Supervised Rule Learning and Reinforcement Learning in A Multi-Agent System for the Fish Banks Game

Bartłomiej Śnieżyński

AGH University of Science and Technology
Poland

1. Introduction

Environment of multi-agent systems is often very complex. Therefore it is sometimes difficult, or even impossible, to specify and implement all system details a priori. Application of machine learning algorithms allows to overcome this problem. One can implement an agent that is not perfect, but improves its performance.

There are many learning methods that can be used to generate knowledge or strategy in a multi-agent system. Choosing an appropriate one, which fits a given problem, can be a difficult task. The aim of the research presented here was to test applicability of reinforcement learning and supervised rule learning strategies in the same problem.

Reinforcement learning is the most common technique in multi-agent systems. It allows to generate a strategy for an agent in a situation, when the environment provides some feedback after the agent has acted. Symbolic, supervised learning is not so widely used in multi-agent systems. There are many methods belonging to this class that generate knowledge from data. Here a rule induction algorithm is used. It generates a rule-based classifier, which assigns a class to a given example. As an input it needs examples, where the class is assigned by some teacher. We show how observation of other agents’ actions can be used instead of the teacher.

As an environment the Fish Banks game is used. It is a simulation, in which agents run fishing companies and its main task is to decide how many ships send for fishing, and where to send them. Four types of agents are created. Reinforcement learning agent and supervised learning agent improve their allocation performance using appropriate learning strategy. As a reference two additional types of agents are introduced: random agent, which chooses allocation action randomly, and predicting agent, which assumes that fishing results will be the same as in previous round, and allocates ships using this simple prediction.

In the next section related research on learning in multi-agent systems is briefly presented. The third section explains details of the environment, architecture and behaviours of the agents. Next, results of several experiments, which were performed to compare mentioned learning methods, are presented and discussed. Results show that both of them give good results. However; both of them have some advantages and disadvantages. In the last two sections conclusions and further research are presented.
This work is an extended version of the paper (Śnieżyński, 2007), in which initial results are published.

2. Learning in multi-agent systems

The problem of learning in multi-agent systems may be considered as a union of research on multi-agent systems and on machine learning. Machine learning focuses mostly on research on isolated process performed by one intelligent module. The multi-agent approach concerns the systems composed of autonomous elements, called agents, whose actions lead to the realization of given goals. In this context, learning is based on the observation of the influences of activities, performed to achieve the goal by an agent itself or by other agents. Learning may proceed in a traditional – centralized (one learning agent) or decentralized manner. In the second case more than one agent is engaged in the learning process (Sen & Weiss, 1999). A good survey on machine learning in the context of multi-agent systems can be found in (Stone & Veloso, 2000).

So far agent-based systems with learning capabilities were applied in many domains: to train agents playing in RoboCup Challenge (Kitano et.al, 1997), adapt user interfaces (Lashkari et.al, 1994), take part in agent-based computational economics simulations (Tesfatsion, 2001), analyze distributed data (Stolfo et.al, 1997), and to discover intrusions (Servin & Kudenko, 2008).

The learning process is strictly associated with reasoning and decision making aspects of agents. The most popular learning technique in multi-agent systems is reinforcement learning. Other techniques can be also applied. Learning process can be based on the symbolic knowledge representation (e.g. rules, decision trees), neural networks, models coming from game theory as well as optimization techniques (like the evolutionary approach, tabu search, etc.).

Reinforcement learning allows to generate a strategy for an agent in a situation, when the environment provides some feedback after the agent has acted. Feedback takes the form of a real number representing reward, which depends on the quality of the action executed by the agent in a given situation. The goal of the learning is to maximize estimated reward.

Supervised learning is not so popular in multi-agent systems. However; there are some works with use of this strategy. Sugawara is using this technique for improving plan coordination. Gehrke is using rule induction for route planning (Gehrke & Wojtusiak, 2008). However; rule induction is done offline. Szita and Lorincz apply global optimization algorithm to select set of rules used by the agent playing Pac-Man game (Szita & Lorincz 2007). Airiau adds learning capabilities into BDI model. Decision tree learning is used to support plan applicability testing (Airiau et. al 2008).

