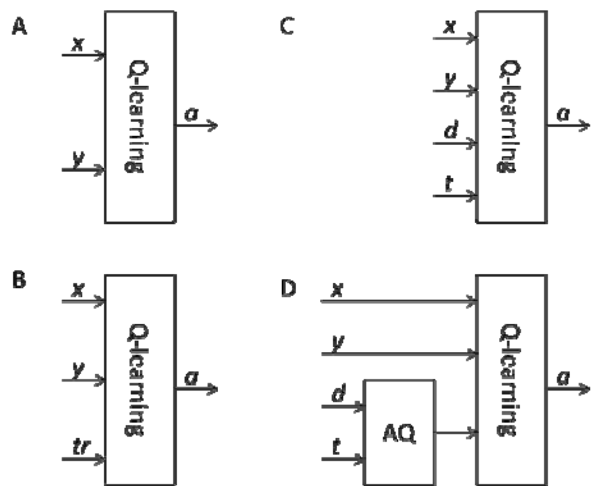

Fig. 2. Learning schemas for four types of agents.

V. Combining Reinforcement and Rule Learning

The AQ-RL agents use rule learning in order to reduce search space in finding the optimal routing strategy. Due to the reduced search space in Q-learning, the agents are able to faster converge to a near-optimal strategy, which affects overall agents' performance. The following sections briefly describe Q-learning, AQ rule induction, and the proposed hybrid method that combines the two.

A. Q-Learning

The most popular learning method in multi-agent systems is reinforcement learning in which agents improve their performance on a specific task by trying different ways the task can be performed. In this method, an agent gets description of the current state and using a selected strategy it chooses an appropriate action from a defined set. Next, using reward from the environment and the next state description, the agent updates its strategy. Several methods of choosing the action and updating the strategy have been developed so far. In Q-learning [21], used in the presented study, action with the highest predicted value (Q) is chosen. The function Q, given by (3) estimates value of each action in a given state,

$$Q: A \times S \rightarrow R, \tag{3}$$

where A is a set of actions, and S is a set of possible states. Q function is updated after the selected action is executed

$$\begin{gathered} Q(a, s) := Q(a, s) + \beta\Delta \\ \Delta = \gamma\, Q_{max} + r - Q(a, s) \\ Q_{max} = max_a\, Q(a, s') \end{gathered} \tag{4}$$

where $s, s' \in S$ are subsequent states, $a \in A$ is an action chosen, r is a reward obtained from the environment, $\gamma \in [0,1]$ is a discount rate (importance of the future rewards), and $\beta \in (0,1)$ is a learning rate. Various techniques are used to prevent from getting into a local optimum. The idea is to better explore the solution space by occasionally choosing not optimal actions (i.e. random action or one not performed yet in a given state). Boltzmann exploration was used in experiments of the presented study.

To speed up the learning process, it is possible to update Q-values for states visited in the recent past. As a result, Q(λ)-learning algorithm has been proposed [17], in which $\lambda \in [0,1]$ is a parameter used to determine how many old states are considered (value 1 means all states in the past). We used λ=0.4 in the experiments.

B. AQ Rule Induction

AQ is a class of machine learning programs that induce attributional rules from data and prior knowledge. A basic form of attributional rules is (5).

$$CONSEQUENT <== PREMISE \tag{5}$$

Here, CONSEQUENT and PREMISE are conjunctions of attributional conditions. Basic attributional conditions are in the form (6):

$$[L\ rel\ R] \tag{6}$$

in which L is an attribute; R is a value, a disjunction of values, or a conjunction of values if L is a compound attribute; and *rel* is a relation that applies to L and R. Other forms of attributional conditions may involve count attributes, simple arithmetical expressions, conjunctions and disjunctions of attributes, comparison of attributes, and others [10] .

In the presented study AQ21, the newest and most advanced program from the AQ family [22] was used. Given input data, problem definition, and optional background knowledge, AQ21 induces attributional rules in the form (5) or in more advanced forms, describing one or more classes in the data. A set of rules constituting a description of a given class is called a *ruleset*. By repeating learning for all classes defined by values of an output attribute, AQ21 generates a classifier.

In order to learn rules for a given class AQ21 starts with one example, called a *seed*, belonging to the class. It generates a *star*, which is a set of maximally general rules that cover the seed and do not cover any examples from other classes. This is done by repeating an *extension-against* operation that generalizes the seed against examples not belonging to the concept being learned. Results of applying the extension-against are intersected and the best rules are selected according to user-defined criteria. If selected rules do not cover all examples belonging to the class, another seed is selected (from the not covered examples) and additional rules are learned. The process is repeated until all examples of the class are covered by the learned rules. AQ21 implements several modifications to the above basic algorithm [22] that allow it to handle noise and perform additional operations.