Universal architecture for learning agent can be found in (Russell & Norvig, 1995). It fits mainly reinforcement learning. Sardinha et. al, propose a learning agent design pattern, which can be used during system implementation (Sardinha et. al, 2004). More abstract architecture is presented in (Śnieżyński, 2008).

3. Multi-agent system for fish banks game

3.1 Environment

Fish Banks game is designed for teaching people effective cooperation in using natural resources (Meadows et.al, 1993). It may be also used in multi-agent systems. In this research
the game is a dynamic environment providing all necessary resources, action execution procedures, and time flow, which is represented by game rounds. Each round consists of the following steps:

- ships and money update,
- ship auctions,
- trading session,
- ship orders,
- ship allocation,
- fishing,
- fish number update.

Agents represent players that manage fishing companies. Each company aims at collecting maximum assets expressed by the amount of money deposited at a bank account and the number of ships. The company earns money by fishing at fish banks.

Environment provides two fishing areas: coastal and a deep-sea. Agents can also keep their ships at the port. Cost of fishing at the deep-sea is the highest. Cost of staying at port is the lowest but such ship does not catch fish.

Initially, it is assumed that the number of fish in both banks is close to the bank’s maximal capacity (equal to 4000 for a deep sea, and 2000 for a coastal area). During the game the number of fish in every bank changes according to the following equation:

$$f_{t+1} = f_t + bf_t \left(1 - \frac{f_t}{f_{\text{max}}}ight) - C_t$$  \hspace{1cm} (1)

where $f_t$ is a fish number at a time $t$, $b$ is a birth rate (value 0.05 was used in experiments), $f_{\text{max}}$ is a maximum number of fish in the bank, $C_t$ is a total fish catch: $n$ is a number of ships of all players sent to the bank, and $c_t$ is a fish catch for one ship at the time $t$:

$$c_t = c_{\text{max}} w_t \sqrt{\frac{f_t}{f_{\text{max}}}}$$  \hspace{1cm} (2)

where $c_{\text{max}}$ is a maximal catch (equal to 25 for a deep sea, and 15 for a coastal area), and $w_t$ is a weather factor at a time $t$. Weather factor is a random number between 0.8 and 1.0.

As we can see, at the beginning of game, when $f_t$ is close to $f_{\text{max}}$, fishing at the deep sea is more profitable. Parameters are set in such a way that exploration overcomes birth, and after several rounds the number of fish can decrease to zero. It is a standard case of “the tragedy of commons” (Hardin, 1968). It is more reasonable to keep ships at the harbor then, therefore companies should change their strategies.

In the original game, fishing companies may order new ships to be built as well as they may cross-sell their ships. The ships may be also sold at the auction organized by the game manager. In the current version of the system ship auctions and trading sessions are not supported.

The costs of building a ship, costs of its maintenance and use and the price of sold fish are fixed for the whole game. At the end of the game the value of the ships owned by the companies is estimated (number of ships is multiplied by a constant) and added to the money balance.
3.2 Architecture of the agents

Four types of agents are implemented: reinforcement learning agent, rule learning agent, predicting agent, and random agent. The first one uses learned strategy to allocate ships, the second one uses rules induced from the experience to classify actions and chose the best one, agent of the third type uses previous fishing results to estimate values of different allocation actions, the last one allocates ships randomly.

All types of agents may observe the following aspects of the environment:
- arriving of new ships bought from a shipyard,
- money earned in the last round,
- ship allocations of all agents,
- fishing results \((c)\) for deep sea and inshore area.

All types of agents can execute the following two types of actions: order ships, allocate ships.

Order ships action is currently very simple. It is implemented in all types of agents in the same way. At the beginning of the game every agent has 10 ships. Every round, if it has less then 15 ships, there is 50% chance that it orders two new ships.

Ships allocation is based on the method used in (Koźlak et.al, 1999). The allocation action is represented by a triple \((h, d, c)\), where \(h\) is the number of ships left in a harbour, \(d\) and \(c\) are numbers of ships sent to a deep sea, and a coastal area, respectively. Agents generate a list of allocations for \(h=0\%\), 25\%, 50\%, 75\%, and 100\% of ships that belong to the agent. The rest of ships \((s)\) is partitioned; for every \(h\) the following candidates are generated:

1. All: \((h, 0, s)\), \((h, s, 0)\) – send all remaining ships to a deep sea or coastal area,
2. Check: \((h, 1, s-1)\), \((h, s-1, 1)\) – send one ship to a deep sea or coastal area and the rest to the other,
3. Three random actions: \((h, x, s-x)\), where \(1 \leq x < s\) is a random number – allocate remaining ships in a random way,
4. Equal: \((h, s/2, s/2)\) – send equal number of ships to both areas (one more ship is sent to a deep sea if \(s\) is odd.)

The random agent allocates ships using one of the action candidates chosen by random.

Predicting agent uses the following formula to estimate the value of each action candidate \(a\):

\[
v(a) = \text{income}(a) + \varepsilon \text{ecology}(a),
\]

where \(\text{income}(a)\) represents the prediction of the income under the assumption that in the current round fishing results will be the same as in the previous round, \(\text{ecology}(a)\) represents ecological effects of the action \(a\) (the value is low if fishing is performed in the area with low fish population), and \(\varepsilon\) represents importance of the ecology factor.

3.3 Learning agents details

Both learning agents have the same general architecture, which is based on one proposed in (Śnieżyński, 2008). It is presented in Fig. 1. Processing module is responsible for analyzing percepts, buying ships, preparing training data, executing learning, and calling a learning module to chose appropriate action in the current situation. To specify details of the learning process and using the learned knowledge, we need the following four-tuple: \((\text{Learning algorithm}, \text{Training data}, \text{Problem}, \text{Answer})\). \text{Learning algorithm} represents a way, in which \text{Training data} is transformed into the internal knowledge representation. This knowledge is used to give an \text{Answer} to a given \text{Problem}. Below specifications for both learning agents are presented.
Supervised Rule Learning and Reinforcement Learning in A Multi-Agent System for the Fish Banks Game

Fig. 1. Architecture of learning agents used in the system implemented for Fish-Banks Game

Agent using reinforcement learning strategy gets description of the current state and using its current strategy chooses an appropriate action from a defined set. Next, using reward from the environment and next state description it updates its strategy. Several methods of choosing the action and updating the strategy have been developed so far. In Q-learning developed by Chris Watkins (Watkins, 1989) \( Q \) is a function that estimates value of the action in a given state:

\[
Q : A \times X \rightarrow \mathbb{R}, \tag{4}
\]

where \( A \) is a set of actions, and \( X \) is a set of possible states. \( Q \) function is updated after action execution:

\[
Q(a, x) := Q(a, x) + \beta \Delta. \tag{5}
\]

\( \Delta \) represents change of the \( Q \) function value that should be applied according to the last reward. It is defined in the following way:

\[
\Delta = \gamma Q_{\text{max}} + r - Q(a, x), \tag{6}
\]

\[
Q_{\text{max}} = \max_a Q(a, x'), \tag{7}
\]

where \( x, x' \in X \) 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.