C. *Hybrid Learning*

Q-learning suffers from inefficiency when the space of possible states is large. In such a case, an intelligent agent needs to examine a large number of states before it learns the optimal set of actions. One possible solution to this problem, investigated in this paper, is to reduce the set of possible states, by invoking a secondary learning process. The secondary learning aims at learning and predicting values that can be used as inputs to Q-learning. Specifically, in the equation (3) let

$$S = S_1 \times S_2 \times \ldots \times S_n \tag{7}$$

be a Cartesian product of domains of n attributes. S is the original search space for Q-learning. The goal of the secondary learning process is to create a model

$$M(S) = U \tag{8}$$

$$U = U_1 \times U_2 \times \ldots \times U_k \tag{9}$$

such that $k < n$, thus reducing the number of possible states in Q-learning. Note that, in addition to reduction in the number of attributes, also reduction of number of values for these attributes affects the number of states. Because of the use of generalization provided by the secondary learning, Q-learning algorithm does not need to search through all possible states in the original space S, but rather through U, a smaller space already processed by the secondary learning.

Supervised learning algorithms can be used for the reduction. Several attributes from S can be replaced by a class attribute (see Fig. 2-D). To apply supervised learning, the agent should have a possibility to create examples for learning.

In the simple case of agent *AQ-RL* described in Section IV, the space S is defined as

$$S = x \times y \times d \times t \tag{10}$$

and

$$U = x \times y \times tr \tag{11}$$

This reduces the space by replacing two variables with one variable tr, with small domain size. Here, the secondary learning is performed by the AQ21 rule induction system. Examples to generate knowledge represent traffic observed by an agent in a given day and time, and have a form $ex=(d_{ex}, t_{ex}, tr_{ex})$. An example is added to the agent's memory after passing an edge. Parameters d_{ex} and t_{ex} are calculated to represent day and time of middle of the travel time. The parameter tr_{ex} is the discretized agent's estimation of traffic density, $dens(T)$, which is calculated using time spent for the travel. In the presented simplified model only traffic density influences speed on a given edge.

VI. Experimental Results

In order to experimentally test the effect of combining rule learning and reinforcement learning, a large number of simulations in the simple environment described in Section III B were executed. The scenario was simple enough to illustrate the effect of learning. Experiments were repeated 100 times. That was sufficient to show that results in mean values in travel times are significantly different, as checked using the two-tailed t-test (null hypothesis is C=D) with the obtained p-values indicating statistical significance.

Table 1 includes means and standard deviations of driving times for the four types of agents during different days of the simulation and p-values for comparing Q-learning with date and time, and hybrid learning. The results indicate that agents employing Q-learning combined with AQ rule learning converge much faster to near optimal performance (below 4 hours driving time) than agents using traditional Q-learning. Before day 25 of the simulation the AQ-RL agent achieves that performance, while the performance of the compared agent at this time is worse. This result indicates that the AQ21 system was able to sufficiently approximate traffic regularities to match the performance of agent B that is provided with traffic information. The table includes results from all agents described in Section IV.

VII. Related Research

Although both reinforcement learning and rule induction are applied in multi-agent systems, combining the two is an original idea that has not been tried before. Paragraphs below briefly describe related research on multiagent simulations, rule induction, and learning in multiagent systems.

Table 1: Mean +/- standard deviation of travel times in experimental comparison of four types of agents: naïve, Q-learning with full information, Q-learning with location and date information, and AQ/reinforcement learning hybrid.

Agent	Day 1	Day 10	Day 25	Day 51	Day 75	Day 100	Day 200
A: Q with location only	16.19+/-3.98	11.92+/-6.27	4.39+/-2.18	4.24+/-0.98	3.98+/-0.69	4.24+/-0.92	4.14+/-0.57
B: Q with location and traffic	15.48+/-4.81	17.80+/-9.34	3.56+/-1.84	3.46+/-0.59	3.34+/-0.34	3.46+/-0.64	3.39+/-0.59
C: Q with location and time	15.80+/-4.54	27.50+/-21.28	15.11+/-13.80	12.51+/-15.75	5.80+/-3.44	5.14+/-6.31	3.66+/-1.43
D: AQ-RL	19.59+/-10.06	20.44+/-10.86	3.88+/-2.04	3.88+/-2.84	3.43+/-0.49	3.63+/-1.91	3.41+/-0.80
p-value (C vs D)	0.0036	0.0106	8.235E-20	4.339E-16	2.021E-55	0.0003	1.099E-22

Multiagent-based simulation (MABS) is a popular means for evaluating agent-based software systems as well as social systems in general. As a microsimulation, MABS offers an accurate mapping from autonomous and interacting sub-systems to simulation entities. In addition to PlaSMA, systems such as MASON, Cougaar, JAMES, and AnyLogic are applied for simulation of agent-based software systems, including the logistics domain [9]. Agent-based simulators like NetLogo, Repast, Swarm, and SeSAm are mainly used in the social sciences. PlaSMA is distinguished by a focus on logistics, FIPA-compliance, formal model semantics, and high modeling flexibility. The latter feature, however, demands Java programming skills while some other platforms (e.g. SeSAm) provide visual programming [9].