Reinforcement learning agent chooses action by random in the first round. In the following rounds, reinforcement learning module is used, and an action with the highest predicted value (\( Q \)) is chosen. Set of possible actions contains ship allocation triples: \( A = \{(h, d, c)\} \) such that \( h, d, c \in \{0\%, 25\%, 50\%, 75\%, 100\%\} \), \( d+c=1 \). Set of possible states \( X = \{(dc, cc)\} \), where \( dc \in \{1, 2, \ldots, 25\} \) represent catch in a deep-sea area, and \( cc \in \{1, 2, \ldots, 15\} \) represents catch in a coastal area in the previous round. Therefore Problem is a pair \((dc, cc)\) and Answer is a triple \((h, d, c)\). The Training data consists of a pair \((dc', cc')\), which is a catch in the current round, and a reward that is equal to the income (money earned by fishing decreased by ship maintenance costs). Learning algorithm applied is the Q-Learning algorithm. In the current implementation, \( Q \) function has tabular representation.
At the beginning Q is initialized as a constant function 0. To provide sufficient exploration, in a game number $g$ a random action is chosen with probability $1/g$ instead of using Q function (all actions have the same probability then).

Generally, supervised learning allows to generate an approximation of a function $f: D \rightarrow C$ from labelled examples, which consist of pairs of arguments and function values. This approximation is called a hypothesis $h$. If the size of the set $C$ is small (like in this application), we call $C$ a set of classes, and hypothesis is called a classifier.

Elements of $D$ (called examples) are described by set of attributes $Attr=(a_1, a_2, \ldots, a_n)$, where $a_i: D \rightarrow V_i$. Therefore $x^{Attr}=(a_1(x), a_2(x), \ldots, a_n(x))$ is used instead of $x$.

In a general case, supervised learning module have Training data in a form of a set $\{(x^{Attr}, f(x))\}$, and generates hypothesis $h$. Problem is a $x^{Attr}$, and the Answer is $h(x^{Attr})$.

The simplest solution in our system would be to learn a classifier, in which classes represent allocation actions, and attributes describe a current situation. Unfortunately, there is no a direct way, in which agent could prepare a training data for such a classifier. Another problem is a big size of $C$ in such a solution. To overcome this problem the following workaround is used. Thank to comparison of income of all agents after action execution, the learning agent has information about quality of actions executed in the current situation and can use it for training. Learning module is used to classify action in the given situation as good or bad. Such classifier may be used to give ranks to action candidates.

More precisely, the Problem is defined as a five-tuple: $(dc, cc, h, d, c)$, it consists of catch in the both areas during the previous round and a ship allocation action parameters. The Answer is an integer, which represents the given allocation action rating. The agent collects ratings for all generated allocation action candidates and for execution chooses the action with the highest rating.

Training examples are generated from agent observations. Every round the learning agent stores ship allocations of all agents, and the fish catch in the previous round. The action of an agent with the highest income is labelled as good, and the action of an agent with the lowest income is labelled as bad. If in some round all agents get the same income, none action is classified, and as a consequence, none of them is used in learning. Training data consists of the following pairs: $((dc, cc, h, d, c), q)$, where $q$ is equal to good or bad. At the end of each game the agent uses training examples, which were generated during all games played so far, to learn a new classifier, which is used in the next game.

Rating $v$ of the action $a$ is calculated according to the formula:

$$v(a) = \alpha \text{good}(a) - \text{bad}(a),$$

where good($a$) and bad($a$) are numbers of rules, which match the action and current environment parameters, with consequence good and bad, respectively, and $\alpha$ is a weight representing a relative importance of rules with consequence good.

Learning algorithm used for supervised learning is AQ algorithm. More specifically, AQ21 program is executed. It is the last implementation of the AQ algorithm (Wojtusiak, 2004). This algorithm was developed by Ryszard Michalski (Michalski & Larson, 1975). Hypothesis is represented by a set of attributional rules, which have tests on attribute values in the premise part, and a class in the conclusion. Rules are generated using sequential covering: the best rule (e.g. giving an appropriate answer for the most examples) is constructed by a beam search, examples covered by this rule are eliminated from a training set, and the procedure repeats. What is important for this system, the rule set produced is not ordered.
(rules can be applied in any order). Therefore we can simply count the number of matching rules during action rating calculation.

### 3.4 Implementation

The software used in experiments is written in Prolog, using Prologix compiler (Majumdar & Tarau, 2004). Every agent is a separate process. It can be executed on a separate machine. Agents communicate with the environment using Linda blackboard.

*Prologix* is an extension of *BinProlog* that has many powerful knowledge-based extensions (e.g. agent language LOT, Conceptual Graphs and KIF support).

Table 1 contains predicates, its argument domains, and predicate descriptions, which are used in supervised learning module knowledge base. They appear in the training data and rules induced by the AQ algorithm.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Argument domains</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate(R)</td>
<td>good, bad</td>
<td>Rating of the allocation strategy</td>
</tr>
<tr>
<td>harbor(N)</td>
<td>100%, 75%, 50%, 25%, 0</td>
<td>Fraction of ships left in a harbour</td>
</tr>
<tr>
<td>alloc(A)</td>
<td>100%-0%, 75%-25%, 50%-50%, 25%-75%, 0%-100%</td>
<td>Allocation: ship fraction sent to a deep sea, and ship fraction sent to a coastal area</td>
</tr>
<tr>
<td>prevCatchDeep(D)</td>
<td>integer numbers</td>
<td>Number of fish caught by every ship on a deep sea</td>
</tr>
<tr>
<td>prevCatchCoastal(C)</td>
<td>integer numbers</td>
<td>Number of fish caught by every ship on a coastal area</td>
</tr>
</tbody>
</table>

Table 1. Predicates and its arguments used in the rules build in the supervised learning module

### 4. Experimental results

To compare reinforcement and supervised learning strategies during controlling ship allocation action of agents, four experiments were performed. Four agents took part in every experiment. Each experiment consisted of 20 repetitions of the sequences of ten games. Knowledge of learning agents was passing along consecutive games in one sequence, but was cleared between sequences. The performance of agents was measured as a balance at the end of every game. In the figures we see average values of balances from repetitions.

In the first experiment there were three random agents and one reinforcement learning agent (with $\gamma=1$ and $\beta=0.1$). Results are presented in Fig.2-(a).

In the second series there were three random agents and one supervised rule learning agent (with weight $\alpha=1$). The performance of these agents is presented in Fig.2-(b).

In the third experiment both types of learning agents (with parameters as above) and two random agents were examined. Results are presented in Fig. 3-(a). Stability of performance (measured by the standard deviation of the balance at the end of games) is presented in Fig. 4.

In the fourth series one supervised learning ($\alpha=1$), one predicting and two random agents were used. The performance of agents is presented in Fig.3-(b).

In all experiments average balance of both types of learning agents increases with the agent's experience, while the performance of the predicting and random agents decreases
Fig. 2. Comparison of performance of reinforcement learning agent (RLA), supervised rule learning agent (SLA) and agents using random strategy of ship allocation (RA1, RA2, RA3). Horizontal axis represents number of a game in the sequence. Vertical axis represents an average balance of an agent at the end of the game.

slightly (because of the learning agents competition). Reinforcement learning agent was worse than a rule learning agent, but tuning of its parameters and taking into account actions of other agents during learning should increase its performance. Also change of the
Q function representation (e.g. into neural-network based approximator) should improve the performance. Results of reinforcement learning agent were also less stable (had higher standard deviation).

Fig. 3. Comparison of performance of reinforcement learning agent (RLA), supervised rule learning agent (SLA), agents using random strategy of ship allocation (RA1, RA2), and predicting agent (PA). Horizontal axis represents number of a game in the sequence. Vertical axis represents an average balance of an agent at the end of the game.
Experimental results show that the supervised rule learning agent performance increases rapidly at the beginning of the learning process, when generated rules are used instead of a random choice. Next it increases slowly, because new examples do not contain any significant new knowledge. The performance stabilizes at the end of the process.