Rule learning is one of the most popular and well studied approaches to supervised learning, although most programs are limited to inducing rules from data, without using any background knowledge. Programs from AQ family, whose AQ21 was used in this study, pioneered separate-and-conquer approach to rule learning. Over four decades of development several implementations with different features have been created. The most notable are AQVAL/AQ7, AQ11, AQ17, and current AQ21.

A review of separate-and-conquer approach to rule learning is in [4]. Among the best well known rule learning programs are CN2, RIPPER and its successor SLIPPER, and several programs based on rough set theory [7]. Currently investigated are statistical approaches to rule learning, i.e. one that uses maximum likelihood estimation to guide search for the best rules [2].

In multi-agent systems two main techniques applied for learning are reinforcement learning, and evolutionary computation. However; other techniques, such as supervised learning are also applied. Good survey of learning in multi-agent systems working in various domains can be found in [14] and [17]. Learning can be applied in various environments. Predator-Prey is one of them, where several learning techniques were applied. [20] is an example of reinforcement learning application. In this work predator agents use reinforcement learning to learn a strategy minimizing time to catch a prey. Agents can cooperate by exchanging sensor data, strategies, or episodes. Experimental results show that cooperation is beneficial. Two other works successfully apply genetic programming [8] and evolutionary computation [6] in this domain. Predator strategy can be also learned using rule induction [18].

Another domain, where several learning techniques were applied is target observation. In [23] rules are evolved to control large area surveillance from the air. In [15] Parker presents 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 a random, but worse than a manually developed one. Results of application of reinforcement learning mixed with state space generalization method can be found in [3], where Evolutionary Nearest Neighbor Classifier – Q-learning (ENNC-QL) is proposed. It is a hybrid model, a combination of supervised function approximation and state space discretization with Q-learning. This method has similar goals to hybrid algorithm presented in this paper: reduction of state space for reinforcement learning, with minimal possible information loss, so that the Markov property can be still be satisfied after applying the reduction. For ENNC-QL this works best in deterministic domains. Technically, ENNC-QL algorithm works as a very sophisticated function approximator with built-in discretization support. The main application domain of ENNC-QL approach consists of problems with possibly large, continuous state spaces. [3] gives no information about experiments with pure discrete state spaces, so the range of applications is basically somewhat different than for the hybrid model proposed here. Additionally, the ENNC-QL algorithm requires several predefined phases in order to compute discretization and state space representation, including explicit exploration phase and two learning phases, so it might be hard to apply in non-stationary, changing environments. On the other hand, it is more generic than hybrid model described here, because it can be easily applied to any continuous state space problem without making any assumptions on the problem's domain.

There are also several other works about learning in multi-agent systems that are using supervised learning. Rule induction is used in a multi-agent solution for vehicle routing problem [5]. However; in this work learning is done off-line. In [19], agents learn coordination rules, which are used in coordination planning. If there is not enough information during learning, agents can communicate additional data during learning. Airiau [1] adds learning capabilities into BDI model. Decision tree learning is used to support plan applicability testing. Nowaczyk and Malec are also using learning for plans evaluation. Inductive Logic Programming is used to generate knowledge for choosing the best partial plan for an agent [12].

VIII. Conclusions

This paper described a method of combining reinforcement learning and rule induction. The method's goal is to improve learning efficiency by decreasing search space in reinforcement learning. Learned rules are used as a classifier that is able to replace several dimensions (attributes) with one (class attribute). Rule-based models can be also easily verified by human experts that can be very important in many application domains.

In order to apply the method, an agent needs to be able to generate examples for rule induction. These examples connect selected input attributes with a class attribute computed by the agent.

The presented experimental work was designed to show working of the method on a simple scenario on which its advantage can be already seen. Experimental evaluation in more complex scenarios and in large-scale simulation systems is needed to test how the method applies in real world complex situations.

Other future work on the method will include investigation of the relation between search parameters in reinforcement learning versus generalization in rule learning. Also, the possibility of communication and cooperation between agents should be taken into account to investigate effects of exchange of agents' knowledge. Because of rule-based representation of knowledge, the process may be very efficient.

ACKNOWLEDGMENTS

This research funded in part by the German Research Foundation (DFG) within Collaborative Research Centre 637 (SFB 637), and in part by the Polish Ministry of Science and Higher Education grant number N N516 366236. Development of the AQ21 system was partially funded by the U.S. National Science Foundation grants.

REFERENCES

[1] S. Airiau, L. Padham, S. Sardina, and S. Sen, "Incorporating learning in BDI agents," Proc. of the ALAMAS+ALAg Workshop, 2008.

[2] K. Dembczyński, W. Kotłowski, and R. Słowiński, "Maximum likelihood rule ensembles," Proc. of the 25th Int'l. Conf. on Machine Learning (ICML 08), ACM Press, 2008, pp. 224–231.

[3] F. Fernández, D. Borrajo, and L. E. Parker, "A reinforcement learning algorithm in cooperative multirobot domains," Journal of Intelligent Robotics Systems, vol. 43, pp. 161–174, Aug. 2005.

[4] J. Fürnkranz, "Separate-and-conquer rule learning," Artificial Intelligence Review, vol. 13, pp. 3–54, Jan. 1999.

[5] J. D. Gehrke and J. Wojtusiak, "Traffic prediction for agent route planning," Proc. of the Int'l. Conf. on Computational Science (ICCS 2008), vol. 3, Springer-Verlag, 2008, pp. 692–701.

[6] C. L. Giles and K.-C. Jim, "Learning communication for multi-agent systems," Proc. of 1st Workshop on Radical Agent Concepts (WRAC 2002), Springer-Verlag, 2003, pp. 377–392.

[7] J. W. Grzymala-Busse, "A New Version of the Rule Induction System LERS," Fund. Informaticae, vol. 31, pp. 27–39, 1997.

[8] T. Haynes and I. Sen, "Evolving behavioral strategies in predators and prey," Proc. of IJCAI'95 Workshop on Adaptation and Learning in Multiagent Systems, Springer-Verlag, 1996, pp. 113–126.

[9] R. Herrler and F. Klügl, "Simulation," in Multiagent Engineering, Theory and Applications in Enterprises, S. Kirn, O. Herzog, P. Lockemann, and O. Spaniol, Eds. Berlin: Springer-Verlag, 2006, pp. 575–596.

[10] R. S. Michalski, "Attributional Calculus: A Logic and Representation Language for Natural Induction," Reports of the Machine Learning and Inference Laboratory, MLI 04-2, George Mason University, 2004.

[11] A. Newell and H. Simon, Human Problem Solving. Prentice-Hall, 1972.

[12] S. Nowaczyk and J. Malec, "Learning to evaluate conditional partial plans," Proc. of the Sixth Int'l. Conf. on Machine Learning and Applications (ICMLA '07), IEEE Computer Society, 2007, pp. 235–240.

[13] A. Schuldt, J. D. Gehrke, and S. Werner, "Designing a simulation middleware for FIPA multiagent systems," Proc. of the Int'l. Conf. on Intelligent Agent Technology (IAT 08), IEEE, 2008, pp. 109–113.

[14] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," Autonomous Agents and Multi-Agent Systems, vol. 11, pp. 387–434, Nov. 2005.

[15] L. E. Parker and C. Touzet, "Multi-robot learning in a cooperative observation task," in Distributed Autonomous Robotic Systems 4, L. E. Parker, G. Bekey, and J. Barhen, Eds. Berlin: Springer-Verlag, 2000, pp. 391–401.

[16] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," Proc. of Int'l. Conf. on Machine Learning (ICML 94), vol. 11, Morgan Kaufmann, 1994, pp. 226–232.

[17] S. Sen and G. Weiss, "Learning in multiagent systems," in Multiagent Systems, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999, pp. 259–298.

[18] B. Śnieżyński, "Agent strategy generation by rule induction in predator-prey problem," Proc. of the 9th Int'l. Conf. on Computational Science (ICCS 2009), Springer-Verlag, 2009, pp. 895–903.

[19] T. Sugawara and V. Lesser, "On-line learning of coordination plans," Proc. of the 12th Int'l. Workshop on Distributed Artificial Intelligence, 1993.

[20] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," Proc. of 10th Int'l. Conference on Machine Learning (ICML 93), Morgan Kaufmann, 1993, pp. 330–337.

[21] C. J. C. H. Watkins, Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, 1989.

[22] J. Wojtusiak, R. S. Michalski, K. Kaufman, and J. Pietrzykowski., "The AQ21 natural induction program for pattern discovery: Initial version and its novel features," Proc. of the 18th IEEE International Conference on Tools with Artificial Intelligence, 2006, pp. 523–526.

[23] A. S. Wu, A. C. Schultz, and A. Agah, "Evolving control for distributed micro air vehicles," Proc. of IEEE Int'l. Symp. on Computational Intelligence in Robotics and Automation (CIRA 99), IEEE, 1999, pp. 174–179.