As we can see in Fig.3-(b), the predicting agent performs better then the supervised learning agent. It suggests, that there is a space for improvement of the learning method. Further research is necessary to check if it is possible to learn such a good strategy.

Examples of rules learned are presented in Fig. 5. They are in the form of Prolog clauses. Capital letters represent variables that can be unified with any value. Predicate member checks if its first argument belongs to the list that is a second argument. It is used to represent an internal disjunction (expression of the form $x = v_1 \text{ or } v_2 \text{ or } \ldots \text{ or } v_n$). Remaining predicates are presented in Table 1. These rules can be interpreted in the following way:

Clause (a): it is a bad decision to keep at a harbour 25, 50, or 75 percent of ships if the previous catch at a deep-sea is greater or equal to 16, and the previous catch at a coastal area is 10.

Clause (b): it is a good decision to send 100% ships to a deep sea or 75% to a deep sea and 25% to a coastal area if previous catch at a deep-sea is greater or equal to 18, and smaller or equal to 21, and previous catch at a coastal area is smaller or equal to 10.

5. Conclusion

As we can see, both learning algorithms can be applied for learning resource allocation in a multi-agent system. Their performance is much better then a random strategy, but there is still a space for improvement.
Supervised Rule Learning and Reinforcement Learning in A Multi-Agent System for the Fish Banks Game

Fig. 5. Examples of rules in the form of Prolog clauses learned by the supervised learning agent

This work shows also that it is possible to use supervised learning method in a case, in which there is no direct training data. It is enough that agent has some qualitative information about action that was executed in a given state.

Both of the considered learning strategies have some advantages and disadvantages. These two methods use different knowledge representation. Reinforcement learning uses the action value function, which is difficult to analyze especially in a case of a large domain. Rules are usually much easier to interpret (unless there are too many of them). Therefore, if learned knowledge is analyzed by a human, rule induction seems to be a better choice.

A disadvantage of reinforcement learning is necessity of tuning its parameters ($\gamma$, $\beta$, and exploration method). The choice has a high impact on the results. What is more, due to necessary exploration, the algorithm's performance is less stable.

On the other hand, reinforcement learning works well even if the reward is delayed. Additionally, it does not need information about other agents' actions. Hence it is more universal.

6. Further research

Currently the architecture of agents supports centralized learning only. In the future it should be extended to cover distributed learning (communication and cooperation during learning). Future works will concern applying other learning algorithms and also other strategies (e.g. unsupervised learning).

Additionally, agents with more then one learning module for different aspects of their activity should be studied and the possibility of interaction between learning modules in the same agent should be examined.

7. Acknowledgements

The author is grateful to Arun Majumdar, Vivomind Intelligence Inc. for providing Prologix system (used for implementation), and for help with using it, Janusz Wojtusiak, MLI Laboratory for AQ21 software and assistance, and last but not least Jaroslaw Koźlak, AGH University of Science and Technology for help with the Fish Bank Game.

8. References


Michalski, R.S., Larson, J. (1975). Aqval/1 (aq7) user’s guide and program description. Technical Report 731, Department of Computer Science, University of Illinois, Urbana, USA


Even since computers were invented, many researchers have been trying to understand how human beings learn and many interesting paradigms and approaches towards emulating human learning abilities have been proposed. The ability of learning is one of the central features of human intelligence, which makes it an important ingredient in both traditional Artificial Intelligence (AI) and emerging Cognitive Science. Machine Learning (ML) draws upon ideas from a diverse set of disciplines, including AI, Probability and Statistics, Computational Complexity, Information Theory, Psychology and Neurobiology, Control Theory and Philosophy. ML involves broad topics including Fuzzy Logic, Neural Networks (NNs), Evolutionary Algorithms (EAs), Probability and Statistics, Decision Trees, etc. Real-world applications of ML are widespread such as Pattern Recognition, Data Mining, Gaming, Bio-science, Telecommunications, Control and Robotics applications. This book reports the latest developments and futuristic trends in ML.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:
